ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р ИСО/МЭК 8825-3— 2016

Информационная технология

ПРАВИЛА КОДИРОВАНИЯ АСН.1

Часть 3

Спецификация нотации контроля кодирования (ECN)

(ISO/IEC 8825-3:2008, IDT)

Издание официальное

Москва
Стандартинформ
2016

Предисловие

1 ПОДГОТОВЛЕН Федеральным государственным унитарным предприятием Государственный научно-исследовательский и конструкторско-технологический институт «ТЕСТ» (ФГУП ГосНИИ «ТЕСТ»), Обществом с ограниченной ответственностью «Информационно-аналитический центр» (ООО ИАВЦ) на основе собственного перевода на русский язык англоязычной версии международного стандарта, указанного в пункте 4

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 22 «Информационные технологии»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 7 ноября 2016 г. № 1597-ст

4 Настоящий стандарт идентичен международному стандарту ИСО/МЭК 8825-3:2008 «Информационная технология. Правила кодирования АСН.1. Часть 3. Спецификация нотации контроля кодирования (ECN)» [ISO/IEC 8825-3:2008 «Information technology — ACH.1 encoding rules — Part 3: Specification of Encoding Control Notation (ECN)», IDT].

При применении настоящего стандарта рекомендуется использовать вместо ссылочных международных стандартов соответствующие им национальные стандарты, сведения о которых приведены в дополнительном приложении ДА

5 ВВЕДЕН ВПЕРВЫЕ

Правила применения настоящего стандарта установлены в статье 26 Федерального закона от 29 июня 2015 г. № 162-ФЗ «О Стандартизации в Российской Федерации». Информация об изменениях к настоящему стандарту публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ежемесячном информационном указателе «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (www.gost.ru)

© Стандартинформ, 2016

Настоящий стандарт не может быть полностью или частично воспроизведен, тиражирован и распространен в качестве официального издания без разрешения Федерального агентства по техническому регулированию и метрологии

Содержание

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная технология

ПРАВИЛА КОДИРОВАНИЯ АСН.1

Часть 3

Спецификация нотации контроля кодирования (ECN)

Information technology. ACH.1 encoding rules. Part 3. Specification of Encoding Control Notation (ECN)

Дата введения — 2017—11—01

1 Область применения

Настоящий стандарт определяет нотацию для спецификации кодирования типов АСН.1 или частей типов.

В настоящем стандарте предусмотрено несколько механизмов такой спецификации, в том числе:

- прямая спецификация кодирования с помощью стандартизованной нотации;
- спецификация кодирования путем ссылки на стандартизованные правила кодирования;
- спецификация кодирования типа АСН.1 путем ссылки на некоторую структуру кодирования;
- спецификация кодирования с использованием не-ECN нотации.

В настоящем стандарте предусмотрены также средства для связи спецификаций кодирования с определениями типов, к которым кодирование должно применяться.

ECN в данный момент не предоставляет поддержку для спецификации, использующей OID тип международного идентификатора ресурсов или относительный OID тип международного идентификатора (см. МСЭ-Т рек. X.680/ ИСО/МЭК 8824-1), и они не указаны далее в настоящем стандарте.

2 Нормативные ссылки

В настоящем стандарте использованы нормативные ссылки на следующие международные стандарты.

2.1 Идентичные международные стандарты

ISO/IEC 9834-1:2005 Information technology — Open Systems Interconnection — Procedures for the operation of OSI Registration Authorities: General procedures and top arcs of the ASN.1 Object Identifier tree (ИСО/МЭК 9834-1:2005 Информационные технологии. Взаимосвязь открытых систем. Процедуры для работы регистрационных органов в системе OSI. Часть 1. Общие процедуры и высшие разряды дерева идентификаторов объекта ASN.1)

ISO/IEC 8824-1:2008 Information technology — Abstract Syntax Notation One (ACH.1): Specification of basic notation [ИСО/МЭК 8824-1:2008 Информационная технология. Абстрактно-синтаксическая нотация 1 (АСН.1). Часть 1. Спецификация базовой нотации]

ISO/IEC 8824-2:2008 Information technology — Abstract Syntax Notation One (ACH.1): Information object specification [ИСО/МЭК 8824-2:2008 Информационная технология. Абстрактно-синтаксическая нотация 1 (АСН.1). Часть 2. Спецификация информационных объектов]

Издание официальное

ISO/IEC 8824-3:2008 Information technology — Abstract Syntax Notation One (ACH.1): Constraint specification [ИСО/МЭК 8824-3:2008 Информационная технология. Абстрактно-синтаксическая нотация 1 (ACH.1). Часть 3. Спецификация ограничений]

ISO/IEC 8824-4:2008 Information technology — Abstract Syntax Notation One (ACH.1): Parameterization of ACH.1 specifications [ИСО/МЭК 8824-4:2008 Информационная технология. Абстрактно-синтаксическая нотация 1 (ACH.1). Часть 4. Параметризация спецификаций ASN-1]

ISO/IEC 8825-1:2008 Information technology — ACH.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER) [ИСО/МЭК 8825-1:2008 Информационная технология. Правила кодирования ACH.1. Часть 1. Спецификация базовых правил кодирования (BER), канонических правил кодирования (CER) и выделенных правил кодирования (DER)]

ISO/IEC 8825-2:2008 Information technology — ACH.1 encoding rules: Specification of Packed Encoding Rules (PER) [ИСО/МЭК 8825-2:2008 Информационная технология. Правила кодирования ACH.1. Часть 2. Спецификация уплотненных правил кодирования (PER)]

Примечания

1 Независимо от даты публикации ИСО вышеприведенные публикации обычно называют «ACH.1:2008».

2 Вышеприведенные ссылки следует понимать как ссылки на указанные международные стандарты вместе со всеми опубликованными поправками и техническими исправлениями.

2.2 Дополнительные ссылки

ИСО/МЭК 10646:2003 Информационная технология. Универсальный набор многооктетных кодированных знаков (UCS) [ISO/IEC 10646:2003 Information technology — Universal Multiple-Octet Coded Character Set (UCS)]

Примечание — Вышеприведенную ссылку следует понимать как ссылку на стандарт ИСО/МЭК 10646 вместе со всеми опубликованными поправками и техническими исправлениями.

3 Термины и определения

В настоящем стандарте применены следующие термины с соответствующими определениями.

3.1 Определения терминов по ACH.1

В настоящем стандарте применены термины по ИСО/МЭК 8824-1, ИСО/МЭК 8824-2, ИСО/МЭК 8824-3, ИСО/МЭК 8824-4, ИСО/МЭК 8825-1, ИСО/МЭК 8825-2.

3.2 Определения терминов, специфичных для ECN

3.2.1 **точка выравнивания** (alignment point): Точка в кодовой последовательности (обычно ее начало), которая служит в качестве эталонной точки, когда спецификация кодирования требует выравнивания (фазирования) до некоторой границы.

3.2.2 **вспомогательное поле** (auxiliairy field): Поле структуры «замена» (которая добавляется к спецификации ECN), значение которой устанавливается прямо кодером без использования какого-либо абстрактного значения, выдаваемого приложением.

Примечание — Примером вспомогательного поля является определитель длины при кодировании целых чисел или при повторении.

3.2.3 **битовое поле** (bit-field): Смежные биты или октеты в кодовой последовательности, которые декодируются как целое и которые либо представляют абстрактное значение, либо содержат информацию (например, определитель длины некоторого другого поля, см. 3.2.31), необходимую для успешного декодирования, либо представляют то и другое.

Примечание — «То и другое» часто встречается в традиционных протоколах.

3.2.4 **класс битового поля** (bit-field class): Класс кодирования, объекты которого определяют кодирование абстрактных значений (некоторого типа ACH.1) в биты.

Примечание — Другие классы кодирования относятся к более общим процедурам кодирования, таким как процедуры, требующие определения конца повторений кодирования некоторого класса битового поля или определения присутствия альтернативного кодирования битового поля из некоторого набора.

3.2.5 **условие границ** (bound condition): Условие наличия границ для поля целых чисел (независимо от того, разрешают они отрицательные значения или нет), которое, если оно удовлетворено, означает, что должны применяться определенные правила кодирования.

3.2.6 **определитель выбора** (choice determinant): Битовое поле, которое определяет, какие из нескольких возможных кодовых последовательностей (каждая из которых представляет разные абстрактные значения) имеются в некотором другом битовом поле.

3.2.7 **комбинированный набор объектов кодирования** (combined encoding object set): Временный набор объектов кодирования, образованный путем комбинации двух наборов объектов кодирования с целью применения этих кодирований.

3.2.8 **условия кодирования** (conditional encoding): Кодирование, которое должно применяться только в случае, когда выполняется некоторое указанное условие границ или условие диапазона размеров.

Примечание — Условие может быть условием границ, или условием диапазона размера, или другими более сложными условиями.

3.2.9 **объемлющий тип** (containing type): Тип АСН.1 (или поле структуры кодирования), у которого ограничение на содержание применяется к значениям этого типа (или к значениям, связанным с этим полем структуры кодирования).

Примечание — Типы АСН.1, к которым может применяться ограничение на содержание (с помощью **CONTAINING/ENCODED BY**), — это типы «цепочка битов» и «цепочка октетов».

3.2.10 **текущая точка применения** (current application point): Точка в структуре кодирования, в которой применяется комбинированный набор объектов кодирования.

3.2.11 **дифференциальное кодирование-декодирование** (differential encoding-decoding): Спецификация правил для кодера, требующих приема кодовых последовательностей, которые не могут вырабатываться кодером, соответствующим текущей спецификации.

Примечание — Дифференциальное кодирование-декодирование поддерживает спецификацию декодирования декодером (соответствующим какой-либо начальной версии стандарта), которая предназначена для создания ему возможности успешно декодировать коды, образованные последующей версией этого стандарта. Это иногда называют «поддержкой растяжимости».

3.2.12 **класс кодирования** (encoding class): Набор всех возможных кодовых последовательностей для конкретной части процедур, необходимых для выполнения кодирования или декодирования некоторого типа АСН.1.

Примечание — Классы кодирования определены для кодирования простейших (примитивных) типов АСН.1, а также определены для процедур, связанных с нотацией тегов АСН.1, использованием **OPTIONAL**, и для конструкторов кодирования.

3.2.13 **категория классов кодирования** (encoding class category): Классы кодирования с некоторыми общими характеристиками.

Примечание — Примерами являются категории «целочисленная», «булева» и «конкатенация».

3.2.14 **конструктор кодирования** (encoding constructor): Класс кодирования, объекты кодирования которого определяют процедуры кодирования для частей комбинирования, выбора и повторения (примерами являются классы #ALTERNATIVES, #CHOICE, #CONCATENATION, #SEQUENCE и др.).

3.2.15 **модули определения кодирования** (EDM) (Encoding Definition Modules): Модули, которые определяют кодовые последовательности для применения в модуле компоновки кодирования.

3.2.16 **модуль компоновки кодирования** (ELM) (Encoding Link Module): Модуль (уникальный для любого заданного применения), который применяет кодовые последовательности к типам АСН.1.

3.2.17 **объект кодирования** (encoding object): Спецификация некоторой части процедур, необходимой для выполнения кодирования или декодирования того или иного типа АСН.1.

Примечание — Объекты кодирования могут определять кодирование простейших типов АСН.1, а также определять процедуры, связанные с нотацией тегов АСН.1, использованием **OPTIONAL** и с конструкторами кодирования.

3.2.18 **набор объектов кодирования** (encoding object set): Набор из объектов кодирования.

Примечание — Набор объектов кодирования обычно используется в модуле компоновки кодирования для определения кодирования всех типов верхнего уровня, используемых в каком-либо применении.

3.2.19 **признак кодирования** (encoding property): Отрезок информации, используемый для определения кодирования с помощью нотации, описанной в разделах 23–25.

3.2.20 **пространство кодирования** (encoding space): Число битов (или октетов, слов, или других единиц), используемых для кодирования абстрактного значения в битовое поле (см. 9.21.5).

3.2.21 **структура кодирования** (encoding structure): Структура для кодирования, определенная либо из структуры определения типа АСН.1, либо в EDM с помощью классов битового поля и конструкторов кодирования.

Примечания

1 Использование структуры кодирования является лишь одним (но важным) механизмом из нескольких, обеспечиваемых нотацией управления кодированием для определения кодовых последовательностей для типов АСН.1.

2 Определение структуры кодирования является также определением соответствующего класса кодирования.

3.2.22 **явно генерируемая структура кодирования** (explicitly generated encoding structure): Структура кодирования, выделенная из неявно генерируемой структуры кодирования путем использования раздела переименований в EDM.

3.2.23 **растяжимость** (extensibility): Положения в ранней версии стандарта, которые рассчитаны на облегчение взаимодействия реализаций этой ранней версии с ожидаемыми реализациями последующей версии этого стандарта.

3.2.24 **полностью определенное имя** (fully-qualified name): Ссылка на класс кодирования, объект кодирования или набор объектов кодирования, которая содержит либо имя модуля EDM, в котором этот класс, объект или набор объектов кодирования определен, либо (в случае неявно генерируемого класса кодирования) имя модуля АСН.1, в котором он был генерирован (см. также 3.2.43).

Примечание — Полностью определенное имя (см. продукцию «ExternalEncodingClassReference» в 10.6) должно использоваться в теле модуля, если классом кодирования является неявно генерируемая структура кодирования, чье имя совпадает с зарезервированным именем класса, либо, если имя используется отдельно, вызовет неоднозначность из-за многих импортов классов с этим именем (см. А.1, 13.16).

3.2.25 **генерируемая структура кодирования** (generated encoding structure): Неявно или явно генерируемая структура кодирования, целью которой является определение кодовых последовательностей соответствующего типа АСН.1 путем применения кодовых последовательностей в ELM.

3.2.26 **руководитель или руководящее указание** (governor): Часть спецификации ECN, которая определяет синтаксическую форму (или семантику) какой-либо другой части спецификации ECN.

Примечание — Руководитель является ссылкой на класс кодирования и указывает синтаксис, который будет использован для определения объекта кодирования (этого класса). Это понятие такое же, как понятие ссылки на тип в АСН.1, играющей роль руководителя для нотации значений АСН.1.

3.2.27 **набор значений описателя** (handle value set): Определенный набор всех возможных значений идентификационного описателя, присущий объекту кодирования.

3.2.28 **идентификационный описатель** (identification handle): Часть кодирования, которая служит для различения способов кодирования одного класса кодирования от способов кодирования других классов кодирования.

Примечание — В базовых правилах кодирования АСН.1 используются теги для обеспечения идентифицирующих описателей в кодовых последовательностях BER.

3.2.29 **неявно генерируемая структура кодирования** (implicitly generated encoding structure): Структура кодирования, которая генерируется и экспортируется неявно, когда тип определен в модуле АСН.1.

3.2.30 **точка начального применения** (initial application point): Точка в структуре кодирования, в которой заданный комбинированный набор объектов кодирования применяется в первый раз (в ELM или в модулях EDM).

3.2.31 **определитель длины** (length determinant): Битовое поле, которое определяет длину другого битового поля.

3.2.32 **отрицательное целочисленное значение** (negative integer value): Значение, меньшее нуля.

3.2.33 **неотрицательное целочисленное значение** (non-negative integer value): Значение, большее нуля или равное нулю.

3.2.34 **неположительное целочисленное значение** (non-positive integer value): Значение, меньшее нуля или равное нулю.

3.2.35 **факультативное битовое поле** (optional bit-field): Битовое поле, которое иногда вставляется (для кодирования абстрактного значения), а иногда опускается.

3.2.36 **положительное целочисленное значение** (positive integer value): Значение, большее нуля.

3.2.37 **определитель наличия** (presence determinant): Битовое поле, которое указывает, присутствует ли факультативное битовое поле.

3.2.38 **простейший (примитивный) класс** (primitive class): Класс кодирования, который не является структурой кодирования и не может быть отнесен к другому классу (см. 16.1.14).

3.2.39 **рекурсивное определение (справочного имени)** [recursive definition (of a reference name)]: Справочное имя, для которого вычисление этого справочного имени или руководителя для определения справочного имени требует вычисления исходного справочного имени.

Примечание — Рекурсивное определение класса кодирования (включая структуру кодирования) или объекта кодирования разрешается (но см. 17.1.4). Рекурсивное определение набора объектов кодирования запрещается в 18.1.3.

3.2.40 **рекурсивная реализация (параметризованного справочного имени)** [recursive instantiation (of a parameterized reference name)]: Создание справочного имени, для которого вычисление реальных параметров требует вычисления исходного справочного имени.

Примечание — Рекурсивное определение класса кодирования (включая структуру кодирования) или объекта кодирования разрешается (но см. 17.1.4). Рекурсивное определение набора объектов кодирования запрещается в 18.1.3.

3.2.41 **структура замены** (replacement structure): Параметризованная структура, используемая для замены некоторых или всех частей конструкции перед кодированием этой конструкции.

3.2.42 **саморазграничивающее кодирование** (self-delimiting encoding): Кодирование набора абстрактных значений, при котором отсутствует такое абстрактное значение, которое имеет кодовую последовательность, являющуюся начальной субцепочкой кодовой последовательности другого абстрактного значения в этом наборе.

Примечание — Это охватывает только кодовые последовательности с фиксированной длиной для ограниченного целого числа, а также кодовые последовательности, описанные обобщенно как «кодовые последовательности Хаффмана» (см. приложение Е).

3.2.43 **простое справочное имя** (simple reference name): Ссылка на класс, объект или набор объектов кодирования, которая не содержит ни имени модуля EDM, в котором определен этот класс, объект или набор объектов кодирования, ни (в случае неявного генерируемого класса кодирования) имени модуля АСН.1, в котором он был генерирован.

Примечание — Простое справочное имя может использоваться лишь в случае, когда ссылка на класс кодирования недвусмысленна, в остальных случаях в теле модуля должно использоваться полностью определенное имя (см. 3.2.24).

3.2.44 **условие диапазона размера** (size range condition): Условие по наличию ограничений на действующий размер поля «цепочка» или «повторение» (а также включение нуля в ограничение и/или разрешение нескольких размеров), которое, если оно удовлетворяется, означает, что должны применяться определенные правила кодирования.

3.2.45 **руководитель источника** (или класс источника) [source governor (or source class)]: Руководитель, который определяет нотацию для указанных абстрактных значений, связанных с классом источника, при отображении их в класс цели.

3.2.46 **начальный указатель** (start pointer): Вспомогательное поле, указывающее на присутствие или отсутствие какого-либо факультативного битового поля, а в случае присутствия также указывающее смещение от текущего положения до этого битового поля.

3.2.47 **руководитель цели** (или класс цели) [target governor (or target class)]: Руководитель, который определяет нотацию для указанных абстрактных значений, связанных с классом цели, при отображении в них из класса источника.

3.2.48 **тип (типы) верхнего уровня** [top-level type(s)]: Такой тип (типы) АСН.1 в приложении, который используется этим приложением способом, отличающимся от способа для определения компонентов других типов АСН.1.

Примечания

1 Типы верхнего уровня могут также использоваться (но обычно не используются) в качестве компонентов других типов АСН.1.

2 Типы верхнего уровня иногда называют «сообщениями приложения» или «протокольными блоками данных (PDU)». Эти типы обычно специально обрабатываются инструментами, так как они образуют в языке программирования верхний уровень тех структур данных, которые предоставляются приложению.

3.2.49 **преобразователи** (transforms): Объекты кодирования класса **#TRANSFORM**, которые указывают, что кодирование абстрактных значений, связанных с некоторым классом (или с композициями преобразования — см. 3.2.50), должно быть кодированием других абстрактных значений, связанных с тем же или с другим классом (или композициями преобразования).

Примечание — Преобразователи могут использоваться, например, для указания простых арифметических операций над значениями целых чисел либо для отображения значений целых чисел в цепочки знаков или цепочки битов.

3.2.50 **композиции преобразования** (transform composites): Упорядоченный список элементов, которые сами могут быть источником или результатом преобразователей.

Примечание — Необходимо, чтобы все такие элементы имели одну и ту же классификацию (см. 9.18.2).

3.2.51 **кодирование значения** (value encoding): Способ использования пространства кодирования для представления абстрактного значения (см. 9.21.5).

4 Сокращения

В настоящем стандарте применены следующие сокращения:
ACH.1 — абстрактно-синтаксическая нотация 1;
BCD — двоично-кодированное десятичное число;
BER — базовые правила кодирования ACH.1;
CER — канонические правила кодирования ACH.1;
DER — выделенные правила кодирования ACH.1;
ECN — нотация управления кодированием для ACH.1;
EDM — модуль определения кодирования;
ELM — модуль компоновки кодирования;
PDU — протокольный блок данных;
PER — уплотненные правила кодирования ACH.1.

5 Определение синтаксиса ECN

5.1 В настоящем стандарте применяется соглашение о нотации, определенное в разделе 5 ИСО МЭК 8824-1.

5.2 В настоящем стандарте применяется нотация для классов информационных объектов, определенная в ИСО/МЭК 8824-2 и уточненная в приложении В.

5.3 В настоящем стандарте приведены ссылки на продукции, определенные в ИСО/МЭК 8824-1 (с уточнениями из приложения А), ИСО/МЭК 8824-2 (с уточнениями из приложения В) и ИСО/МЭК 8824-4 (с уточнениями из приложения С).

6 Соглашения о кодировании и нотация

6.1 В настоящем стандарте определены значения каждого объекта в кодовой последовательности при помощи терминов «бит старшего порядка» и «бит младшего порядка».

Примечание — В спецификациях нижнего уровня используется такая же нотация для определения порядка передачи битов в последовательной линии или для прикрепления битов к параллельным каналам.

6.2 В настоящем стандарте биты в октете пронумерованы с 8 до 1, причем бит 8 является «битом старшего порядка», а бит 1 — «битом младшего порядка».

6.3 В настоящем стандарте кодирование определяется в виде цепочки битов, которая начинается «начальным битом» и заканчивается «конечным битом». При передаче первые восемь битов этой цепочки битов, начинающейся с «начального бита», помещаются в первый передаваемый октет, в котором «начальный бит» будет «битом старшего порядка» у этого октета. Следующие восемь битов помещаются в следующий октет и т. д. Если в кодируемой последовательности число битов

не кратно восьми, то оставшиеся биты передаются так, как будто имеются 8 битов перед последующим октетом.

Примечание — Полная кодируемая последовательность ECN не обязательно всегда кратна восьми битам, но спецификация ECN может указывать на добавление заполняющих битов для обеспечения этого свойства.

6.4 В настоящем стандарте на рисунках «начальный бит» всегда показан слева.

7 Набор знаков ECN

7.1 Использование термина «знак» в настоящем стандарте относится к знакам, определенным в ИСО/МЭК 10646, причем полная поддержка всех возможных спецификаций ECN может потребовать представления всех этих знаков.

7.2 В спецификациях ECN, за исключением комментариев (определенных в ИСО/МЭК 8824-1, подраздел 12.6), не-ECN определений объектов кодирования (см. 17.8) и значений цепочек (строк) знаков, используются только знаки, перечисленные в таблице 1.

7.3 Лексические элементы, определенные в разделе 8, содержат последовательность знаков, перечисленных в таблице 1.

Примечание — Дополнительные ограничения на разрешенные знаки для каждого лексического элемента указываются в разделе 8.

Таблица 1 — Знаки ECN

0 до 9	(ЦИФРА НУЛЬ ... ЦИФРА 9)
A до Z	(ЛАТИНСКИЕ ПРОПИСНЫЕ БУКВЫ ОТ A ДО Z)
a до z	(ЛАТИНСКИЕ СТРОЧНЫЕ БУКВЫ ОТ A ДО Z)
"	(КАВЫЧКИ)
#	(ЗНАК НОМЕРА)
&	(СОЮЗ «И»)
'	(АПОСТРОФ)
(	(ЛЕВАЯ КРУГЛАЯ СКОБКА)
)	(ПРАВАЯ КРУГЛАЯ СКОБКА)
,	(ЗАПЯТАЯ)
.	(ТОЧКА)
—	(ДЕФИС, МИНУС)
:	(ДВОЕТОЧИЕ)
;	(ТОЧКА С ЗАПЯТОЙ)
<	(ЗНАК «МЕНЬШЕ, ЧЕМ»)
=	(ЗНАК РАВЕНСТВА)
>	(ЗНАК «БОЛЬШЕ, ЧЕМ»)
{	(ЛЕВАЯ ФИГУРНАЯ СКОБКА)
|	(ВЕРТИКАЛЬНАЯ ЧЕРТА)
}	(ПРАВАЯ ФИГУРНАЯ СКОБКА)

7.4 Типографский шрифт, размер, цвет, яркость и другие характеристики отображения не имеют значения.

7.5 Прописные и строчные буквы должны считаться разными знаками.

8 Лексические элементы ECN

В дополнение к лексическим элементам, определенным в ИСО/МЭК 8824-1 [раздел 12], в настоящем стандарте использованы лексические элементы, которые определяются в последующих подразделах. В настоящем разделе применяются общие правила, определенные в ИСО/МЭК 8824-1, подраздел 12.1.

Примечание — В приложении G перечислены все лексические элементы и все продукции, использованные в настоящем стандарте, с указанием тех, которые определены в ИСО/МЭК 8824-1, ИСО/МЭК 8824-2 и ИСО/МЭК 8824-4.

8.1 Справочные имена объектов кодирования

Имя элемента — encodingobjectreference

Элемент «encodingobjectreference» содержит последовательность знаков, указанную для «valuereference» в ИСО/МЭК 8824-1, подраздел 12.4. При анализе примера использования этой нотации элемент «encodingobjectreference» различается от «identifier» согласно контексту, в котором он появляется.

8.2 Справочные имена набора объектов кодирования

Имя элемента — encodingobjectsetreference

Элемент «encodingobjectsetreference» содержит последовательность знаков, указанную для «typereference» в ИСО/МЭК 8824-1, подраздел 11.2. Она не должна быть какой-либо последовательностью знаков из перечисленных в 8.4.

8.3 Справочные имена классов кодирования

Имя элемента — encodingclassreference

Элемент «encodingclassreference» содержит знак «#», за которым следует последовательность знаков, указанная для «typereference» в ИСО/МЭК 8824-1, подраздел 11.2. Она не должна быть какой-либо последовательностью знаков из перечисленных в 8.5, за исключением применения в списке импортов EDM (см. в ИСО/МЭК 8824-1, пункт 13.20 с учетом изменений из А.1) или в «ExternalEncodingClassReference» (см. примечание в 14.11).

8.4 Элементы «зарезервированное слово»

Имена элементов «зарезервированное слово»:

ALL	**FIELDS**	**PER-BASIC-UNALIGNED**
AS	**FROM**	**PER-CANONICAL-ALIGNED**
BEGIN	**GENERATES**	**PER-CANONICAL-UNALIGNED**
BER	**IF**	**PLUS-INFINITY**
BITS	**IMPORTS**	**REFERENCE**
BY	**IN**	**REMAINDER**
CER	**LINK-DEFINITIONS**	**RENAMES**
COMPLETED	**MAPPING**	**SIZE**
DECODE	**MAX**	**STRUCTURE**
DER	**MIN**	**STRUCTURED**
DISTRIBUTION	**MINUS-INFINITY**	**TO**
ENCODE	**NON-ECN-BEGIN**	**TRANSFORMS**
ENCODING-CLASS	**NON-ECN-END**	**TRUE**
ENCODE-DECODE	**NULL**	**UNION**
ENCODING-DEFINITIONS	**OPTIONAL-ENCODING**	**USE**
END	**OPTIONS**	**USE-SET**
EXCEPT	**ORDERED**	**VALUES**
EXPORTS	**OUTER**	**WITH**
FALSE	**PER-BASIC-ALIGNED**	

Элементы с вышеприведенными именами должны содержать указанную последовательность знаков в имени.

Примечание — Слова (см. в ИСО/МЭК 8824-2, подраздел 7.9), используемые в определении классов кодирования (в операторе «WITH SYNTAX») в разделе 23, не являются зарезервированными словами (см. также В.14).

8.5 Элементы «зарезервированное имя класса кодирования»

Имена элементов «зарезервированное имя класса кодирования»:

#ALTERNATIVES	**#ENUMERATED**	**#PAD**
#BITS	**#EXTERNAL**	**#PrintableString**
#BIT-STRING	**#GeneralizedTime**	**#REAL**
#BMPString	**#GeneralString**	**#RELATIVE-OID**
#BOOL	**#GraphicString**	**#REPETITION**
#BOOLEAN	**#IA5String**	**#SEQUENCE**
#CHARACTER-STRING	**#INT**	**#SEQUENCE-OF**
#CHARS	**#INTEGER**	**#SET**
#CHOICE	**#NUL**	**#SET-OF**
#CONCATENATION	**#NULL**	**#TAG**
#CONDITIONAL-INT	**#NumericString**	**#TeletexString**
#CONDITIONAL-REPETITION	**#OBJECT-IDENTIFIER**	**#TIME-OF-DAY**
#DATE	**#ObjectDescriptor**	**#TRANSFORM**
#DATE-TIME	**#OCTETS**	**#UniversalString**
#DURATION	**#OCTET-STRING**	**#UTCTime**
#EMBEDDED-PDV	**#OPEN-TYPE**	**#UTF8String**
#ENCODINGS	**#OPTIONAL**	**#VideotexString**
	#OUTER	**#VisibleString**

Элементы с вышеприведенными именами должны содержать указанную последовательность знаков в имени.

8.6 Не-ECN элемент

Имя элемента — anystringexceptnonecnend

Элемент «anystringexceptnonecnend» содержит один или несколько знаков из набора знаков ИСО/МЭК 10646, но не должен быть последовательностью знаков NON-ECN-END и не должен содержать внутри эту последовательность знаков.

9 Понятия ECN

В настоящем разделе описаны главные понятия, лежащие в основе настоящего стандарта.

9.1 Спецификация нотации управления кодированием (ECN)

9.1.1 Спецификации ECN содержат один или несколько модулей определения кодирования (EDM), которые определяют правила кодирования для типов ACH.1, и один модуль компоновки кодирования (ELM), который применяет эти правила кодирования к типам ACH.1.

9.1.2 Наиболее важная часть ECN — это понятие определения структуры кодирования. ACH.1 используется для определения составных абстрактных значений с помощью простейших типов и конструкторов. Таким же образом составные кодовые последовательности могут определяться с помощью похожей нотации, где конструкционные механизмы используются для комбинирования простых битовых полей в более сложные кодовые последовательности и в конечном счете в полные сообщения. Это называется определением структуры кодирования. При использовании ECN совместно с ACH.1 необходимо, в принципе:

а) определить абстрактный синтаксис (набор абстрактных значений, подлежащих передаче и их семантику), а также

b) определить структуру кодирования (структуру полей), которая используется для переноса этих абстрактных значений, а также

c) связать компоненты абстрактного значения с полями структуры кодирования, а также

d) определить кодирование для каждого поля структуры кодирования, а также механизмы для распознавания повторений полей и распознавания альтернатив и т. д.

9.1.3 Вышеприведенный процесс обычно используется на начальных этапах. Вначале определение ACH.1 дает подробное описание конкретного абстрактного синтаксиса. Из этого автоматически генерируется предварительная структура кодирования (теоретически, внутри модуля ACH.1). Эта неявно генерируемая структура содержит только те поля, которые переносят семантику приложения, и не содержит полей для таких понятий, как определение длины, выбор альтернативы и т. п.

9.1.4 Эта структура может быть трансформирована с помощью нескольких механизмов в структуру полей, которая фактически нужна, включая поля, необходимые для поддержки работы декодера (определители). Эти механизмы в целом используют некоторую форму замены простого поля, переносящего прикладную семантику, на более сложную структуру. Такие замены образуют важную часть спецификации ECN.

9.1.5 Мы можем позже определить объекты кодирования для каждого поля в окончательной структуре. Это определит не только кодирование полей, но и способ, которым одно поле определяет длину, например, другого поля или содержит его разрешенные факультативные возможности.

9.1.6 Вышеприведенные определения расположены в модулях определения кодирования (EDM). Последним этапом является приложение набора определенных объектов кодирования к окончательной структуре кодирования с целью полного определения кодовой последовательности. Это делается в модуле компоновки кодирования (ELM).

9.2 Классы кодирования

9.2.1 Класс кодирования — это неявный признак всех типов ACH.1, который представляет набор всех возможных спецификаций кодирования для конкретного типа. Он дает ссылку, которая позволяет модулям определения кодирования определять правила кодирования для полей структуры кодирования, соответствующие этому типу. Имена классов кодирования начинаются со знака «#».

Пример — Правила кодирования для предопределенного типа ACH.1 INTEGER определяются путем ссылки на класс кодирования #INTEGER, а правила кодирования для определенного пользователем типа «My-Type» определяются путем ссылки на класс кодирования #My-Type.

9.2.2 Имеются несколько семейств классов кодирования.

9.2.2.1 Предопределенные классы кодирования

Имеются предопределенные классы кодирования, например, с именами **#INTEGER** и **#BOOLEAN**. Они позволяют определять конкретные кодирования для простейших типов ACH.1. Имеются также предопределенные классы кодирования для кодирования конструкций, таких как **#SEQUENCE**, **#SEQUENCE-OF** и **#CHOICE** (см. также 9.3.2), а также для определения правил кодирования при обработке факультативных возможностей с помощью **#OPTIONAL**. Кодирование тегов поддерживается классом **#TAG**. Наконец, имеются некоторые предопределенные классы (**#OUTER**, **#TRANSFORM** и другие), которые позволяют определять процедуры кодирования, составляющие часть процесса кодирования-декодирования, но не относящиеся прямо к какому-либо конкретному битовому полю или к конструкции ACH.1.

9.2.2.2 Классы кодирования для неявно генерируемых структур кодирования

Они имеют имена, содержащие знак «#», за которым следует «typereference», появившаяся в «TypeAssignment» в модуле ACH.1. Такие классы кодирования неявно генерируют каждый раз, когда «typereference» (непараметризованная) присваивается в модуле ACH.1, и могут импортироваться в модуль определения кодирования, чтобы позволить определение конкретных кодовых последовательностей для соответствующего типа ACH.1. Эти классы кодирования представляют структуру кодирования ACH.1 и формируются из предопределенных классов кодирования, отражая структуру определения типа ACH.1.

9.2.2.3 Классы кодирования для определенных пользователей структур кодирования

Эти классы кодирования определяются пользователем ECN путем указания структуры кодирования (см. 9.3) в виде некоторой структуры, объединяющей битовые поля и конструкторы кодирования. Эти структуры кодирования аналогичны неявно генерируемым структурам кодирования, но пользователь ECN полностью контролирует их структуру. Эти классы позволяют определять комплексные

правила кодирования и важны для использования АСН.1 с ECN при спецификации традиционных протоколов, в которых требуются дополнительные битовые поля для кодирования определителей.

9.2.2.4 Классы кодирования для явно генерируемых структур кодирования

Эти классы кодирования образуются из неявно генерируемых структур кодирования путем выборочного изменения имен определяемых классов, чтобы указать места, в которых нужны специализированные кодовые последовательности для факультативных возможностей («опций»), окончаний «последовательностей-из» и т. п.

9.3 Структуры кодирования

9.3.1 Определения структур кодирования имеют некоторое сходство с определениями типов АСН.1, они имеют имя, начинающееся знаком «#» и содержащее далее прописные буквы. Каждое определение структуры кодирования определяет новый класс кодирования (набор всех возможных кодовых последовательностей этой структуры кодирования). Структуры кодирования формируются из полей, которые являются либо предопределенными классами кодирования, либо именами других структур кодирования, составленных с помощью конструкторов кодирования (которые представляют набор всех возможных правил кодирования, поддерживающих свой тип конструкционного механизма, и поэтому называются классами кодирования) (см. в D.2.8.4 пример определения структуры кодирования).

9.3.2 Самыми основными конструкторами кодирования являются **#CONCATENATION**, **#REPETITION** и **#ALTERNATIVES**, приблизительно соответствующие типам АСН.1 sequence [«последовательность»] (и set [«множество»]), sequence-of (и set-of [«множество-из»]), а также choice [выборочный]. Имеется также класс кодирования **#OPTIONAL**, который представляет факультативное присутствие кодовых последовательностей, приблизительно соответствующих маркерам АСН.1 **DEFAULT** и **OPTIONAL**.

9.3.3 Определение структуры кодирования определяет класс кодирования, основанный на структуре. Такие классы не могут иметь те же имена, что и классы кодирования, которые импортированы в модуль (см. в ИСО/МЭК 8824-1, пункт 13.13 с учетом изменений из А.1 настоящего стандарта).

9.3.4 Имена структур кодирования могут экспортироваться и импортироваться между модулями определения кодирования и использоваться каждый раз, когда требуется имя класса кодирования в группе категорий битовых полей (см. 9.6).

9.3.5 Значения типов АСН.1 (простейших или определенных пользователем) могут быть отображены в поля структуры кодирования, а правила кодирования для этой структуры затем обеспечат кодовые последовательности этого типа АСН.1 (значения, отображенные в структуры кодирования, могут затем отображаться в поля или более сложные структуры кодирования). Это обеспечивает весьма мощный механизм для определения сложных правил кодирования.

9.4 Объекты кодирования

9.4.1 Объекты кодирования представляют конкретное определение правил кодирования для заданного класса кодирования. Обычно правила относятся к конкретным битам, которые должны вырабатываться, но могут также определять процедуры, относящиеся к кодированию и декодированию, например, способ указания на присутствие или отсутствие факультативных компонентов.

9.4.2 Чтобы полностью определить кодирование типов АСН.1 (обычно типов верхнего уровня для какого-либо приложения), необходимо определить (или получить из стандартизованных правил кодирования) объекты кодирования для всех классов, которые соответствуют компонентам этих типов АСН.1, и для используемых конструкторов кодирования.

9.4.3 Для традиционных протоколов это может делаться путем определения отдельного объекта кодирования для каждого компонента типа АСН.1, но более общую возможность дает использование объектов кодирования, определяемых с помощью стандартизованных правил кодирования (например, PER).

9.4.4 Хотя спецификации кодирования BER и PER появились раньше, чем ECN, внутри модели ECN они просто определяют объекты кодирования для всех классов, соответствующих простейшим типам и конструкторам АСН.1 (то есть для всех предопределенных классов кодирования). BER и PER рассматриваются также при создании объектов кодирования для классов кодирования, применяемых в определении структур кодирования (см. 18.2).

9.5 Наборы объектов кодирования

9.5.1 Объекты кодирования могут группироваться в наборы таким же образом, как информационные объекты в АСН.1, причем имеются такие наборы объектов кодирования, которые применяются

(в ELM) к типам АСН.1 для определения их кодирования. Руководителем, применяемым при формировании таких наборов объектов кодирования, является зарезервированное слово **#ENCODINGS** (см. пример в D.1.14).

9.5.2 Фундаментальное правило конструирования набора объектов кодирования заключается в том, что любой набор может содержать только один объект кодирования заданного класса кодирования (см. также 9.6.2). Поэтому не будет двусмысленности, когда набор объектов кодирования применяется к типу для определения его кодирования.

9.5.3 Наборы предопределенных объектов кодирования имеются для всех вариантов BER и PER, причем они могут использоваться для полных наборов, определенных пользователем объектов кодирования.

9.6 Определение новых классов кодирования

9.6.1 Те, кто осведомлен об АСН.1, знают, что присвоение типа может использоваться для создания новых имен (новых типов), например, из типов **INTEGER** и **BOOLEAN**. Новые имена определяют типы, которые будут теми же **INTEGER** или **BOOLEAN**, но несут другую семантику. Это понятие расширено в ECN, чтобы позволить создание (в присвоении класса — см. 16.1.1) новых имен (новых классов) для конструкторов, таких как **#SEQUENCE**. Новые имена определяют классы, которые выполняют такие же функции при структурировании кодовых последовательностей (например, при конкатенации), но должны иметь другие объекты кодирования, применяемые к ним. Новое имя класса, присвоенное некоторому старому классу, сохраняет определенные характеристики этого старого класса. Итак, присвоение, например, «**#My-Sequence ::= #SEQUENCE**» создает новое имя класса **#My-Sequence**, которое все еще является некоторым классом кодирования, относящимся к конкатенации компонентов. Мы говорим, что такие классы кодирования имеют одну и ту же категорию.

9.6.2 Если новый класс кодирования создан из существующего класса кодирования, то объекты кодирования как старого, так и нового классов кодирования могут появляться в одном наборе объектов кодирования.

9.6.3 Все предопределенные классы кодирования получаются от одного класса из небольшого числа простейших классов кодирования. Поэтому **#SEQUENCE** и **#SET** оба получены от класса **#CONCATENATION**, классы **#INTEGER** и **#ENUMERATED** получены от класса **#INT**, а классы для различных типов АСН.1 «цепочка знаков» все получены от класса **#CHARS**. Структура кодирования (например, неявно генерируемая из некоторого типа АСН.1) может содержать смесь разных классов, которые все получены из одного и того же простейшего класса и которые позволяют разные кодирования, применяемые, например, к **#SEQUENCE** и **#SET**.

9.6.4 Часто удобно вложить классы кодирования в категории, основанные на простейшем классе, из которого они получены. Поэтому мы говорим, что **#INTEGER**, **#ENUMERATED** и **#INT** (и любой класс, полученный из них в операторе присвоения класса, таком как «**#My-int ::= #INT**») находятся в категории «целочисленная». Имеются также группы категорий, которые содержат весьма различные классы, имеющие некоторые одинаковые характеристики. Поэтому любой класс, который может иметь абстрактные значения, прямо связанные с ним, и который, следовательно, вырабатывает биты при кодировании, будет находиться, можно сказать, в группе категорий «битовое поле». Поэтому все классы, которые находятся в категории «целочисленная», или «булева», или «цепочка знаков», относятся к группе категорий «битовое поле». Классы, которые ответственны за кодирования с группированием или повторением (например, классы в категории «альтернативы» или «повторение»), относятся к группе категорий «конструктор кодирования». Имеются также два класса, чьи объекты кодирования определяют процедуры, не относящиеся прямо к конструированию кодирования (**#TRANSFORM** и **#OUTER**): они считаются находящимися в группе категорий «процедура кодирования». Структуры кодирования определяются с помощью классов в группе категорий «битовое поле», которые комбинируются с классами из группы категорий «конструктор кодирования» вместе с классами из категорий «факультативные возможности» (представляют процедуры кодирования для разрешенных факультативных возможностей) и «тег» (представляют кодирование тегов). Все такие классы находятся в категории «структура кодирования», а также в группе категорий «битовое поле».

9.6.5 Для простейших классов категория присваивается прямо. Для классов, созданных в операторе присвоения класса кодирования, категория определяется с помощью нотации справа от символа «**::=**». Если эта нотация является определением структуры кодирования, то класс будет относиться как к категории «структура кодирования», так и к группе категорий «битовое поле». Если эта нотация

является справочным именем простого класса, то категория нового класса будет такой же, как категория присвоенного класса.

9.6.6 Категориями класса кодирования (см. 16.1.3) являются:

- категория «альтернативы» (alternatives) (классы, которые получены путем присвоения класса из **#ALTERNATIVES**);
- категория «конкатенация» (concatenation) (классы, которые получены путем присвоения класса из **#CONCATENATION**);
- категория «повторение» (repetition) (классы, которые получены путем присвоения класса из **#REPETITION**);
- категория «факультативные возможности (опции)» (optionality) (классы, которые получены путем присвоения класса из **#OPTIONAL**);
- категория «тег» (tag) (классы, которые получены путем присвоения класса из **#TAG**);
- категории «булева» (boolean), «цепочка битов» (bitstring), «цепочка знаков» (characterstring), «целочисленная» (integer), «вырожденная» (null), «идентификатор объекта» (objectidentifier), «цепочка октетов» (octetstring), «открытый тип» (opentype), «pad» и «действительное число» (real) (категории для классов, которые получаются из соответствующих простейших классов);
- категория «структура кодирования» (encoding structure) (классы, генерируемые из определений типов АСН.1 или с помощью явного определения структуры кодирования).

9.6.7 Определены следующие группы категорий:

- «битовое поле» (bit-field) (классы, которые соответствуют реальным полям при кодировании, таким как поля в категории «целочисленная» или «булева», вместе с любым классом в категории «структура кодирования»). Классы в этой группе категорий называются также классами «битовое поле»;
- «конструктор кодирования» (encoding constructor) (классы, которые находятся в категориях «альтернативы», «конкатенация» или «повторение»). Классы в этой группе категорий называются также классами «конструктор кодирования»;
- «процедура кодирования» (encoding procedure) (классы, которые прямо не связаны с конструкторами АСН.1 и которым не могут быть присвоены новые имена — **#OUTER**, **#TRANSFORM**, **#CONDITIONAL-INT**, **#CONDITIONAL-REPETITION**). Классы в этой группе категорий называются также классами «процедуры кодирования».

9.7 Определение объектов кодирования

Имеются восемь механизмов, доступных для определения объектов кодирования заданного класса кодирования. Они не все доступны для всех классов кодирования.

9.7.1 Первый механизм определяет объект такой же, как некоторый другой определенный объект кодирования затребованного класса. Это всего лишь обеспечение синонима для объектов кодирования.

9.7.2 Вторым механизмом, доступным для ограниченного набора классов кодирования, является использование определенного синтаксиса (см. 17.2) для указания информации, необходимой для определения объекта кодирования этого класса. Многое из необходимой информации является общим для всех классов кодирования, а некоторая информация всегда зависит от конкретного класса кодирования (см. в D.1.1.2 пример определения объекта кодирования класса **#BOOLEAN**, который содержит кодирования для булева типа АСН.1).

9.7.3 Третий механизм, доступный для всех классов кодирования — это определение объекта кодирования в виде кодовой последовательности необходимого класса, которая содержится в некотором существующем наборе объектов кодирования. Это используется главным образом при именовании объекта кодирования для отдельного класса, который будет выполнять кодирования BER или PER для этого класса.

Примечание — Это часто может быть полезным, но требует знания процедур кодирования в виде стандартизованных правил кодирования.

9.7.4 Четвертый механизм — это отображение абстрактных значений, связанных с некоторым классом кодирования (например, **«#A»**), в абстрактные значения, связанные с другим (обычно более сложным) классом кодирования (например, **«#B»**), и определение объекта кодирования для **«#B»** (используя любой доступный механизм). Объект кодирования для абстрактных значений, связанных с **«#A»**, может быть теперь определен как приложение объекта кодирования для **«#B»** к соответствующим

абстрактным значениям, связанным с **«#B»** (см. пример D.2.8.3). Вариантов такого отображения много (см. 9.17).

Примечание — Это является моделью, лежащей в основе определения объекта для кодирования целочисленного типа в BER. Целое число отображается в структуру кодирования, которая содержит поле какого-либо класса «тег» (**UNIVERSAL, APPLICATION, PRIVATE** или зависящий от контекста), примитива/конструктора булево значение, поле номера тега и часть значения, которая кодирует абстрактные значения исходного целого числа.

9.7.5 Пятый механизм — это определение объекта кодирования для некоторого класса (например, соответствующего определенному пользователем типу ACH.1) путем отдельного определения объектов кодирования для компонентов и конструктора кодирования, используемых в определении этого класса кодирования.

9.7.6 Шестой механизм — это определение объекта кодирования для дифференциального кодирования-декодирования (см. 9.8) с использованием двух отдельных объектов кодирования, один из которых определяет поведение кодера, а другой указывает декодеру, как должно учитываться кодирование.

Примечание — Примером будет кодирование поля, которое «зарезервировано для будущего использования» в виде всех нулей, но обеспечивается прием некоторого значения при декодировании.

9.7.7 Седьмой механизм — это определение такого объекта кодирования с факультативными способами кодирования, который содержит упорядоченный список объектов кодирования одного и того же класса. Кодер может выбирать для применения объект кодирования из списка с учетом следующего ограничения: если только один факультативный способ кодирования может кодировать заданное абстрактное значение, то он будет применяться, при этом рекомендуется, чтобы использовался первый доступный способ кодирования из списка.

Примечание — Объект кодирования с факультативными возможностями может, например, использоваться в спецификации кодовых последовательностей с короткой длиной, где они могут кодировать конкретную длину цепочки, используя кодовые последовательности с большой длиной, где короткая длина не может использоваться. Сейчас нет механизма для спецификатора ECN, запрашивающего использование первого доступного объекта кодирования (если более чем один объект могут кодировать абстрактное значение), кроме применения комментария.

9.7.8 Наконец, объект кодирования может быть определен с помощью не-ECN нотации. Это является средством, позволяющим использовать любую желательную нотацию (включая единственный язык) для определения объекта кодирования (см. D.2.7.3).

Примечание — Не-ECN нотация должна использоваться с осторожностью, так как в этом случае обычно невозможна инструментальная поддержка реализации.

9.8 Дифференциальное кодирование-декодирование

9.8.1 Дифференциальное («различительное») кодирование-декодирование — это термин, применяемый к спецификации, которая требует реализации, принимающей (при декодировании) дополнительные битовые последовательности к тем, которые разрешены для генерирования при выполнении кодирования.

9.8.2 Дифференциальное кодирование-декодирование лежит в основе всей поддержки «растяжимости» (способности реализации ранней версии стандарта иметь хорошую возможность взаимодействия с реализацией поздней версии настоящего стандарта).

9.8.3 Дифференциальное кодирование-декодирование может быть очень сложным. Обычно оно содержит требование, чтобы декодер принимал (и молчаливо игнорировал) поля заполнения (обычно с переменной длиной), которые в последующих версиях стандарта будут использоваться для переноса информации, дополняющей ту, которая переносится в ранней версии связи.

9.8.4 Поддержка дифференциального кодирования-декодирования в ECN обеспечивается синтаксисом, который позволяет определять (для любого класса) такой объект кодирования, в который вложены два объекта кодирования. Каждый объект кодирования определяет правила кодирования. Первый объект кодирования определяет правила, используемые кодером. Декодер использует второй объект кодирования в качестве спецификации способа, которым было выполнено кодирование.

Примечание — В ECN правила, используемые декодером (в ранней версии стандарта), всегда выражаются при помощи правил кодирования, которые, как предполагает декодер, использует его партнер по связи. Правила декодирования не даются в виде явных правил кодирования. Спецификатор ECN будет гарантировать, что такие правила декодирования обеспечат некоторую необходимую «растяжимость».

9.9 Факультативные возможности в кодовых последовательностях

9.9.1 Факультативные возможности («опции») в протоколах обычно считаются сегодня чем-то, чего следует избегать, но ECN должна обеспечивать поддержку таких факультативных возможностей, если разработчик протокола решает (или решил ранее) применить их.

9.9.2 Когда значения кодируются в пространство кодирования, возможно указать, что размер пространства кодирования (см. 9.21.5) является факультативной возможностью кодера, обеспечив наличие некоторой формы определителя длины, связанного с кодированием (расширение факультативных возможностей может быть ограничено максимальным значением, которое может кодироваться в определителе длины). Это обеспечивает некоторый детальный уровень поддержки факультативных возможностей кодирования.

9.9.3 Более общим является механизм, похожий на поддержку дифференциального кодирования-декодирования (см. 9.8), но в этом случае объект кодирования для некоторого класса может определяться в виде выбора в кодере объекта кодирования из упорядоченного списка определенных объектов кодирования для этого класса. Дополнительно к определению списка возможных кодовых последовательностей необходимо также обеспечить спецификацию объекта кодирования для класса в категории «альтернативы» (см. 9.6). Этот объект кодирования определяет кодовые последовательности и процедуры, необходимые декодеру, чтобы определять, какой именно объект кодирования был применен кодером.

9.10 Свойства объектов кодирования

9.10.1 Объекты кодирования имеют некоторые общие свойства. В большинстве случаев они полностью определяют кодирование, но в некоторых случаях они являются конструкторами кодирования, то есть они определяют только структурные аспекты кодирования, которые затребуют объекты кодирования для компонентов структуры кодирования, чтобы завершить определение кодовой последовательности.

9.10.2 Другой ключевой характеристикой объекта кодирования является то, что он может затребовать информацию от вычислительной среды, в которой его правила в конечном счете применяются. Одним из аспектов такой вычислительной среды, который всегда поддерживается, является наличие границ в определении типа ACH.1, которые будут «видимыми для PER» (см. ИСО/МЭК 8825-2, подраздел 9.3).

Примечание — Несколько другой (и нестандартизованной) внешней зависимостью могло бы стать определение не-ECN объекта кодирования для класса кодирования **#ALTERNATIVES**, который указывает выбранную альтернативу, основанную на внешних данных, например, о канале, по которому передано сообщение.

9.10.3 Третьей ключевой характеристикой является то, что объект кодирования может применять идентификационный описатель в своих кодовых последовательностях. Это является такой частью (состоящей из фиксированного набора битовых позиций) всех кодовых последовательностей, которая образует и различает свои кодовые последовательности от кодовых последовательностей других объектов кодирования (любого класса), которые применяют такой же идентификационный описатель. Идентификационные описатели должны быть видимы для декодеров, не имеющих информации о том, что было кодировано — класс кодирования или абстрактное значение (но имеющих информацию об имени используемого идентификационного описателя). Это понятие моделирует (и обобщает) использование тегов в кодированиях BER: значение тега в BER может определяться без знания класса кодирования для всех кодовых последовательностей BER и служит для указания кодовой последовательности с целью различать функциональные возможности, упорядочивать наборы и выбирать альтернативы.

9.11 Параметризация

9.11.1 Объекты кодирования, наборы объектов кодирования и классы кодирования могут быть параметризованы так же, как типы и значения ACH.1. Это является просто расширением нормального механизма ACH.1.

9.11.2 В первую очередь параметризация используется при определении объекта кодирования, который нуждается в идентификации некоторого определителя для завершения определения кодовой последовательности (см. 9.13.2) (см. в D.1.11.3 пример параметризованного определения ECN).

9.11.3 Другим важным использованием параметризации является определение структуры кодирования, которая будет использована для замены многих разных классов в кодировании (см. также 9.16.5).

Например, механизмом, применяемым для обработки факультативных возможностей, часто является непосредственный (обязательный) предшествующий «бит-наличия» для каждого факультативного компонента. Параметризованная структура может быть определена включением конкатенации **#BOOLEAN** (используемой в качестве определителя наличия), за которой следует факультативный компонент, определенный в качестве фиктивного параметра (который должен выдаваться с компонентом, заменяющим структуру), на наличие которого указывает этот **#BOOLEAN**. Исходная процедура кодирования **#OPTIONAL** определяется теперь как замена исходного компонента с его обязательной структурой путем использования исходного факультативного компонента в качестве реального параметра (в D.3.2 приводится более полный пример этого процесса).

9.11.4 Фиктивными параметрами могут быть объекты кодирования, наборы объектов кодирования, классы кодирования, ссылки на поля структур кодирования и значения любых типов АСН.1, используемых в предопределенных классах кодирования, определяемых в разделе 23 и описанных в ИСО/МЭК 8824-4, с учетом изменений из В.10 (приложение В).

9.11.5 Изменение синтаксиса параметризации, указанное в приложении С, требует использования символа «**{<**» (без пробелов) вместо «**{**» для начала списка фиктивных или реальных параметров и «**>}**» для его окончания.

Примечание — Это сделано для облегчения анализа ECN в компьютере и для избежания двусмысленности, когда определенные пользователем классы используются в определениях структур вместо **#SEQUENCE**, **#CHOICE**, **#REPETITION**, **#SEQUENCE-OF** или **#SET-OF**.

9.12 Руководители

9.12.1 Понятия «руководитель или руководящее указание» и «руководимая нотация» будут близки к нотации значения АСН.1, в которой всегда имеется определение типа, которое «руководит» нотацией значения и определяет ее синтаксис и смысл.

9.12.2 Это же понятие распространяется на определение объектов кодирования заданного класса кодирования. Синтаксис для определения объекта кодирования класса **#BOOLEAN** (к примеру) сильно отличается от синтаксиса для определения объекта кодирования класса **#INTEGER** (к примеру). Во всех случаях, где требуется определение объекта кодирования, имеется некоторая связанная нотация, которая определяет класс этого объекта кодирования и «руководит» синтаксисом, который следует использовать в его спецификации.

9.12.3 Синтаксис ECN нуждается в руководителях, которые являются классами кодирования и будут справочными именами классов или именами параметризованных классов.

9.12.4 Если руководимая нотация является справочным именем объекта кодирования, то этот объект кодирования должен быть того же класса, что и руководитель (см. 17.1.7).

9.13 Общие аспекты кодирования

9.13.1 ECN обеспечивает поддержку ряда методов, обычно используемых в определениях правил кодирования (не просто тех методов, которые использованы в BER и PER). Например, ECN признает, что факультативные возможности могут разрешаться любым из трех путей: использованием определителя наличия, использованием идентификационного описателя (см. 9.13.3) или достижением конца некоторого контейнера с ограниченной длиной (или конца PDU) перед появлением факультативного компонента.

9.13.2 Аналогично она признает, что разделение повторений может выполняться, например, с помощью:

- какой-либо формы подсчета длины;
- обнаружения конца некоторого контейнера (или PDU), в котором оно является последним элементом;
- использования описателя идентификации в каждом повторении и в каждой последующей кодовой последовательности (см. 9.13.3);
- некоторой оканчивающей комбинации, которая никогда не может появиться в кодировании повторяющихся серий (простым примером является цепочка знаков, оканчивающаяся нулем).
- использования с каждым элементом «бита ЕЩЕ», который устанавливается в ЕДИНИЦУ для указания, что последует другое повторение, и устанавливается в НУЛЬ для указания конца повторений.

ECN поддерживает все эти механизмы для разделения повторений и аналогичные механизмы для идентификации альтернатив и для разрешения функциональных возможностей.

9.13.3 Кроме использования в окончаниях повторений, метод идентификационного описателя может использоваться также для определения наличия факультативных компонентов или альтернатив и упорядочения наборов. Механизм одинаков во всех этих случаях. Учитывая класс кодирования, который является «возможным следующим классом», и объект кодирования, примененный к нему, любое произведенное кодирование будет содержать на некоторых битовых позициях (в описателе идентификации) битовую комбинацию, соответствующую битовой комбинации в пределах заданного набора битовых комбинаций (набора значений описателя), характеризующую этот класс, но не соответствующую ни одной битовой комбинации, характеризующей любой другой «возможный следующий класс». Все такие способы кодирования могут распознаваться декодером как кодирование «возможного следующего класса», а битовая комбинация, найденная при кодировании, будет определять, какое именно кодирование «возможного следующего класса» присутствует. Это понятие аналогично использованию тегов для таких целей в BER. Описатели идентификации имеют имена, которые должны быть уникальными в пределах спецификации ECN.

9.13.4 Здесь важно отметить, что ECN позволяет определять способы кодирования очень гибким способом, но не может гарантировать, что спецификация кодирования правильна, то есть что декодер сможет успешно восстановить исходные абстрактные значения из кодирования. Например, определитель ECN может назначить одну и ту же битовую комбинацию для булевых значений ИСТИНА и ЛОЖЬ. Это будет ошибкой, и в этом случае некоторый инструмент достаточно легко обнаружит ошибку. Другой ошибкой может быть утверждение, что кодирование имеет саморазграничение (и не требует определителя длины), когда фактически это не так. Эта ошибка также может быть обнаружена некоторым инструментом. В более тонких и сложных случаях, однако, инструмент может не справиться с диагностикой ошибочной спецификации (с ошибкой, которая не всегда успешно обнаруживается).

9.14 Идентификация информационных элементов

9.14.1 Многие протоколы имеют кодирование (обычно с фиксированным числом битов) для определения в протоколе того, что часто называют «информационными элементами» или «элементами данных». Такие идентификации приблизительно соответствуют тегам АСН.1, но обычно менее сложны. Они часто используются в качестве идентификационных описателей, однако используются так не всегда.

9.14.2 ECN содержит класс **#TAG** для поддержки определения кодирования идентификаторов информационных элементов путем использования нотации тегов АСН.1 (он поддерживает также включение таких элементов в структуру кодирования без ссылки на теги АСН.1).

9.14.3 Когда структура кодирования неявно генерируется из определения типа АСН.1 (см. раздел 11), первая текстуально выраженная нотация тега АСН.1 в этом определении генерирует экземпляр класса **#TAG** с номером тега АСН.1, связанным с этим экземпляром класса **#TAG**. Последующие текстуально выраженные экземпляры нотации тега АСН.1 не отображаются в классы **#TAG** в этой неявно генерируемой структуре, но эти теги и их значения становятся признаками этих элементов. Кодирование для этого класса кодирования может определяться таким же способом, как кодирование для класса **#INTEGER**, и будет кодировать номер в нотации тега.

9.14.4 Полный список тегов АСН.1 (множество тегов, каждый с классом и номером) теоретически связан со всеми абстрактными значениями тегированного типа согласно модели АСН.1. Такая информация является, однако, доступной только в существующей версии ECN с помощью не-ECN определителя объекта кодирования (см. 9.7.8). Генерация класса **#TAG** является отдельным механизмом, более простой и более определенной и имеет полную поддержку в ECN.

9.14.5 Важно, однако, заметить, что в целях генерации класса **#TAG** видимой является только текстуально выраженная нотация тегов. Теги универсального класса и теги, генерируемые автоматическим тегированием, не видны. Аналогично игнорируется класс любой текстуально выраженной нотации тегов. Для кодирования объектов класса **#TAG** доступен только номер тега.

9.15 Ссылочные поля и определители

9.15.1 Очень общим (но не единственным) способом определения наличия факультативного поля, длины повторения или выбора альтернативы является включение поля определителя (где-нибудь в сообщении). Если этот механизм применяется для определения, то должны указываться поля определителя, а это часто требует некоторого фиктивного параметра в определении объекта кодирования, обеспечивающего вместе с реальным параметром имя поля структуры кодирования в определителе, который будет выдаваться, когда объект кодирования прилагается к структуре кодирования.

9.15.2 Новое понятие — ссылочное поле — вводится для удовлетворения потребности в фиктивном параметре, который ссылается на поле структуры кодирования. Руководителем является зарезервированное слово **REFERENCE**, а разрешенной нотацией для реального параметра с этим руководителем является некоторое имя поля структуры кодирования внутри этой структуры кодирования, к которой приложен объект кодирования или набор объектов кодирования с таким параметром (см. 17.5.15) (см. в D.1.11.3 пример ссылок на имена полей структуры кодирования).

9.16 Классы и структуры замены

9.16.1 При наличии спецификаций ACH.1 для традиционных протоколов (или при генерировании специальных кодовых последовательностей для новых протоколов) считается нормальным игнорировать элементы кодирования, в частности поля определителя, которые присутствуют только для поддержки декодирования. В спецификацию ACH.1 включаются только поля, уместные для прикладного кода (переносящие прикладную семантику).

9.16.2 Когда в таких протоколах используется более одного механизма для поддержки (к примеру) конструкций **SEQENCE OF** в разных местах протокола, невозможно (либо не подходит) формально указывать это в самой ACH.1.

9.16.3 Это означает, что неявно генерируемая структура кодирования не будет различать такие конструкции и не будет содержать относящиеся к кодированию поля для определителей, и необходимо изменить ее для «решения» обеих проблем, пока недоступна некоторая структура, соответствующая требованиям кодирования.

9.16.4 Первым и простейшим изменением является замена некоторых экземпляров класса (внутри неявно генерируемой структуры) на имена нового класса, которые присваиваются старому классу в операторе присвоения класса. Это делается путем создания явно генерируемой структуры с использованием раздела переименований в EDM. Этот раздел импортирует неявно генерируемую структуру из модуля ACH.1 и выполняет указанные замены мест появления (текстуальных) именованных классов. Замена может быть для всех текстуальных мест в пределах списка неявно генерируемых классов (соответствующих определениям типов ACH.1 в модуле) либо в пределах компонентов одного из таких классов, либо «всех мест появления, за исключением» мест в заданном определении или заданном компоненте (см. 15.3). Здесь важно отметить, что эти замены ограничены использованием классов, которые были определены в том операторе присвоения класса кодирования, который присваивает имя класса замены старому классу (например, «**#Replacement-class ::= #Old-class**»), поэтому данный механизм иногда называют в разговорах «окрашиванием». Это «окрашивание» указывает те части спецификации, которым необходимы способы кодирования, отличающиеся от способов других частей (пример «окрашивания» дается в D.3.7).

9.16.5 Даже с использованием «окрашивания» явно генерируемая структура кодирования, как и неявно генерируемая структура кодирования, содержит только поля, соответствующие полям в спецификации ACH.1, а обычно необходимо изменять генерируемые структуры, чтобы добавить поля для определителей и т. п. Необходима новая структура замены (для всей или части исходной структуры) с добавленными полями. Важно также указывать (для каждого поля исходной структуры), какие именно поля структуры замены (и какие абстрактные значения такого поля) используются для переноса семантики исходных абстрактных значений. Мы говорим об отображении абстрактных значений из исходной структуры в структуру замены.

9.16.6 Имеется много механизмов для определения объекта кодирования существующей структуры в качестве объекта кодирования для полностью отличающейся структуры замены с определенными отображениями значений между старой структурой и структурой замены. Эти механизмы описываются в 9.17.

9.16.7 Однако часто встречается более простая ситуация, в которой разработчик требует от старой структуры сформировать (сохраняя свою целостность) один компонент структуры замены со всеми абстрактными значениями, отображенными из старой структуры в соответствующее значение такого компонента структуры замены. Для этого механизма, получающего всеобщее применение, структура замены нуждается в фиктивном параметре для этого одного компонента и для создания его с реальным параметром, установленным в старую структуру. Это описано в 9.11.3.

9.16.8 При определении объектов кодирования для какого-либо класса (любого класса) всегда возможно указать, что первым действием этого объекта кодирования будет замена класса, кодируемого им, на параметризованную структуру замены, создаваемую согласно 9.16.7, с абстрактными значениями, отображенными из старого класса в этот компонент.

9.16.9 Возможно также определить объекты кодирования для класса **#OPTIONAL** (или для любого класса категории «факультативные возможности»), который заменяет факультативный компонент на параметризованную структуру замены (часто структуру, содержащую поле #BOOLEAN в качестве определителя наличия) (пример этого приводится в D.3.2.3).

9.16.10 Для таких классов конструктора, как **#CONCATENATION, #REPETITION** и т. п., возможно также определить объекты кодирования, которые заменяют не полную структуру, а отдельно каждый компонент (или только обязательные, или только факультативные компоненты).

9.16.11 Более развитым, но и более мощным механизмом является требование к действию замены включать также введение определенного поля в заголовок структуры **#CONCATENATION** (или аналогичной структуры) (пример этого приводится в D.3.1.5).

9.17 Отображение абстрактных значений в поля структур кодирования

Имеются шесть механизмов, предусмотренных для этой цели.

9.17.1 Первый механизм — это отображение указанных абстрактных значений, связанных с одним простым классом кодирования, в указанные абстрактные значения, связанные с другим простым классом кодирования. Это может использоваться многими способами. Например, значения «цепочки знаков» (цифр) могут быть отображены в целочисленные значения и т. д. (см. 19.2) (см. пример в D.1.10.2).

9.17.2 Второй механизм — это отображение полного поля одной структуры кодирования в поле совместимой структуры кодирования, которая может содержать дополнительные поля обычно для использования в качестве определителей длины или выбора (см. 19.3) (см. пример в D.2.8.3).

9.17.3 Третий механизм — это отображение путем трансформирования всех абстрактных значений, связанных одним классом кодирования, в абстрактные значения, связанные с другим (обычно, но не обязательно) классом кодирования, используя некоторый трансформирующий объект кодирования (см. 9.18). Этим механизмом возможно, например, отобразить **#INTEGER** в **#CHARS**, чтобы получить знаки, которые могут быть затем кодированы любым желательным способом (например, способом «двоично-кодированное десятичное число» или способом ASCII) (см. пример в D.1.6.3).

9.17.4 Четвертый механизм отображения — это использование указанного упорядочения абстрактных значений определенных типов и конструкций и отображение согласно этому упорядочению. Это образует очень мощные средства кодирования абстрактных значений, связанных с одним классом кодирования, как будто они были абстрактными значениями, связанными с полностью независимым классом кодирования (см. 19.5) (см. пример в D.1.4.2).

9.17.5 Пятый механизм — это распределение абстрактных значений (используя нотацию диапазона значений), связанных с одним классом кодирования (обычно **#INTEGER**), в поля другого класса кодирования (см. примеры в 19.6 и D.2.1.3).

9.17.6 Последний механизм позволяет спецификатору ECN обеспечить явное отображение целочисленных значений (которые могут быть образованы более ранним отображением, например, из класса **#ENUMERATED**) в биты, которые должны использоваться для кодирования таких значений. Это предназначается для поддержки кодовых последовательностей Хаффмана, где частость появления каждого значения известна (по крайней мере приблизительно) и где требуется оптимальное кодирование. В приложении Е подробно описываются коды последовательностей Хаффмана и даются примеры этого механизма, а также ссылки на программное обеспечение, которое будет генерировать синтаксис ECN для этих отображений, задавшись только относительной частостью, с которой, как ожидается, будет использоваться каждое значение целого числа (см. 19.7).

9.18 Преобразователи и композиции преобразования

9.18.1 Преобразователи — это объекты кодирования из класса **#TRANSFORM**. Они могут использоваться для преобразования абстрактных значений между разными классами кодирования и для определения простых арифметических функций, таких как умножение на фиксированное значение, вычитание фиксированного значения и т. п. При последовательном применении они дают возможность указывать общую арифметику (см. 19.4) (см. пример в D.2.4.2).

9.18.2 Преобразователь может взять одиночное значение в качестве своего источника и затем образовать одиночное значение в качестве своего результата. Ниже даются классификационные группы значений, которые могут быть источниками и результатами преобразователей:

- целое число;
- булево значение;

- цепочка знаков;
- цепочка битов;
- одиночный знак;
- одиночный бит (только источник, поддерживающий кодирование цепочки битов, см. 23.2).

9.18.3 Композиции преобразования — это упорядоченный список элементов, которые все являются одиночными значениями и имеют одну и ту же классификационную группу (из перечисленных в 9.18.2) (например, упорядоченный список одиночных знаков, или одиночных октетов, или целых чисел). Они создаются только в виде результатов преобразователей и могут использоваться только как источник последующего преобразования.

9.18.4 Если классификационной группой является цепочка битов, то размеры каждого значения цепочки битов в композиции будут одинаковыми и статически определяться преобразователем, который вырабатывает эту композицию (например, упорядоченный список одиночных битов или шестибитовых блоков).

9.18.5 Имеются преобразователи из следующих абстрактных значений в композиции:

- из цепочки знаков в композицию одиночных знаков;
- из цепочки битов в композицию цепочек битов (все значения цепочек битов в композиции имеют один и тот же размер);
- из цепочки октетов в композицию цепочек битов (все значения цепочек битов в композиции имеют размер 8 битов).

9.18.6 Имеются преобразователи из следующих композиций в абстрактные значения:

- из композиций одиночных знаков в значения цепочки знаков;
- из композиций цепочек битов в значения цепочки битов;
- из композиций цепочек битов (со значениями цепочки битов размером 8 битов) в значения цепочки октетов.

9.18.7 Все другие преобразователи могут принимать некоторое значение в качестве своего источника и вырабатывать новое значение (той же или другой классификационной группы). Они могут также использовать некоторую композицию преобразования в качестве своего источника и вырабатывать композицию в качестве своего результата, преобразуя каждый элемент композиции-источника в элемент композиции-результата.

9.19 Содержимое модулей определения кодирования

9.19.1 Модули определения кодирования (EDM) содержат операторы экспорта и импорта такие же, как в ACH.1 (но они могут импортировать только объекты кодирования, наборы объектов кодирования и классы кодирования из других модулей EDM или из модулей ACH.1 в случае неявно генерируемых структур кодирования).

9.19.2 EDM может содержать также раздел переименований (см. раздел 15), который ссылается на неявно генерируемые структуры кодирования из одного или нескольких модулей ACH.1 и генерирует путем их «окрашивания» (см. 9.16.4) явно генерируемую структуру кодирования для каждой из них. Эти явно генерируемые структуры кодирования доступны для использования внутри этого EDM, но также автоматически экспортируются для возможного импорта в модуль компоновки кодирования.

9.19.3 Тело модуля EDM содержит:

операторы «EncodingObjectAssignment», которые определяют и именуют объект кодирования для некоторого класса кодирования (имеются восемь форм этого оператора, обсужденные в 9.7 и определяемые в разделе 17);

операторы «EncodingObjectSetAssignment», которые определяют наборы объектов кодирования (см. раздел 17);

операторы «EncodingClassAssignment», которые определяют и именуют новые классы кодирования (см. раздел 15).

9.19.4 EDM может также содержать параметризованные версии этих операторов, как определяется в разделе 14 и в С.1 (приложение С).

9.19.5 Объекты кодирования могут определяться для предопределенных классов кодирования внутри любого модуля EDM. Объекты кодирования могут определяться для генерируемой структуры кодирования только в модулях EDM, которые импортируют неявно генерируемую структуру кодирования из модуля ACH.1, определяющего соответствующий тип (используя раздел импортов или

переименований), или которые импортируют генерируемую структуру кодирования из модуля EDM, экспортирующего ее.

Примечание — Если неявно генерируемая структура кодирования случайно имеет имя, совпадающее с именем зарезервированного класса кодирования (см. 8.5), то она, тем не менее, может быть импортирована в EDM, но должна быть указана в теле EDM с использованием полностью уточненного имени (см. «ExternalEncodingClassReference» в 10.6).

9.20 Содержимое модуля компоновки кодирования

9.20.1 Все приложения нотации управления кодированием нуждаются в идентификации единственного модуля компоновки кодирования (или ELM).

9.20.2 Модуль ELM применяет наборы объектов кодирования к типам ACH.1 (формально, к некоторой генерируемой структуре кодирования, соответствующей типу ACH.1). Эти наборы объектов кодирования (или составляющие их объекты кодирования) импортируются в модуль ELM из одного или нескольких модулей EDM.

9.20.3 Имеются ограничения на применение наборов объектов кодирования, гарантирующие, что не будет двусмысленности о реальных правилах кодирования, которые применяются (см. 12.2.5). Например, в ELM не разрешается применять более одного набора объектов кодирования для определения неявно генерируемой структуры.

9.20.4 В простом случае возможно, что модуль ELM содержит только один элемент (за которым следует раздел импортов), который применяет набор объектов кодирования к неявно генерируемой структуре кодирования, соответствующей одиночному типу верхнего уровня в некотором приложении (см. пример в D.1.17).

9.21 Определение способов кодирования для простейших классов кодирования

9.21.1 Правила кодирования для некоторых простейших классов кодирования могут определяться с помощью дружественного к пользователю синтаксиса, который указан в командах **WITH SYNTAX** определений класса кодирования (см. разделы 23 и 25). Этот синтаксис может также использоваться при определении правил кодирования для классов кодирования, полученных из этих простейших классов кодирования (при помощи операторов присвоения класса кодирования).

9.21.2 Нотация, которая используется для определений класса кодирования в разделах 23 и 25, основана на нотации, примененной для определения класса информационных объектов. Этот синтаксис (и связанная с ним семантика) определен путем ссылки на ИСО/МЭК 8824-2 с учетом изменений из приложения В.

9.21.3 Определение класса кодирования указывает информацию, которая должна поставляться с целью определить правила кодирования для конкретных классов кодирования. Набор правил кодирования, который может быть определен этим способом, не охватывает, конечно, все возможные правила, но считается достаточным для спецификаций кодирования, которые потребуются, вероятно, пользователям ECN.

9.21.4 Эти определения класса кодирования описывают серии полей (с соответствующими типами ACH.1 и семантиками). Правила кодирования описываются путем выдачи значений этих полей. Значения этих полей фактически дают значения ряда признаков кодирования, которые совместно определяют кодирование.

9.21.5 Смысл признаков кодирования описывается с использованием модели кодирования (см. рисунок 1), в которой значения каждого класса битовых полей образуют кодирования значений, которые помещаются (с левым или правым выравниванием) в пространство кодирования.

9.21.6 Пространство кодирования может иметь свой передний край, выровненный (сфазированный) по некоторой границе (например, границе октета) путем предварительного заполнения пространства кодирования, а его размер может быть фиксированным или переменным. Код значения помещается внутри него, возможно с левым или правым выравниванием и с окружающим заполнением. Если пространство кодирования имеет переменный размер, то либо кодирование значения должно быть саморазграничивающим, либо должен быть некоторый внешний механизм, позволяющий декодеру определять размер пространства кодирования. Имеются разные механизмы для этого определения.

9.21.7 Наконец, полное пространство кодирования с кодом значения, с предварительным заполнением значения и последующим заполнением значения отображается в «биты в строке» с факультативным определением реверсии битов. Оно обрабатывает кодовые последовательности, которые требуют

условия «старший значащий байт первым» или «старший значащий байт последним» для целых чисел либо требуют, чтобы биты в октете были реверсированы по сравнению с нормальным порядком.

9.21.8 Таким образом, имеются три широкие категории необходимой информации:

- первая относится к пространству кодирования, в котором размещаются кодовые последовательности;
- вторая относится к способу отображения абстрактного значения в биты (кодирование значения) и к размещению таких битов внутри пространства кодирования;
- третья относится к какой-либо реверсии битов.

9.21.9 На рисунке 1 показаны пространство кодирования (с предварительным заполнением) и код значения (с предварительным заполнением значения и последующим заполнением значения). Рисунок 1 иллюстрирует также определение единицы пространства кодирования. Пространство кодирования всегда является целым, кратным этому указанному числу битов.

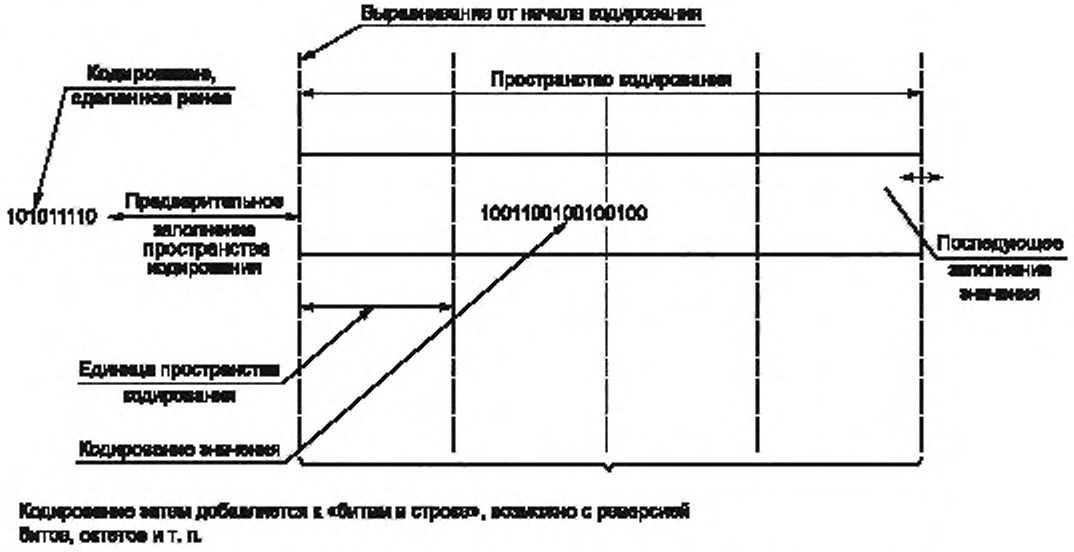

Рисунок 1 — Пространство кодирования, кодирование значения и понятия «заполнение»

9.21.10 Если пространство кодирования не имеет одного размера для всех значений, кодируемых объектом кодирования, то необходим некоторый дополнительный механизм для определения фактического пространства кодирования, используемого в экземпляре кодирования.

9.21.11 Возможно также указать произвольное значение предварительного заполнения кодера (за пределами, необходимыми для выравнивания), которое заканчивается, когда значение более раннего указателя старта укажет начало некоторого поля.

9.21.12 Определение кодирования для простейшего класса кодирования битового поля состоит из следующих шагов:

- указать выравнивание (если оно есть), необходимое для переднего края пространства кодирования (относительно точки выравнивания — обычно начала кодирования типа верхнего уровня, то есть типа, к которому прикладывается набор объектов кодирования в ELM) (см. 22.2);
- указать форму какого-либо необходимого заполнения для этой точки (предварительное заполнение пространства кодирования) (см. 22.2);
- указать (если необходимо) поле, которое обеспечит указатель для начальной точки пространства кодирования (см. 22.3);
- указать кодирование абстрактных значений в биты (кодирование значения);
- указать единицы пространства кодирования (пространство кодирования всегда будет целым, кратным этим единицам) (см. 22.4);
- указать размер пространства кодирования в этих единицах. Он может быть фиксированным (с использованием знания целого числа или границ размера, связанных с кодируемыми абстрактными

значениями) или переменным (разным для каждого абстрактного значения). Спецификация может также (во всех случаях) указать на использование определителя длины, который должен кодироваться с длиной поля и либо создавать возможность декодирования, либо предоставлять избыточную информацию (в случае пространства кодирования с фиксированным размером), которую декодер может проверять (см. 22.4);

- указать выравнивание кодирования значения внутри пространства кодирования (см. 22.8);
- указать форму какого-либо необходимого заполнения от начала пространства кодирования до начала кодирования значения (предварительное заполнение значения) (см. 22.8);
- указать форму какого-либо необходимого заполнения между концом кодирования значения и концом пространства кодирования (последующее заполнение значения) (см. 22.8);
- указать какие-либо необходимые реверсии битов для содержания пространства кодирования до добавления битов к кодированию, сделанному ранее (см. 22.12).

9.21.13 Имеются признаки кодирования, поддерживающие спецификацию правил кодирования для всех этих шагов.

9.21.14 В реальных случаях только некоторые (или никакие!) из этих признаков кодирования могут иметь неиспользуемые значения и должны работать по умолчанию, если они не указаны (см. в D.1.3 пример определения кодирования для целого числа, имеющего выравнивание к правой стороне в фиксированном двухоктетном поле, которое начинается на границе октета).

9.22 Приложение кодирований

9.22.1 Приложение кодирований (правил кодирования) к структурам кодирования является ключевой частью работы ECN, но сильно отличается от определения правил кодирования. Окончательное приложение кодирований (к структуре кодирования, генерируемой из определения типа АСН.1) происходит только внутри модуля компоновки кодирования, но приложение кодирований к полям структуры кодирования может использоваться в определении кодирований для более широкой структуры кодирования.

9.22.2 Кодирования прикладываются путем ссылки на набор объектов кодирования (или на одиночный объект кодирования). Такое приложение может происходить в EDM в определении объектов кодирования для какого-нибудь класса (включая объекты кодирования для генерируемой структуры кодирования и для определяемой пользователем структуры кодирования). Такое приложение в EDM является просто определением большего числа объектов кодирования для этого класса кодирования. Определяющее приложение к реальному типу происходит только в ELM.

9.22.3 Когда прикладывается набор объектов кодирования, он всегда дает в результате полную спецификацию кодирования для классов кодирования, к которым эти объекты прикладываются. Если в каком-либо заданном приложении требуются кодирования для классов кодирования (имеющихся внутри кодируемой структуры кодирования), для которых нет объектов кодирования в приложенном наборе, то это будет ошибкой (см. 13.2.11).

Примечание — Если бы даже спецификация правил кодирования была завершена, точная форма реального кодирования (например, наличие или отсутствие предварительного заполнения пространства кодирования или влияние значений границ, указанных в правилах кодирования) может быть определена только в случаях, когда определение кодирования применяется к типу АСН.1 верхнего уровня.

9.22.4 Имеются два исключения к 9.22.3. Первое исключение будет в случае, когда механизм параметризации (похожий на АСН.1) используется для определения параметризованного объекта кодирования. В этом случае полное кодирование будет определено только после установки реальных параметров. Второе исключение будет в случае, когда объект кодирования определен для конструктора кодирования (**#CONCATENATION, #ALTERNATIVES, #REPETITION, #SEQUENCE** и т. п.). В этом последнем случае правила кодирования, связанные с классом кодирования, просто определяют правила, связанные с аспектами структурирования. Полная спецификация кодирования для структуры кодирования, использующей эти классы кодирования, будет требовать также правил для кодирования компонентов этой структуры кодирования.

Примечание — Здесь имеется различие между объектами кодирования класса **#SEQUENCE** (конструктора кодирования) и объектов кодирования для неявно генерируемой структуры кодирования «**#My-Type**», которая определяется с использованием типа АСН.1 **SEQUENCE**. Последняя не является конструктором кодирования, а объекты кодирования этого класса будут обеспечивать полные правила кодирования для кодирования значений типа «**My-Type**».

9.23 Комбинированный набор объектов кодирования

9.23.1 Чтобы обеспечить полное кодирование, пользователь ECN может выдать первичный набор объектов кодирования и второй набор объектов кодирования, введенный зарезервированными словами **COMPLETED BY**.

9.23.2 Прилагаемый набор объектов кодирования определяется как комбинированный набор объектов кодирования, формируемый путем добавления к первому набору объектов кодирования некоторого класса кодирования, для которого у первого класса не хватает некоторого объекта кодирования, а второй набор содержит его (см. 13.2). Набором для использования с **COMPLETED BY** часто является предопределенный набор **PER-BASICUNALIGNED** (см. в D.1.17 пример приложения комбинированного набора объектов кодирования).

9.23.3 Когда набор объектов кодирования может содержать только один объект кодирования, например для класса **#SEQUENCE-OF**, он может также содержать объект кодирования, например для класса **#Special-sequence-of,** который определяется как «**#Special-sequence-of ::= #SEQUENCE-OF**». Явно генерируемая структура кодирования может содержать в своем определении как класс **#SEQUENCE-OF**, так и класс **#Specialsequence-of**. Таким способом один комбинированный набор объектов кодирования может быть приложен для получения стандартных кодирований для некоторых исходных конструкций **SEQUENCE OF** и специализированных кодирований для других.

9.24 Точка приложения

9.24.1 В любом заданном приложении кодирований имеется определенная стартовая точка (для ELM — это генерируемая структура или структуры кодирования верхнего уровня, к которым прикладываются кодирования). Это называется «начальной точкой приложения» для структуры, которая кодируется модулем ELM.

9.24.2 Комбинированный набор объектов кодирования прикладывается к генерируемой структуре кодирования, и это будет кодированиями, определенными для абстрактных значений этой структуры кодирования, которая кодирует абстрактные значения типа ACH.1.

9.24.3 Если в комбинированном наборе объектов кодирования имеется объект кодирования, который соответствует некоторому классу кодирования с битовым полем (первоначально — некоторой генерируемой структуре кодирования) в точке приложения, то он прикладывается, и этот процесс заканчивается. В остальных случаях класс в точке приложения «расширяется» с помощью разыменования. Это расширение с помощью разыменования будет продолжаться до нахождения объекта кодирования или достижения простейшего класса. Если классом в точке приложения является конструктор кодирования и имеется объект кодирования для этого конструктора кодирования (**#CHOICE**, **#SEQUENCE**, **#SEQUENCE-OF** и др.), то он прикладывается, а точка приложения затем переходит к другому компоненту (в виде параллельного действия).

9.24.4 В более сложном случае может быть класс **#OPTIONAL**, следующий за некоторым классом компонента (и классом **#TAG**, предшествующим ему). Точка приложения переходит сначала к **#OPTIONAL**, и объект кодирования этого класса может заменить этот компонент (см. 9.16.9). Затем точка приложения переходит к тегу, а в конце — к самому компоненту.

9.25 Условные кодирования

9.25.1 Уже упоминался класс кодирования **#TRANSFORM** как средство выражения простых арифметических действий над целочисленными значениями (см. 9.17.3). Этот класс кодирования, однако, играет более фундаментальную роль в спецификации кодирований для некоторых простейших классов. Как правило, спецификация кодирований для многих предопределенных типов ACH.1 является двух- или трехстадийным процессом, использующим объекты кодирования класса **#TRANSFORM** и, например, класса **#CONDITIONAL-INT** или **#CONDITIONAL-REPETITION**.

9.25.2 Классы кодирования **#TRANSFORM**, **#CONDITIONAL-INT** и **#CONDITIONAL-REPETITION** ограничены в своем использовании. Объекты кодирования для этих классов могут быть определены только с использованием либо синтаксиса из раздела 24, подразделов 23.7 и 23.14 соответственно, либо не-ECN определения объекта кодирования, причем они могут использоваться только в определениях других объектов кодирования. Они не могут появляться в наборах объектов кодирования или применяться прямо для кодирования полей структур кодирования (см. 18.1.7).

9.25.3 Спецификация кодирования для классов кодирования в категории «целочисленная» выполняется следующим образом. Определяются кодирования (класса кодирования **#CONDITIONAL-INT**)

для конкретного условия границ, указывающего размер контейнера (и как он разделяется), преобразование целого числа в биты (с использованием либо поразрядного дополнения до двух, либо кодирования положительных целых чисел) и способ вставления этих битов в контейнер (примером условия границ является наличие верхней границы и неотрицательной нижней границы). Это называется условным кодированием. Кодирование класса из категории «целочисленная» определяется в виде списка этих условных кодирований, а реальное кодирование, применяемое в любом заданном случае, будет тем, которое находится в начале списка и чье условие границ соблюдается (см. пример в D.1.5.4).

9.25.4 Спецификация кодирования для классов кодирования в категории «повторение» использует класс кодирования **#CONDITIONAL-REPETITION**, который определяет способ разделения пространства кодирования для повторяющихся элементов и способ размещения в нем повторяющихся кодирований для заданного условия диапазонов, опять образуя условное кодирование. Как и при кодировании классов в категории «целочисленная», окончательное кодирование определяется в виде упорядоченного списка условных кодирований.

9.25.5 Спецификация кодирования для классов кодирования в категории «цепочка октетов» выполняется следующим образом. Во-первых, определяются объекты кодирования **#TRANSFORM** для отображения одиночного октета в саморазграничивающую цепочку битов. Во-вторых, определяется один или несколько объектов кодирования **#CONDITIONAL-REPETITION** (при конкретных условиях диапазона размера) для взятия каждой цепочки битов (преобразованной из октета цепочки октетов) и конкатенации (сцепления) их в размеченный контейнер (определение таких объектов кодирования нехарактерно для кодирования **#OCTETS**). Окончательное кодирование класса в категории «цепочка октетов» определяется в виде упорядоченного списка объектов кодирования **#CONDITIONAL-REPETITION** (см. пример в D.1.8.2).

9.25.6 Спецификации кодирования для классов кодирования в категории «цепочка битов» выполняются следующим образом. Во-первых, определяются объекты кодирования **#TRANSFORM** для отображения одиночного бита в цепочку битов аналогично кодированию целого числа в биты, но в этом случае отображение бита должно быть в саморазграничивающую цепочку. Во-вторых, определяется один или несколько объектов кодирования **#CONDITIONAL-REPETITION** для повторения битов (это могут быть те же объекты кодирования, которые были определены для использования с классом кодирования в категориях «повторение» или «цепочка октетов»). Наконец, кодирование класса в категории «цепочка битов» определяется в виде упорядоченного списка объектов кодирования **#CONDITIONAL-REPETITION** (см. пример в D.1.7.3).

9.25.7 Спецификации кодирования классов кодирования в категории «цепочка знаков» выполняются следующим образом. Во-первых, определяются объекты кодирования **#TRANSFORM** для отображения одиночного знака в саморазграничивающую цепочку битов, используя несколько возможных механизмов для определения кодирования знака и используя, где подходит, действующее разрешенное ограничение алфавита. Во-вторых, определяется один или несколько объектов кодирования **#CONDITIONAL-REPETITION** и, наконец, определяется кодирование класса в категории «цепочка знаков» в виде упорядоченного списка этих объектов (см. пример в D.1.9.2).

9.26 Другие условия для применения кодирований

9.26.1 Существует целый ряд различных условий, которые могут быть проверены для того, чтобы выбрать соответствующий тип кодирования. К ним относятся фактическое значение и диапазон границ.

9.26.2 Также можно требовать, чтобы был удовлетворен целый ряд условий.

9.26.3 Для проверки выполнения условия используется либо одно перечислимое значение (такое как «bounded-without-negatives»), которое содержит всю проверку в спецификации одного перечислимого, или три перечислимых.

9.26.4 Если используются три, то первый определяет (путем перечисления) элемент, который проходит проверку (например, «test-upper-bound»), второй — природу проверки (например, «greater-than») и третий предоставляет целочисленное значение для проверки.

9.27 Управление кодированием для открытого типа

9.27.1 Открытые типы часто предоставляют средства расширяемости за счет добавления новых значений для идентификационных полей и новых типов для открытого типа в последующих версиях (и часто доступных для специализированных расширений).

9.27.2 Эти две возможности означают, что декодер может быть использован для декодирования открытого типа при том, что конкретная имплементация не знает ничего про тип, который был закодирован в нем.

9.27.3 Поддержка кодирования, предоставляемая для открытого типа, является в точности той же, что и для большинства других классов в bitfield категории, но с дополнительной возможностью указания, что другой набор объектов кодирования применяется к типу, который кодируется в открытый тип.

Примечание — Это является признанием того, что многие протоколы выбирают для использования другой стиль кодирования (часто на основе type-length-value подхода) для типа, содержащегося в открытом типе, сохраняя при этом более компактный стиль кодирование для полей сообщения, содержащих открытый тип.

9.27.4 Модель, используемая для декодирования открытого типа, определяет, что декодер не будет знать, какой тип заполняет открытый тип (таблицы и реляционные ограничения не видны ни для PER, ни для ECN), но приложение может быть в состоянии определить это из какого-либо другого поля в протоколе, или в предыдущем сообщении, или (для специализированных дополнений) на основе адреса вызывающей стороны.

9.27.5 Модель является такой потому, что, разобравшись с любым заданным предварительным заполнением и определив пространство кодирования и любое предварительное и последующее заполнение значения, декодер запросит у приложения тип, который был закодирован (в случае инструментов приложение почти наверняка будет иметь предварительно настроенный инструмент со списком известных типов, которые могут присутствовать, и просто вернет указатель на один из них). Затем декодирование может продолжиться в обычном режиме.

9.27.6 Однако приложение может сказать «unknown» (см. 9.27.4), тогда декодеру нужно узнать, как определить конец такого неизвестного кодирования. Этого можно добиться за счет разрешения спецификатору ECN предоставить структуру кодирования и (опционально) объект кодирования, установленный для использования с ней, что должно использоваться декодерами для декодирования неизвестных типов. Необходимый синтаксис приведен в разделе 23.

Примечание — Примером такой структуры кодирования может быть структура, определяющая широко известное «Type, Length, Value» кодирование, конец которого может быть определен без знания закодированного типа.

9.28 Изменения международных стандартов об АСН.1

9.28.1 В настоящем стандарте приведены ссылки на другие стандарты по АСН.1 с целью определения их нотации без повторения. Чтобы такие ссылки были корректными, необходимо расширить семантику этих нотаций (например, раздел об импортах, определение параметризации и информационного объекта) с целью признания справочных номеров классов кодирования, объектов кодирования и другого, что составляет часть ECN.

9.28.2 Имеется также необходимость в расширении нотации класса информационных объектов для разрешения полей, которые являются упорядоченными списками значений или объектов, а не просто неупорядоченными наборами объектов, с целью позволить использование этой нотации в определении синтаксиса ECN при определении объектов кодирования некоторых классов.

9.28.3 Правила параметризации ослабляются для разрешения использования фиктивного параметра ссылки на объект кодирования (присвоенной в операторе присвоения) в качестве реального параметра ссылки на класс кодирования, который руководит нотацией, определяющей справочное имя объекта кодирования. В частности, параметризованный класс кодирования может использоваться в качестве руководителя в операторе присвоения объекта кодирования (см. С.2, 8.4) с реальным параметром, являющимся фиктивным параметром объекта кодирования, который определяется.

9.28.4 Эти изменения к другим стандартам по АСН.1 определяются в приложениях А—С и предназначены только для целей настоящего стандарта.

10 Определения классов кодирования, объектов кодирования и наборов объектов кодирования

10.1 Многие продукции внутри настоящего стандарта нуждаются в определении классов кодирования, объектов кодирования или наборов объектов кодирования.

10.2 Для каждого из них имеются пять способов выполнения идентификации:

a) использование простого справочного имени;

b) использование предопределенного справочного имени (неприменимо к объектам кодирования, так как нет предопределенных объектов кодирования);

c) использование внешней ссылки (называемой также полностью определенным именем);

d) использование параметризованной ссылки;

e) инлайновое (подставляемое) определение.

Примечание — Форма параметризованной ссылки может использоваться с простым справочным именем или с внешней ссылкой (см. С.3).

10.3 Имеются продукции (или лексические единицы) для всех этих средств идентификации. Имеются также продукции, которые допускают несколько альтернатив. Эти лексические единицы или имена продукций используются, где возможно, в других продукциях и определяются в остальной части этого раздела.

10.4 Лексическими единицами для использования простого справочного имени являются:

класс кодирования «**encodingclassreference**» (см. 8.3);

объект кодирования «**encodingobjectreference**» (см. 8.1);

набор объектов кодирования «**encodingobjectsetreference**» (см. 8.2).

10.4.1 Имя «encodingclassreference»:

a) присваивается классу кодирования в «EncodingClassAssignment» (см. раздел 16), либо

b) импортируется в EDM из другого EDM, из которого он экспортирован, либо

c) импортируется в качестве имени неявно генерируемой структуры кодирования из модуля ACH.1 (см. 14.11), либо

d) генерируется разделом переименований в EDM (см. раздел 15).

Примечание — Только классы, которые образуют структуры кодирования, могут импортироваться в ELM (см. 12.1.8).

10.4.2 Имя «encodingclassreference» не должно импортироваться из EDM (как указано в 10.4.1), если:

a) оно не определено в указанном модуле или импортировано в него, а этот модуль не имеет раздела экспортов.

Примечание 1 — Если указанный модуль не имеет раздела экспортов, то это эквивалентно экспортированию всего;

b) оно не определено в указанном модуле или импортировано в него, появившись в виде символа в разделе экспортов этого модуля;

c) оно не является одним из справочных имен, явно генерируемых разделом переименований в модуле, из которого оно было импортировано.

Примечание 2 — Неявно генерируемые структуры кодирования могут импортироваться только из модулей ACH.1, которые генерируют их.

10.4.3 Ссылка на неявно генерируемую структуру кодирования никогда не появляется в разделе экспортов любого модуля ACH.1, но всегда может быть импортирована из любого модуля ACH.1, в котором соответствующий тип определен или в который он экспортирован.

10.4.4 Ссылка на явно генерируемую структуру кодирования (которая автоматически экспортируется разделом переименований, генерирующим ее) не должна появляться в разделе экспортов модуля EDM, в котором она генерируется, но любое ее использование в другом EDM или в ELM требует ее импортирования из этого модуля EDM.

10.4.5 Имя «encodingobjectreference»:

a) присваивается объекту кодирования в «EncodingObjectAssignment» (см. раздел 17) в EDM либо

b) импортируется в EDM или ELM из другого EDM, в котором оно присвоено объекту кодирования или импортировано.

10.4.6 Имя «encodingobjectreference» не должно импортироваться из EDM, если указанный модуль имеет раздел экспорта, а «encodingobjectreference» не появляется в виде символа в этом разделе экспорта.

Примечание — Если указанный модуль не имеет раздела экспортов, то это эквивалентно экспортированию всего.

10.4.7 Имя «encodingobjectsetreference»:

a) присваивается набору объектов кодирования в «EncodingObjectSetAssignment» (см. раздел 18) в EDM либо

b) импортируется в EDM или ELM из другого EDM, в котором оно присвоено набору объектов кодирования или из которого оно импортировано.

10.4.8 Имя «encodingobjectsetreference» не должно импортироваться из EDM, если указанный модуль имеет раздел экспортов, а «encodingobjectsetreference» не появляется в виде символа в этом разделе экспортов.

Примечание — Если указанный модуль не имеет раздела экспортов, то это эквивалентно экспортированию всего.

10.5 Продукциями для использования предопределенного справочного имени являются:

- класс кодирования **«BuiltinEncodingClassReference»** (см. 16.1.6);
- набор объектов кодирования **«BuiltinEncodingObjectSetReference»** (см. 18.2.1).

10.6 Продукциями для использования внешнего справочного имени являются:

```
ExternalEncodingClassReference ::=
        modulereference "." encodingclassreference |
modulereference "." BuiltinEncodingClassReference
        ExternalEncodingObjectReference ::=
modulereference "." encodingobjectreference
        ExternalEncodingObjectSetReference ::=
modulereference "." encodingobjectsetreference
```

10.6.1 Имя «modulereference» определено в ИСО/МЭК 8824-1, подраздел 12.5 и указывает модуль, на который сделана ссылка в списке импортов в EDM или ELM.

10.6.2 Альтернатива «ExternalEncodingClassReference», содержащая «BuiltinEncodingClassReference», применяется в теле EDM, если, и только если, имеется генерируемая структура кодирования (у которой имя совпадает с именем «BuiltinEncodingClassReference»), которая:

a) неявно определена в модуле ACH.1, на который имеется ссылка «modulereference» (см. 11.4.1), или

b) импортирована в другой EDM, на который имеется ссылка «modulereference», и экспортирована из этого модуля, или

c) генерирована в разделе переименований в другом EDM, на который имеется ссылка «modulereference», или

d) генерирована в этом EDM в разделе переименований, в этом случае «modulereference» должна ссылаться на этот EDM.

Примечание — Имя «BuiltinEncodingClassReference» может появляться в виде «Symbol» в разделе импортов (см. A.1).

10.6.3 Продукции, определенные в 10.6 (кроме указанной в 10.6.2), должны использоваться, если, и только если, соответствующее простое справочное имя было импортировано из модуля, указанного ссылкой «modulereference», и либо

a) идентичные справочные имена были импортированы из разных модулей или были генерированы в разделе переименований в этом EDM, или были как импортированы, так и генерированы, либо

b) простым справочным номером является «BuiltinEncodingClassReference» (см. 10.5), либо

c) выдерживаются оба условия.

10.7 Параметризованной ссылкой является справочное имя, определенное в «ParameterizedAssignment» (см. C.1) и выданное с реальным параметром согласно синтаксису из C.3. Использованными продукциями являются:

- классы кодирования **«ParameterizedEncodingClassAssignment»** (см. C.1);
  «ParameterizedEncodingClass» (см. C.3);
- объекты кодирования **«ParameterizedEncodingObjectAssignment»** (см. C.1);
  «ParameterizedEncodingObject» (см. C.3);
- наборы объектов кодирования
  «ParameterizedEncodingObjectSetAssignment» (см. C.1),
  «ParameterizedEncodingObjectSet» (см. C.3).

10.8 Продукциями, которые позволяют все формы идентификации, являются:

- классы кодирования **«EncodingClass»** (см. 16.1.5);
- объекты кодирования **«EncodingObject»** (см. 17.1.5);
- наборы объектов кодирования **«EncodingObjectSet»** (см. 18.1).

10.9 Продукциями, которые позволяют все формы, кроме инлайнового определения, являются:

- классы кодирования **«DefinedEncodingClass»** и **«DefinedOrBuiltinEncodingClass»;**
- объекты кодирования **«DefinedEncodingObject»**
- наборы объектов кодирования **«DefinedEncodingObjectSet»** и **«DefinedOrBuiltinEncodingObjectSet»**,

за исключением того, что предопределенные классы кодирования и предопределенные наборы объектов кодирования не разрешаются именами «DefinedEncodingClass» и «DefineEncodingObjectSet».

Примечание — Используется также дополнительная продукция «SimpleDefinedEncodingClass». Она определена в С.3 и позволяет только «encodingclassreference» и «ExternalEncodingClassReference».

10.9.1 «DefinedEncodingClass» и «DefinedOrBuiltinEncodingClass» имеют следующий вид:

```
DefinedEncodingClass ::=
        encodingclassreference
      | ExternalEncodingClassReference
      | ParameterizedEncodingClass
DefinedOrBuiltinEncodingClass ::=
        DefinedEncodingClass
      | BuiltinEncodingClassReference
```

10.9.2 «DefinedEncodingObject» имеет следующий вид:

```
DefinedEncodingObject ::=
        encodingobjectreference
      | ExternalEncodingObjectReference
      | ParameterizedEncodingObject
```

10.9.3 «DefinedEncodingObjectSet» и «DefinedOrBuiltinEncodingObjectSet» имеют следующий вид:

```
DefinedEncodingObjectSet ::=
        encodingobjectsetreference
      | ExternalEncodingObjectSetReference
      | ParameterizedEncodingObjectSet
DefinedOrBuiltinEncodingObjectSet ::=
        DefinedEncodingObjectSet
      | BuiltinEncodingObjectSetReference
```

11 Кодирование типов АСН.1

11.1 Общие положения

11.1.1 Для всех типов АСН.1 имеется соответствующая неявно кодируемая структура кодирования. Эта структура кодирования неявно генерируется для каждого присвоения типа АСН.1 и автоматически экспортируется из модуля АСН.1, содержащего это присвоение типа (она должна, однако, импортироваться в модуль EDM, если она должна использоваться). Именем соответствующей структуры кодирования будет имя типа, перед которым ставится знак «#». Эта структура кодирования определяет класс кодирования и называется **неявно генерируемой структурой кодирования**.

11.1.2 Могут быть также одна или несколько **явно генерируемых структур кодирования**. Они генерируются в EDM с помощью раздела переименований.

11.1.3 Кодирование типа АСН.1 формально определяется как результат кодирований, примененных именно к одной структуре кодирования (неявной или явной), генерируемой из типа АСН.1. Кодирования применяются с помощью операторов в ELM (см. раздел 12), используя объекты кодирования

из комбинированного набора объектов кодирования. ELM применяют кодирования не более чем к одной из генерируемых структур кодирования, соответствующих заданному типу АСН.1.

11.1.4 Неявно генерируемая структура кодирования определяется первой упрощенной и расширенной нотацией АСН.1 (определенной в 11.3), а затем определяется отображением типов АСН.1, конструкторов типов и имен компонентов в соответствующие предопределенные классы кодирования, конструкторы кодирования и имена полей структуры кодирования.

11.1.5 Явно генерируемая структура кодирования определяется путем выполнения указанных изменений к неявно генерируемой структуре кодирования при помощи раздела переименований.

11.1.6 Каждое поле генерируемой структуры кодирования связано с полем абстрактных значений соответствующего типа и с информацией, относящейся к ограничениям и полученной из определения типа АСН.1 (см. 11.4.2). Кодирования абстрактных значений генерируемой структуры кодирования определяются в виде кодирований для соответствующих абстрактных значений исходного типа АСН.1.

11.1.7 Этот раздел 11 указывает:

a) предопределенные классы кодирования, которые используются при определении неявно кодируемых структур кодирования, соответствующих типам АСН.1 (см. 11.2).

Примечание — Пункт 16.1.14 определяет дополнительные классы, которые используются при описании определенных пользователем структур кодирования;

b) преобразования синтаксиса АСН.1 (упрощение и расширение) перед выработкой неявно генерируемой структуры (см. 11.3).

c) неявно генерируемую структуру кодирования для любого типа АСН.1 (см. 11.4).

11.2 Предопределенные классы кодирования, используемые для неявно генерируемых структур кодирования

11.2.1 Классы кодирования, используемые для неявно генерируемых структур кодирования, и типы АСН.1 или конструкторы, которым они соответствуют, перечислены в таблице 2.

Таблица 2 — Классы кодирования для нотации АСН.1

Нотация АСН.1	Класс кодирования	Простейший класс
BIT STRING	**#BIT-STRING**	**#BITS**
BOOLEAN	**#BOOLEAN**	**#BOOL**
CHARACTER STRING	**#CHARACTER-STRING**	**Определен с помощью #SEQUENCE**
CHOICE	**#CHOICE**	**#ALTERNATIVES**
EMBEDDED PDV	**#EMBEDDED-PDV**	**Определен с помощью #SEQUENCE**
ENUMERATED	**#ENUMERATED**	**#INT**
EXTERNAL	**EXTERNAL#EXTERNAL**	**Определен с помощью #SEQUENCE**
INTEGER	**#INTEGER**	**#INT**
NULL	**#NULL**	**#NUL**
OBJECT IDENTIFIER	**#OBJECT-IDENTIFIER**	**#OBJECT-IDENTIFIER**
OCTET STRING	**#OCTET-STRING**	**#OCTETS**
нотация открытого типа	**#OPEN-TYPE**	**#OPEN-TYPE**
OPTIONAL	**#OPTIONAL**	**#OPTIONAL**
REAL	**#REAL**	**#REAL**
RELATIVE-OID	**#RELATIVE-OID**	**#OBJECT-IDENTIFIER**
SEQUENCE	**#SEQUENCE**	**#CONCATENATION**
SEQUENCE OF	**#SEQUENCE-OF**	**#REPETITION**
SET	**#SET**	**#CONCATENATION**
SET OF	**#SET-OF**	**#REPETITION**
TIME	**#TIME**	**#TIME**
DATE	**#DATE**	**#TIME**
TIME-OF-DAY	**#TIMEh-OF-DAY**	**#TIME**
DATE-TIME	**#DATE-TIME**	**#TIME**
DURATION	**#DURATION**	**#TIME**
GeneralizedTime	**#GeneralizedTime**	**#CHARS**

Окончание таблицы 2

Нотация АСН.1	Класс кодирования	Простейший класс
UTCTime	**#UTCTime**	**#CHARS**
ObjectDescriptor	**#ObjectDescriptor**	**#CHARS**
BMPString	**#BMPString**	**#CHARS**
GeneralString	**#GeneralString**	**#CHARS**
GraphicString	**#GraphicString**	**#CHARS**
IA5String	**#IA5String**	**#CHARS**
NumericString	**#NumericString**	**#CHARS**
PrintableString	**#PrintableString**	**#CHARS**
TeletexString	**#TeletexString**	**#CHARS**
UniversalString	**#UniversalString**	**#CHARS**
UTF8String	**#UTF8String**	**#CHARS**
VideotexString	**#VideotexString**	**#CHARS**
VisibleString	**#VisibleString**	**#CHARS**
Текстуально изложенная нотация тега	**#TAG**	**#TAG**

11.2.2 В столбце 1 приведена нотация АСН.1, которая заменяется классом кодирования в неявно генерируемой структуре кодирования. В столбце 2 приводится класс кодирования, который заменяет нотацию столбца 1. В столбце 3 приведен простейший класс, из которого получен класс столбца 2.

11.3 Упрощение и расширение нотации АСН.1 для целей кодирования

11.3.1 В ECN предполагается, что определенные синтаксические конструкции АСН.1 будут расширены (или сокращены) в эквивалентные или упрощенные конструкции.

Примечание — Типы, определенные упрощенными конструкциями, способны переносить такой же набор абстрактных значений, что и исходные синтаксические структуры АСН.1, и эти абстрактные значения отображаются в упрощенные конструкции.

11.3.2 Расширениями или упрощениями синтаксических конструкций АСН.1 являются следующие:

a) полностью определенные в разделе 11.3.4, или

b) упомянутые в разделах «См. перечисление b) 11.3.2» и полностью определенные в ИСО/МЭК 8824-1 (включая приложение С) со всеми опубликованными поправками и техническими опечатками, или

c) упомянутые в разделах «См. перечисление c) 11.3.2» и полностью определенные в ИСО/МЭК 8824-2 со всеми опубликованными поправками и техническими опечатками, или

d) упомянутые в разделах «См. перечисление d) 11.3.2» и полностью определенные в ИСО/МЭК 8824-4 со всеми опубликованными поправками и техническими опечатками.

11.3.3 Синтаксические конструкции АСН.1, удаленные ниже при расширении и упрощении, далее в настоящем стандарте не упоминаются.

11.3.4 Следующие расширения и упрощения будут применимы ко всем модулям АСН.1.

11.3.4.1 Следующие трансформации не являются рекурсивными и, следовательно, применяются только один раз:

a) все «ValueSetTypeAssignment» заменяются их эквивалентами «TypeAssignment» с ограничениями на подтипы [см. перечисление b) 11.3.2];

b) конструкция АСН.1 **INSTANCE OF** расширяется в ее эквивалентный тип «последовательность» [см. перечисление c) 11.3.2];

c) «TypeFromObject» заменяется типом, на который дана ссылка [см. перечисление c) 11.3.2];

d) «ValueSetFromObjects» заменяется типом, на который дана ссылка [см. перечисление c) 11.3.2];

e) если экземпляр нотации тега АСН.1 дан текстуально, а за ним следуют один или несколько дальнейших экземпляров нотации тега АСН.1, то второй и последующие экземпляры нотации тега отбрасываются.

Примечание — Это похоже на правила неявного тегирования в АСН.1, но применяется для любой среды тегирования. Многократное тегирование одного и того же типа, тем не менее, возможно путем использования справочных имен типа.

11.3.4.2 Следующие трансформации применяются рекурсивно в указанном порядке до достижения некоторой фиксированной точки:

a) вся параметризация ACH.1 полностью разрешается путем замены реальных параметров на холостые параметры [см. перечисление d) 11.3.2].

Примечание — Это означает, что если нотация типа ACH.1 содержит экземпляр параметризованного типа ACH.1, то этот экземпляр становится инлайновым определением;

b) все «ComponentsOf» расширяются до их полных форм [см. перечисление b) 11.3.2];

c) все использования «SelectionType» должны быть разрешены [см. перечисление b) 11.3.2].

11.3.4.3 Затем применяются следующие трансформации:

a) списки поименованных номеров в определениях целочисленных типов удаляются. Поименованные номера для ECN не видны. ECN видит один класс **#INTEGER** [возможно, с границами, определенными в перечислении c) 11.3.4.3];

b) списки поименованных битов в определениях цепочек битов удаляются. Поименованные биты для ECN не видны;

c) все нотации ограничений, невидимых в PER, кроме ограничения на содержимое, удаляются. Ограничения, видимые в PER, должны разрешаться для обеспечения следующих значений, которые могут быть указаны в определении правил кодирования:

i) верхняя граница целых чисел и перечислений;

ii) нижняя граница целых чисел и перечислений;

iii) ограничения на действующий разрешенный алфавит PER и на действующий размер (см. ИСО/МЭК 8825-2, пункт 10.3);

d) если имеется ограничение на содержимое с конструкцией **CONTAINING**, то наличие ограничения на содержимое, тип содержимого и присутствие или отсутствие раздела **ENCODED BY** получают признаки, связанные с абстрактными значениями такого ограниченного типа «цепочка октетов» или «цепочка битов», и это ограничение затем сбрасывается. Если имеется ограничение на содержимое без конструкции **CONTAINING**, то оно не видно для ECN и сбрасывается.

Примечание — Когда определяются кодирования для значений со связанным ограничением на содержимое, для кодирования типа содержимого может выдаваться отдельный комбинированный набор объектов кодирования. Это может быть определено по выбору разработчика, с отменой или без отмены существующего **ENCODED BY** (см. 11.3 и 13.2);

e) все тегирования, которые даны в нотации ACH.1 не текстуально, должны игнорироваться при отображении в структуры кодирования, но (для моделирования кодирований BER и процедур PER) полный список тегов типа получает признак поля структуры кодирования, в которую отображаются соответствующие значения;

f) текстуально представленная нотация тегов имеет класс удаленного тега [см. также перечисление e) 11.3.4.1];

g) значение **«DEFAULT»** заменяется на **«OPTIONAL-ENCODING #OPTIONAL»**, а безусловное значение (по умолчанию) связывается с полем структуры, в которую отображен компонент ACH.1;

h) **OPTIONAL** заменяется на **«OPTIONAL-ENCODING #OPTIONAL»**;

i) **T61String** заменяется на **#TeletexString**;

j) **ISO646String** заменяется на **#VisibleString**.

11.3.4.4 Далее применяются следующие трансформации:

a) выполняется автоматическое распределение значений по нумерациям (если это применимо). Синтаксис **ENUMERATED** заменяется на класс кодирования **#ENUMERATED** с установкой верхней и нижней границы [см. перечисление c) 11.3.4.3].

Примечание 1 — Класс **#ENUMERATED** разыменует к классу **#INT** (см. 11.2.2), а нумерации отображаются в ограниченные целочисленные значения класса. Реальные имена нумераций для ECN не видны;

b) все появления «ObjectClassFieldType» (см. ИСО/МЭК 8824-2, раздел 14), которые ссылаются на поле типа, поле значения переменного типа или поле набора значений переменного типа, заменяются на класс кодирования **#OPEN-TYPE** [см. перечисление c) 11.3.2];

c) маркеры растяжимости и квадратные скобки в последовательности, наборе и конструкциях выбора удаляются, но (для моделирования кодирований BER и процедур PER) идентификация компонента как части корня или версии 1 и 2 и т. д. получает признак компонента, а наличие маркера растяжимости получает признак класса, в который конструктор отображает;

d) маркер растяжимости в ограничениях удаляется, но наличие маркера растяжимости получает признак класса, а если абстрактное значение находится в корне или в расширении, получает признак абстрактного значения.

Примечание 2 — Признаки, упомянутые в перечислениях c) и d), опрашиваются благодаря не-ECN определению объектов кодирования в данной версии настоящего стандарта. Полная поддержка расширяемости, как ожидается, будет обеспечена в последующей версии настоящего стандарта.

11.3.5 С помощью этих трансформаций все конструкции, относящиеся к типам ACH.1, получают соответствующие классы кодирования, перечисленные в таблице 2. Неявно генерируемые структуры кодирования будут конструироваться путем отображения относящихся к типам ACH.1 конструкций из столбца 1 в классы из столбца 2 таблицы 2 (как описано в 11.4).

11.4 Неявно генерируемые структуры кодирования

11.4.1 Для каждого определения типа ACH.1 имеется неявно генерируемая структура кодирования с именем, сконструированным из справочного имени типа ACH.1 путем приставления впереди знака «**#**». Если для неявно генерируемой структуры кодирования требуется полностью определенное имя, то полностью определенное имя должно иметь «ModuleIdentifier» модуля ACH.1, содержащего определение типа (пример неявно генерируемой структуры приведен в D.1.9.2).

Примечание — Неявно генерируемая структура генерируется и экспортируется для каждого типа ACH.1 в модуле ACH.1 независимо от того, присутствует ли этот тип в разделе **EXPORTS**.

11.4.2 Неявно генерируемая структура кодирования имеет такую же структуру, что и определение типа ACH.1, а именно:

a) идентификаторы компонентов ACH.1 отображаются в имена полей структур кодирования;

b) нотация ACH.1 из столбца 1 таблицы 2 отображается в предопределенные классы кодирования из столбца 2 таблицы 2.

Примечание — Первый текстуально представленный тег отображается в конструкцию «**[#TAG]**» неявно генерируемой структуры. Неявно генерируемая структура не содержит конструкций «**[#TAG]**» для последующих текстуально представленных тегов;

c) компоненты ACH.1 «DefinedType» отображаются в имя класса кодирования, полученное из имени типа путем добавления знака «**#**». Если тип импортирован в модуль ACH.1, то любая нотация «ExternalEncodingClassReference» к соответствующему классу в неявно генерируемой структуре должна указывать модуль ACH.1, который содержит определение указанного типа.

Примечание — Если получающийся класс является именем предопределенного класса кодирования, то все ссылки на него в разделе переименований или в ELM будут использовать нотацию «ExternalEncodingClass Reference»;

d) абстрактные значения отображаются из поля определения типа в соответствующее поле структуры кодирования;

e) верхняя и нижняя границы для целочисленного и перечисленного типов, все действующие ограничения на размер и действующие ограничения на разрешенный алфавит (см. ИСО/МЭК 8825-2, пункт 10.3) отображаются из определения типа в соответствующее поле структуры кодирования;

f) номер тега для первого текстуально представленного тега отображается в класс **#TAG**.

11.4.3 Три следующие неявно генерируемые структуры вырабатываются и экспортируются из всех модулей ACH.1. Эти структуры имеют имена **#CHARACTER-STRING**, **#EMBEDDED-PDV** и **#EXTERNAL**, а структуры, к которым они разыменуют, являются неявно генерируемыми структурами, соответствующими связанным типам **CHARACTER STRING**, **EMBEDDED PDV** и **EXTERNAL**, указанным соответственно в ИСО/МЭК 8824-1, пункты 44.5, 36.5 и 37.5.

11.4.4 Все неявно генерируемые структуры кодирования могут кодироваться предопределенными наборами объектов кодирования (см. 18.2) и будут обеспечивать те же кодирования, которые определены в соответствующих стандартах для таких кодирований, применяемых к типам ACH.1.

12 Модуль компоновки кодирования (ELM)

Примечание — В ECN имеются две продукции верхнего уровня: «ELMDefinition», определяемая в настоящем разделе, и «EDMDefinition», определяемая в разделе 14. Они определяют синтаксис для описания модуля ELM и модулей EDM соответственно.

12.1 Структура ELM

12.1.1 Продукцией «ELMDefinition» является:

```
ELMDefinition ::=
        ModuleIdentifier
        LINK-DEFINITIONS
        "::="
        BEGIN
        ELMModuleBody
        END
```

12.1.2 В любом заданном приложении ECN должен быть точно один ELM, который определяет кодирование всех сообщений, используемых в этом приложении.

Примечание — Тип (типы) ACH.1, определяющие «сообщения», часто называют «типами верхнего уровня».

12.1.3 Продукция «ModuleIdentifier» и ее семантика определены в ИСО/МЭК 8824-1, подраздел 13.1.

12.1.4 «ModuleIdentifier» обеспечивает однозначную идентификацию любого модуля в совокупности всех модулей ACH.1, ELM и EDM.

12.1.5 Продукцией «ELMModuleBody» является:

```
ELMModuleBody ::=
        Imports ?
        EncodingApplicationList
EncodingApplicationList ::=
        EncodingApplication
        EncodingApplicationList ?
```

12.1.6 Продукция «Imports» и ее семантика определены в ИСО/МЭК 8824-1, пункты 13.1, 13.16 и 13.17 с учетом изменений из А.1.

12.1.7 Все справочные имена, используемые в «ELMModuleBody», импортируются в ELM.

Примечание — Это является более строгим требованием, чем предъявляемое к модулям ACH.1. В модулях ACH.1 внешние ссылки могут использоваться для типов и значений, которые не были импортированы. В модуле ELM (и в модуле EDM) внешние ссылки могут использоваться только для классов кодирования, которые были указаны в разделе импортов. Цель внешних ссылок — это устранение совпадений между импортируемыми именами и предопределенными именами либо между двумя идентичными именами, импортированными из разных модулей.

12.1.8 «Imports» делает доступным внутри ELM:

a) неявно генерируемые структуры кодирования из модуля ACH.1;

b) явно генерируемые структуры из модуля EDM.

Примечание — Когда ELM импортирует явно генерируемую структуру кодирования из EDM, разделы переименований в других EDM не влияют на кодирование этой структуры (см. 15.2.4);

c) объекты и наборы объектов кодирования из модуля EDM.

12.1.9 «EncodingApplicationList» должен содержать по крайней мере одно «EncodingApplication», так как единственной функцией ELM является применение кодирования.

12.2 Типы кодирования

12.2.1 Продукцией «EncodingApplication» является:

```
EncodingApplication ::=
        ENCODE
        SimpleDefinedEncodingClass "," +
        CombinedEncodings
```

12.2.2 «EncodingApplication» определяет кодирование типов ACH.1, соответствующих классам «SimpleDefinedEncodingClass», которые будут генерируемыми структурами кодирования. Кодирование

этих типов определяется «CombinedEncodings», применяемым к генерируемым структурам кодирования, как указано в 13.2.

Примечание — Для ELM будет обычным кодировать одиночный тип одиночного модуля, но если кодируются несколько типов, то поставщики оборудования могут (но не обязательно) предполагать, что это неявно указывает на типы верхнего уровня, нуждающиеся в поддержке в генерируемых структурах данных.

12.2.3 Кодирования, применяемые к генерируемым структурам кодирования, соответствующим типу ACH.1, который определен в каком-либо модуле ACH.1, предназначены для использования только этого типа в виде прикладных сообщений. Они не имеют значения при кодировании этого типа, когда ссылка на них сделана другими типами или когда они экспортированы из этого модуля ACH.1 и импортированы в другой модуль ACH.1.

12.2.4 Кодированием типа в ограничениях содержимого является то, которое указано объектом кодирования, приложенным к этому ограниченному классу в категории «цепочка октетов» или «цепочка битов», а также может быть любым комбинированным набором объектов кодирования или может быть комбинированным набором объектов кодирования, который был приложен к классу, содержащемуся в этой категории «цепочка октетов» или «цепочка битов».

12.2.5 В ELM должно применяться только одно кодирование к одному типу ACH.1.

Примечание — Правила применения кодирований (определяемые в разделе 13) означают, что «EncodingApplication» полностью описывает кодирование типа, если он не содержит экземпляра ограничения содержимого.

13 Применение кодирований

13.1 Общие положения

13.1.1 Кодирования применяются в ELM к генерируемой структуре (или независимо, к нескольким генерируемым структурам) с помощью «CombinedEncodings», определяемого в 13.1.3. Этот раздел вместе с 13.2 определяет применение «CombinedEncodings» к генерируемой структуре кодирования.

13.1.2 В ELM приложение выполняется к генерируемым структурам кодирования, указанным в «EncodingApplication». Последующие разделы определяют также приложение кодирований ко всем или к части определений произвольных структур кодирования. Этот раздел применим в обоих случаях.

13.1.3 Продукцией «CombinedEncodings» является:

```
CombinedEncodings ::=
        WITH
        PrimaryEncodings
        CompletionClause ?
CompletionClause ::=
        COMPLETED BY
        SecondaryEncodings
PrimaryEncodings ::= EncodingObjectSet
SecondaryEncodings ::= EncodingObjectSet
```

13.1.4 «EncodingObjectSet» определяется в 18.1.1.

13.1.5 Использование «CombinedEncodings» определяется в 13.2.

13.2 Комбинированный набор объектов кодирования и его применение

13.2.1 Комбинированный набор объектов кодирования формируется из продукции «CombinedEncodings» (см. 13.1.3) следующим образом.

13.2.2 Если не имеется «CompletionClause», то «PrimaryEncodings» формирует комбинированный набор объектов кодирования.

13.2.3 В остальных случаях:

a) все объекты кодирования из «PrimaryEncodings» помещаются в комбинированный набор объектов кодирования, а затем

b) каждый объект кодирования из «SecondaryEncodings» добавляется в комбинированный набор объектов кодирования, если, и только если, в этом комбинированном наборе объектов кодирования уже нет объекта кодирования, который имеет тот же класс кодирования (см. 17.1.7 и 9.23.2).

13.2.4 После этого схематического конструирования начинается кодирование с именем «encodingclassreference» структур кодирования, указанных в приложении кодирования (см. 13.1.2 и 17.5).

13.2.5 Когда в ELM имеется несколько приложений кодирования, правила из 12.2 гарантируют, что применения не будут перекрываться. Они обрабатываются независимо. Аналогично приложения кодирований к структурам кодирования в модулях EDM (определяемые в 13.2.10) всегда не перекрываются. В последующих подразделах приведены правила применения к одиночной структуре кодирования.

13.2.6 Объекты кодирования из комбинированного набора объектов кодирования применяются в точке приложения. Точкой приложения является первоначально «encodingclassreference» для генерируемой структуры кодирования (когда применение происходит в ELM, как определено в 13.1.2) или компонент структуры кодирования (когда применение происходит в EDM, как определено в 17.5).

13.2.7 Любой класс кодирования в категориях «альтернативы», «конкатенация» и «повторение» (см. 16.1.8, 16.1.9 и 16.1.10) является конструктором кодирования.

13.2.8 Термин «компонент» в последующем тексте относится к любым таким элементам.

a) альтернативы конструктора, относящегося к категории «альтернативы»;

b) поле, следующее за конструктором, относящимся к категории «повторение»;

c) компоненты конструктора, относящегося к категории «конкатенация»;

d) вложенный тип (тип, указанный в ограничении содержимого);

e) тип, выбранный (в экземпляре связи) для использования с классом в категории «открытый тип».

13.2.9 Точка приложения на последующих стадиях этих процедур может быть любым из следующих элементов:

a) имя класса кодирования. Это полностью подходит для кодирования с использованием спецификации в объекте кодирования того же класса (см. 17.1.7);

b) конструктор кодирования (см. 16.2.12). Процедуры конструирования могут быть определены спецификацией, содержащейся в объекте кодирования с классом конструктора кодирования, но этот объект кодирования не определяет кодирования этих компонентов. Спецификация применяемого объекта кодирования может потребовать, чтобы один или несколько компонентов конструктора были заменены другими структурами (параметризованными), перед тем как точка приложения перейдет к компонентам;

c) класс в категории «цепочка битов» или «цепочка октетов», который имеет некоторый вложенный тип в качестве признака, связанного со значениями [см. перечисление d) 11.3.4.3]. Кодирование вложенного типа зависит от того, присутствует ли **ENCODED BY**, и от спецификации применяемого объекта кодирования (см. 22.11);

d) класс в категории «открытый тип». Кодирование компонента открытого типа зависит от того, присутствует ли **ENCODED WITH**, и от спецификации применяемого объекта кодирования (см. 23.10.2);

e) компонент, который является классом кодирования (перед которым, возможно, имеются один или несколько классов в категории «тег»), за которым следует класс кодирования в категорию «факультативные возможности». Процедуры и кодирования для указания наличия или отсутствия определяются спецификацией, содержащейся в объекте кодирования классов в категории «факультативные возможности». Этот объект кодирования может также потребовать замены класса кодирования (вместе со всеми предшествующими ему классами в категории «тег») на структуру замены (параметризованную), перед тем как этот класс кодировать. Затем точка приложения переходит к первому классу в категории «тег» (если она имеется) или к компоненту, или к его заменителю;

f) класс кодирования, перед которым имеется класс кодирования в категории «тег». Номер тега, связанный с классом в категории «тег», кодируется с помощью спецификации в объекте кодирования с классом в категории «тег», а точка приложения переходит затем к тегированному классу;

g) любой другой предопределенный класс кодирования. Это полностью подходит для кодирования с использованием спецификации, содержащейся в объекте кодирования этого класса.

13.2.10 Кодирование продолжается следующим образом.

13.2.10.1 Если комбинированный набор объектов кодирования содержит объект кодирования того же класса (см. 17.1.7), что и текущая точка приложения, то применяется этот объект кодирования. Это применение может вызвать замену одного или нескольких компонентов с классом, к которому это кодирование применяется. Если же комбинированный набор объектов кодирования не содержит такого объекта кодирования, то либо:

a) класс кодирования в текущей точке приложения является ссылкой на другой класс кодирования; в этом случае ссылка на него убирается, а процедуры из 13.2.10 применяются рекурсивно, либо

b) класс кодирования в текущей точке приложения не является ссылкой на другой класс кодирования; в этом случае спецификация ECN имеет ошибку.

13.2.10.2 Если кодирование применено в точке приложения к классу кодирования, а он не относится к категории «факультативные возможности» или «тег» и не имеет компонентов (см. 13.2.7), то такое применение полностью определяет кодирование этого класса и заканчивает процедуру.

13.2.10.3 Если кодирование применено в точке приложения к классу кодирования, который относится к категории «факультативные возможности», то точка приложения переходит к факультативному компоненту (возможно, тегированному).

13.2.10.4 Если кодирование применено в точке приложения к классу кодирования, который относится к категории «тег», то точка приложения переходит к тегированному элементу, а процедуры из 13.2.10 применяются рекурсивно.

13.2.10.5 Если кодирование применимо в точке приложения к классу кодирования, который имеет компонент, относящийся к какому-либо вложенному типу, то процедуры из 13.2.10 применяются рекурсивно к каждому компоненту.

Примечание — Если объект кодирования будучи применен к классу в категории «открытый тип» содержит ENCODED WITH, то это определяет набор объектов кодирования, применяемый к компоненту, в противном случае комбинированный набор объектов кодирования, применяемый к этому классу, применяется и к компоненту (см. 23.10.2).

13.2.10.6 Если кодирование применено в точке приложения к классу кодирования, который имеет компонент, относящийся к классу в категории «цепочка битов» или «цепочка октетов» с вложенным типом, связанным со значениями, то могут встретиться четыре случая:

a) ограничение содержимого имеет **ENCODED BY**, а объект кодирования для этого класса либо не содержит спецификации кодирования вложенного типа, либо указывает, что он не отменяет **ENCODED BY** (см. 22.11). В этом случае для вложенного типа используется спецификация **ENCODED BY**, а точка приложения переходит к вложенному типу с использованием этой спецификации кодирования;

b) ограничение содержимого имеет **ENCODED BY**, но объект кодирования для этого класса содержит спецификацию кодирования вложенного типа и указывает, что он отменяет **ENCODED BY**. В этом случае для вложенного типа применяется спецификация в объекте кодирования, а точка приложения переходит к вложенному типу с использованием этой спецификации кодирования;

c) ограничение содержимого не имеет **ENCODED BY**, а объект кодирования для этого класса содержит спецификацию кодирования вложенного типа. В этом случае к вложенному типу применяется спецификация в объекте кодирования, а точка приложения переходит к вложенному типу с использованием этой спецификации кодирования;

d) ограничение содержимого не содержит **ENCODED BY**, а объект кодирования для этого класса не содержит спецификации кодирования вложенного типа. В этом случае комбинированный набор объектов кодирования, прилагаемый к этому классу, применяется также к типу содержимого, а точка приложения переходит к вложенному типу с использованием этой спецификации кодирования.

13.2.10.7 Если в комбинированном наборе объектов кодирования нет объекта кодирования того же класса (см. 17.1.7), что и текущая точка приложения, а текущая точка приложения является справочным именем, то она разыменуется, а эти процедуры применяются рекурсивно к новой структуре кодирования.

13.2.10.8 В других случаях спецификация ECN будет ошибочной.

13.2.11 Вышеописанный алгоритм может быть суммирован следующим образом: комбинированный набор объектов кодирования применяется по принципу «сверху вниз». Если в этом процессе встречается справочное имя структуры кодирования, а в комбинированном наборе объектов кодирования имеется объект, который может кодировать его, то этот объект определяет его кодирование. В других случаях справочное имя расширяется путем разыменования. Если на какой-либо стадии потребуется (но не присутствует) кодирование для класса кодирования, который не может быть разыменован, то спецификация ECN неверна, а комбинированный класс кодирования считается неполным. Когда достигнут простейший класс битового поля, кодирование заканчивается на кодировании этого класса; однако если он имеет какой-либо вложенный тип, кодирование продолжается до генерируемой структуры кодирования, соответствующей этому вложенному типу. Когда достигнут тип с компонентами, процесс продолжается путем применения комбинированного набора объектов кодирования к каждому компоненту независимо. Когда участвуют теги и факультативные возможности, класс функциональных возможностей кодируется первым, затем кодируется класс в категории «тег» и, наконец, элемент. Когда

кодирования применяются к классам конструктора, они могут вызывать замену одного или нескольких компонентов. Когда они применяются к классу факультативных возможностей, они могут вызывать замену всего элемента (отдельно от класса факультативных возможностей, но включая любой класс кодирования в категории «тег»).

13.2.12 В процессе кодирования объекты кодирования, приложенные к конструкторам кодирования (и к классам в категории «факультативные возможности»), могут потребовать, чтобы объекты кодирования, приложенные к компонентам конструкций, определяемым этими конструкторами, показывали идентификационные описатели (заданного имени), чтобы различать альтернативы, или факультативные возможности, или прекращение повторения, или порядок в конкатенации, похожей на набор. Они могут также потребовать, чтобы объекты кодирования, применяемые для других классов кодирования (следующие за этими конструкциями), показывали такой же идентификационный описатель и чтобы наборы значений описателя всех участвующих объектов кодирования (показывающих такой же описатель) были непересекающимися. Если эти условия не соблюдаются, спецификация ECN будет ошибочной.

Примечание — Эта проблема наиболее вероятно возникает в объектах кодирования BER, примененных к конструкторам кодирования, а не к их компонентам, так как BER сильно зависит от идентификационных описателей. Объекты кодирования PER не используют идентификационные описатели.

14 Модуль определения кодирования (EDM)

Примечание — В ECN имеются две продукции верхнего уровня: «EDMDefinition», определяемая в этом разделе, и «ELMDefinition», определяемая в разделе 12. Они определяют синтаксис для описания модулей EDM и модуля ELM соответственно.

14.1 Продукцией «EDMDefinition» является:

```
EDMDefinition ::=
    ModuleIdentifier
    ENCODING-DEFINITIONS
    "::="
    BEGIN
    EDMModuleBody
    END
```

14.2 В любом заданном приложении ECN имеются нуль, один или несколько EDM, которые определяют объекты кодирования для приложения в ELM.

Примечание — Если не имеется EDM, то в ELM могут использоваться только предопределенные объекты кодирования.

14.3 Продукция «ModuleIdentifier» (и ее семантика) определена в ИСО/МЭК 8824-1, пункт 13.1.

14.4 «ModuleIdentifier» обеспечивает однозначную идентификацию любого модуля в совокупности всех модулей ACH.1, ELM и EDM.

14.5 Продукцией «EDMModuleBody» является:

```
EDMModuleBody ::=
    Exports ?
    RenamesAndExports ?
    Imports ?
    EDMAssignmentList ?
EDMAssignmentList ::=
    EDMAssignment
    EDMAssignmentList ?

EDMAssignment ::=
        EncodingClassAssignment
      | EncodingObjectAssignment
      | EncodingObjectSetAssignment
      | ParameterizedAssignment
```

14.6 Продукции «Exports» и «Imports» (и их семантика) определены в ИСО/МЭК 8824-1, пункт 13.1 с учетом изменений из А.1.

14.7 «Exports» делает доступным импорт в другие EDM (и ELM) любого справочного имени, определенного в текущем EDM или импортированного в него, кроме имени из неявно генерируемой структуры. «Symbol» в «Exports» может указывать на любой класс кодирования (кроме предопределенного класса кодирования или неявно генерируемой структуры), объект кодирования и набор объектов кодирования. Этот «Symbol» должен быть определен в этом EDM или импортирован в него.

Примечание — Когда имя импортированной неявно генерируемой структуры кодирования является ссылкой на предопределенный класс кодирования, оно может использоваться внутри EDM с полностью определенным именем. Неявно генерируемая структура кодирования не может быть экспортирована из EDM (однако структуры кодирования, определенные с ее помощью, могут быть, конечно, экспортированы).

14.8 Продукция «RenamesAndExports» определяется в разделе 15.

14.9 Элемент «RenamesAndExports» (называемый разделом переименований) делает доступными (внутри EDM) явно генерируемые структуры кодирования, полученные из неявно генерируемых структур кодирования в указанных модулях АСН.1. Он делает также эти явно генерируемые структуры кодирования доступными для импорта в другие EDM (и в ELM) (см. раздел 15).

14.10 «Imports» делает доступными (внутри EDM) классы кодирования, объекты кодирования и наборы объектов кодирования, экспортированные из других EDM или автоматически экспортированные из модулей АСН.1.

14.11 Все модули АСН.1, которые определяют справочные имена непараметризованных типов, автоматически вырабатывают и экспортируют неявно генерируемую структуру кодирования с тем же именем, перед которым ставится знак «#». Такие классы кодирования могут быть импортированы в EDM из указанного модуля АСН.1.

Примечание — Если такие имена совпадают с именами предопределенного класса кодирования, то внешняя форма, определенная в А.1, должна использоваться в теле импортируемого модуля и в любом разделе переименований.

14.12 Каждая продукция «EDMAssignment» определяет справочное имя и может использовать другие справочные имена. Каждое справочное имя, использованное в модуле, будет импортироваться в этот модуль либо будет один раз точно определено внутри этого модуля.

Примечание — Это является более строгим требованием, чем предъявляемое к модулям АСН.1. В модулях АСН.1 внешние ссылки могут использоваться для типов и значений, которые не были импортированы. В модуле EDM (и в модуле ELM) внешние ссылки могут использоваться только для классов кодирования, которые были указаны в разделе импортов. Цель внешних ссылок — это только устранение совпадений между импортируемыми именами и предопределенными именами либо между двумя идентичными именами, импортированными из разных модулей.

14.13 Не требуется, чтобы любое справочное имя, использованное в одном присвоении, было определено текстуально (в другом операторе присвоения) перед его использованием.

14.14 Продукции из «EDMAssignment» определяются в последующих разделах:

EncodingClassAssignment раздел 16;
EncodingObjectAssignment раздел 17;
EncodingObjectSetAssignment раздел 18;
ParameterizedAssignment подраздел С.1.

Примечание — «ParameterizedAssignment» позволяет параметризацию «EncodingClassAssignment», «EncodingObjectAssignment» и «EncodingObjectSetAssignment», как показано в С.1.

15 Раздел переименований

15.1 Явно генерируемые и экспортируемые структуры

15.1.1 Продукцией «RenamesAndExports» является:

```
RenamesAndExports ::=
        RENAMES
        ExplicitGenerationList ";"
```

```
ExplicitGenerationList ::=
        ExplicitGeneration
        ExplicitGenerationList ?
ExplicitGeneration ::=
        OptionalNameChanges
        FROM GlobalModuleReference
OptionalNameChanges ::=
        NameChanges | GENERATES
```

Примечание — Пример использования раздела переименований для создания явно генерируемых структур кодирования приведен в D.3.7.

15.1.2 Продукция «GlobalModuleReference» определена в ИСО/МЭК 8824-1, пункт 13.1; она должна указывать модуль ACH.1.

15.1.3 «RenamesAndExports» называется разделом переименований.

15.1.4 Каждый элемент «ExplicitGeneration» генерирует и экспортирует из этого модуля явно генерируемую структуру кодирования для каждой из неявно генерируемых структур кодирования модуля ACH.1, указанных в «GlobalModuleReference». Каждое поле явно генерируемой структуры кодирования имеет связанные с ним такие же значения, что и соответствующее поле неявно генерируемой структуры кодирования (той, которая связана с соответствующим полем типа ACH.1, из которого она была генерирована).

15.1.5 Если раздел переименований ссылается более чем на один модуль ACH.1 и в результате этих двух явно генерируемых структур получает одно и то же простое имя, то никакая структура не будет доступна для явного импорта в модуль ELM или EDM.

Примечание — Эти явно генерируемые структуры, тем не менее, существуют и, вероятно, будут неявно указываться другими явно генерируемыми структурами, которые экспортированы без ограничения.

15.1.6 Основная цель раздела переименований — это обеспечение доступности явно генерируемых структур для импорта в другие модули, в частности в ELM. Однако этот раздел делает эти структуры также доступными для ссылки внутри модуля EDM, за исключением случая, указанного в 15.1.7. Если простое имя допускает неоднозначность, то в этом модуле EDM, содержащем раздел переименований, как описывается в 15.1.9, должно использоваться полностью определенное имя.

Примечание — Неоднозначность может возникнуть либо из-за совпадения с именами предопределенных классов, либо из-за совпадения простых имен из структур, генерируемых из более чем одного модуля ACH.1, либо по обеим причинам.

15.1.7 Когда раздел переименований вырабатывает явно генерируемую структуру из неявно генерируемой структуры, эта неявно генерируемая структура не может импортироваться в этот модуль EDM с помощью раздела импортов, и эта неявно генерируемая структура будет недоступной в этом модуле EDM.

15.1.8 Эти явно генерируемые структуры кодирования имеют то же простое справочное имя, что и неявно генерируемая структура кодирования, из которой они были сформированы (но будут иметь другие классы). Если для явно генерируемой структуры кодирования требуется полностью определенное имя, то оно должно содержать «ModuleIdentifier» модуля EDM, имеющего раздел переименований, как описывается в 15.1.9.

Примечание — Неявно генерируемые структуры кодирования, используемые в их генерировании, имеют то же самое простое справочное имя, но их полностью определенное имя содержит «ModuleIdentifier» модуля ACH.1, в котором был определен соответствующий тип.

15.1.9 Если EDM вырабатывает явно генерируемые структуры кодирования из более чем одного модуля ACH.1, то возможно, что некоторые из этих структур будут иметь одинаковые простые имена классов кодирования. Если некоторые из этих структур указаны в теле этого EDM, то такой ссылкой должна быть «ExternalEncodingClassReference», содержащая «modulereference», использованную в качестве ссылки на модуль ACH.1 в компоненте «замены» модуля EDM.

15.1.10 Нотация «ExternalEncodingClassReference» не используется в разделе импортов, за исключением требования из 15.1.9.

15.1.11 Если имя, импортированное с помощью «ExternalEncodingClassRefererence», использовано в теле модуля, то может применяться простая «encodingclassreference», когда не требуется «ExternalEncodingClassReference» согласно 15.1.9.

15.1.12 Если «OptionalNameChanges» является **GENERATES**, то все явно генерируемые структуры кодирования будут теми же структурами, что и неявно генерируемые структуры кодирования, используемые при их генерации, за исключением случаев, определенных в 15.1.14.

Примечание — (Руководство) Если в модуле EDM имеются несколько структур с одинаковыми простыми справочными именами (эти имена возникли или из компонента импортов, или из компонента переименований, или из-за совпадений с предопределенными классами, или из-за любой комбинации этих причин), то используется полностью определенное имя, за исключением ссылок на предопределенный класс. Для неявно генерируемых структур полностью определенное имя всегда использует имя модуля ACH.1. Для структур, генерируемых компонентом переименований в модуле EDM, используется полностью определенное имя. Это полностью определенное имя в теле этого EDM всегда использует имя модуля ACH.1, указанное в компоненте переименований. Для структур, импортированных из другого модуля EDM, полностью определенное имя использует имя этого модуля EDM. Это всегда обеспечивает недвусмысленность, так как импортирование не разрешается, если модуль EDM генерирует несколько явно генерируемых структур с одним и тем же простым справочным именем.

15.1.13 Если «OptionalNameChanges» является «NameChanges», то все еще применяется 15.1.14, но явно генерируемые структуры кодирования затем изменяются, как описывается в 15.2.

15.1.14 Рассмотрим неявно генерируемую структуру кодирования (назовем ее A), которая содержит ссылку на класс кодирования некоторой другой неявно генерируемой структуры кодирования (назовем ее B). Тогда:

a) если этот компонент переименований (в любой из его «ExplicitGeneration») вырабатывает явно генерируемую структуру кодирования, соответствующую B (назовем ее B1), то соответствующая ссылка в явно генерируемой структуре кодирования, соответствующей A, будет ссылкой на B1;

b) если нет явно генерируемой структуры кодирования, соответствующей B, то ссылка в генерируемой структуре кодирования, соответствующей A, будет ссылкой на B.

15.2 Изменения имен

15.2.1 Продукцией «NameChanges» является:

```
NameChanges ::=
        NameChange
        NameChanges ?
NameChange ::=
      OriginalClassName
      AS
    NewClassName
    IN
    NameChangeDomain
OriginalClassName ::= SimpleDefinedEncodingClass| BuiltinEncodingClassReference
NewClassName ::= encodingclassreference
```

15.2.2 Каждая «NameChanges» описывает, что при генерировании явно генерируемой структуры кодирования все появившиеся «OriginalClassName» внутри «NameChangeDomain» в неявно генерируемых структурах кодирования должны переименовываться в класс «NewClassName». «NameChangeDomain» определяется в 15.3; он определяет одну или более неявно генерируемых структур кодирования (или компоненты таких структур) из модуля ACH.1, указанного ссылкой «GlobalModuleReference» в «ExplicitGeneration».

Примечания

1 Это дает возможность применения к некоторым случаям появления класса других кодирований, отличающихся от примененных к другим случаям появления этого класса.

2 Это означает, что «OriginalClassName» может быть только именем, неявно генерируемым из типа ACH.1, то есть именем определенного пользователем типа ACH.1 (перед которым ставится **«#»**) или одним из имен классов, перечисленных в столбце 2 таблицы 2.

15.2.3 Ссылки в «OriginalClassName» на поля неявно генерируемой структуры кодирования, которая соответствует использованию «ExternalTypeReference» в определении типа ACH.1, должны использовать нотацию «SimpleDefinedEncodingClass» с тем же «modulereference», что и «ExternalTypeReference». В другом случае, если «DefinedType» (перед которым имеется **«#»**) не является «BuiltinEncodingClass Reference», то должна использоваться простая «encodingclassreference». Если «typereference» (перед

которой имеется **«#»**) является «BuiltinEncodingClassReference», то должна использоваться нотация «SimpleDefinedEncodingClass» с тем же «modulereference», что у модуля ACH.1, который генерирует неявно генерируемую структуру кодирования.

15.2.4 Когда ELM импортирует явно генерируемую структуру кодирования из EDM, компоненты переименований в других EDM не влияют на кодирование этой структуры.

Примечание — Это означает, что все «окрашивание» (см. 9.16.4), необходимое для любого конкретного сообщения, должно выполняться в одном EDM.

15.2.5 «NewClassName» должен определяться в операторе присвоения класса кодирования (см. раздел 16) в следующей форме:

<NewClassName> ::= <OriginalClassName>,

где «<*NewClassName*>» и «<*OriginalClassName*>» являются именами нового и исходного классов, появившихся в продукции «NameChanges». Это присвоение должно быть в модуле EDM с компонентом «переименования».

Примечание — «<*OriginalClassName*>» требуется для того, чтобы сослаться на предопределенный класс кодирования или на внешнюю генерируемую структуру кодирования, созданную компонентом «переименования» в этом модуле. В случае неоднозначности необходимо будет использовать внешнюю ссылку в «<*OriginalClassName*>».

15.3 Определение области для изменений имени

15.3.1 Продукцией «NameChangeDomain» является:

```
NameChangeDomain ::=
        IncludedRegions
        Exception ?
    Exception ::=
        EXCEPT
        ExcludedRegions
    IncludedRegions ::=
        ALL | RegionList
    ExcludedRegions ::= RegionList
    RegionList ::=
    Region "," +
Region ::=
    SimpleDefinedEncodingClass |
    ComponentReference
ComponentReference ::=
    SimpleDefinedEncodingClass
    "."
    ComponentIdList
ComponentIdList ::=
    identifier "." +
```

15.3.2 Каждый «SimpleDefinedEncodingClass» должен быть именем неявно генерируемой структуры кодирования из модуля ACH.1, указанного ссылкой «GlobalModuleReference» в «ExplicitGeneration». При использовании в «Region» он определяет полное описание этой структуры кодирования.

Примечание — Форма «ExternalEncodingClassReference» класса «SimpleDefinedEncodingClass» используется, когда указанный класс выделен из имени «typereference», который (когда имеет предшествующий «#») является ссылкой «BuiltinEncodingClassReference» (см. 15.2.3).

15.3.3 Каждый «identifier» должен быть «identifier» из «NamedField» неявно генерируемой структуры кодирования, указанной ссылкой «encodingclassreference» в «ComponentReference». Эта «ComponentReference» определяет полное описание указанного компонента этой структуры кодирования.

15.3.4 Первым «identifier» в «ComponentIdList» должен быть «identifier» из «NamedField» неявно генерируемой структуры кодирования, указанной ссылкой «encodingclassreference» в «ComponentReference»; он определяет полное описание этого компонента структуры кодирования. Каждым последующим «identifier» в «ComponentIdList» будет «identifier» из «NamedField» неявно генерируемой структуры кодирования, указанной предыдущей частью «ComponentIdList»; он определяет полное описание этого компонента.

15.3.5 Определения, указанные разными «Region» в «RegionList», должны быть несвязанными. Определение указывается в «RegionList», если, и только если, оно указано в «Region» этого «RegionList».

15.3.6 Если «IncludedRegions» имеет значение ALL, то он указывает на все части всех неявно генерируемых структур кодирования из модуля ACH.1, указанного ссылкой «GlobalModuleReference» в «ExplicitGeneration».

15.3.7 Определения, указанные в «ExcludedRegions», должны быть правильным подмножеством определений, указанных в «IncludedRegions».

15.3.8 Спецификация «NameChangeDomain» указывает определения, в которых следует выполнить изменения имени. Определениями в «NameChangeDomain» являются определения, указанные в «IncludedRegions», но не указанные также в «ExcludedRegions».

16 Присвоения классов кодирования

16.1 Общие положения

16.1.1 Продукцией «EncodingClassAssignment» является:

```
EncodingClassAssignment ::=
        encodingclassreference
        "::="
        EncodingClass
```

16.1.2 «EncodingClassAssignment» назначает «EncodingClass» для «encodingclassreference».

Примечание — Любая нотация «EncodingObject», которая действительна с «EncodingClass» в качестве руководителя, будет действительной с «encodingclassreference» в качестве руководителя.

16.1.3 Любой класс кодирования будет относиться к одной из следующих категорий:

a) какая-либо категория из группы категорий «битовое поле» (см. 16.1.7);
b) категория «альтернативы» (см. 16.1.8);
c) категория «конкатенация» (см. 16.1.9);
d) категория «повторение» (см. 16.1.10);
e) категория «факультативные возможности» (см. 16.1.11);
f) категория «тег» (см. 16.1.12);
g) какая-либо категория из группы категорий «процедура кодирования» (см. 16.1.13).

Примечание — Термин «конструктор кодирования» используется для любого класса в категориях «альтернативы», «конкатенация» и «повторение». Они называются также группой категорий «конструктор кодирования».

16.1.4 Категория для каждого предопределенного класса кодирования определена в 16.1.14.

Примечание — Если классом кодирования является тегированный класс (см. 16.2.1) или класс имеет границы (см. 16.2.6), то категорией этого класса является категория класса с удаленными тегом и границами.

16.1.5 Продукцией «EncodingClass» является:

```
EncodingClass ::=
        BuiltinEncodingClassReference
      | EncodingStructure
```

16.1.6 Продукцией «BuiltinEncodingClassReference» является:

```
BuiltinEncodingClassReference ::=
        BitfieldClassReference
      | AlternativesClassReference
```

```
                | ConcatenationClassReference
                | RepetitionClassReference
                | OptionalityClassReference
                | TagClassReference
                | EncodingProcedureClassReference
```

16.1.7 Продукцией «BitfieldClassReference» является:

```
BitfieldClassReference ::=
                    #NUL
                  | #BOOL
                  | #INT
                  | #BITS
                  | #OCTETS
                  | #CHARS
                  | #PAD
                  | #BIT-STRING
                  | #BOOLEAN
                  | #CHARACTER-STRING
                  | #EMBEDDED-PDV
                  | #ENUMERATED
                  | #EXTERNAL
                  | #INTEGER
                  | #NULL
                  | #OBJECT-IDENTIFIER
                  | #OCTET-STRING
                  | #OPEN-TYPE
                  | #REAL
                  | #RELATIVE-OID
                  | #TIME
                  | #DATE
                  | #DATE-TIME
                  | #TIME-OF-DAY
                  | #DURATION
                  | #GeneralizedTime
                  | #UTCTime
                  | #ObjectDescriptor
                  | #BMPString
                  | #GeneralString
                  | #GraphicString
                  | #IA5String
                  | #NumericString
                  | #PrintableString
                  | #TeletexString
                | #UniversalString
                | #UTF8String
                | #VideotexString
                | #VisibleString
```

Для всех категорий классов, которые указывают на эти предопределенные имена (см. 16.1.14), определено, что они относятся к группе категорий «битовое поле».

16.1.8 Продукцией «AlternativesClassReference» является:

```
AlternativesClassReference ::=
                    #ALTERNATIVES
                  | #CHOICE
```

16.1.9 Продукцией «ConcatenationClassReference» является:

```
ConcatenationClassReference ::=
                    #CONCATENATION
                  | #SEQUENCE
                  | #SET
```

16.1.10 Продукцией «RepetitionClassReference» является:

```
RepetitionClassReference ::=
                    #REPETITION
                  | #SEQUENCE-OF
                  | #SET-OF
```

16.1.11 Продукцией «OptionalityClassReference» является:

```
OptionalityClassReference ::=
                    #OPTIONAL
```

16.1.12 Продукцией «TagClassReference» является:

```
TagClassReference ::=
                    #TAG
```

16.1.13 Продукцией «EncodingProcedureClassReference» является:

```
EncodingProcedureClassReference ::=
                    #TRANSFORM
                  | #CONDITIONAL-INT
                  | #CONDITIONAL-REPETITION
                  | #OUTER
```

16.1.14 Некоторые из этих классов являются простейшими и могут кодироваться только с помощью объектов кодирования своего собственного класса. Другие образуются из какого-либо простейшего класса с помощью операторов присвоения класса и могут быть разыменованы к этим классам. Их категорией будет категория класса, из которого они образованы. Ниже показаны простейшие классы, каждый из которых образован из предопределенного класса при помощи операторов присвоения класса. При определении объектов кодирования образованных классов любой синтаксис, разрешенный для соответствующего простейшего класса, может использоваться для образованного класса. Третий столбец дает категорию каждого из предопределенных классов, которые не образованы из других классов.

Предопределенный класс	Образован из	Категория
#ALTERNATIVES	(простейший)	alternatives
#BITS	(простейший)	bitstring
#BIT-STRING	#BITS	
#BOOL	(простейший)	boolean
#BOOLEAN	#BOOL	
#CHARACTER-STRING	(определен с помощью #SEQUENCE)	
#CHARS	(простейший)	characterstring
#CHOICE	#ALTERNATIVES	
#CONCATENATION	(простейший)	concatenation
#CONDITIONAL-INT	(простейший)	encoding procedure
#CONDITIONAL-REPETITION	(простейший)	encoding procedure
#EMBEDDED-PDV	(определен с помощью #SEQUENCE)	
#ENUMERATED	#INT	

#EXTERNAL	(определен с помощью #SEQUENCE)	
#INT	(простейший)	integer
#INTEGER	#INT	
#NUL	(простейший)	null
#NULL	#NUL	
#OBJECT-IDENTIFIER	(простейший)	objectidentifier
#OCTETS	(простейший)	octetstring
#OCTET-STRING	#OCTETS	
#OPEN-TYPE	(простейший)	opentype
#OPTIONAL	(простейший)	optionality
#OUTER	(простейший)	encoding procedure
#PAD	(простейший)	pad
#REAL	(простейший)	real
#RELATIVE-OID	#OBJECT-IDENTIFIER	
#REPETITION	(простейший)	repetition
#SEQUENCE	#CONCATENATION	
#SEQUENCE-OF	#REPETITION	
#SET		#CONCATENATION
#SET-OF	#REPETITION	
#TAG	(простейший)	tag
#TIME	(простейший)	time
#DATE	#TIME	
#TIME-OF-DAY	#TIME	
#DATE-TIME	#TIME	
#DURATION	#TIME	
#TRANSFORM	(простейший)	encoding procedure
#GeneralizedTime	#CHARS	
#UTCTime	#CHARS	
#ObjectDescriptor	#CHARS	
#BMPString	#CHARS	
#GeneralString	#CHARS	
#GraphicString	#CHARS	
#IA5String	#CHARS	
#NumericString	#CHARS	
#PrintableString	#CHARS	
#TeletexString	#CHARS	
#UniversalString	#CHARS	
#UTF8String	#CHARS	
#VideotexString	#CHARS	
#VisibleString	#CHARS	

16.2 Определение структуры кодирования

16.2.1 Продукцией «EncodingStructure» является:

```
EncodingStructure ::=
        TaggedStructure
      | UntaggedStructure
TaggedStructure ::=
        "["
        TagClass
        TagValue ?
        "]"
        UntaggedStructure
UntaggedStructure ::=
        DefinedEncodingClass
```

```
            | EncodingStructureField
            | EncodingStructureDefn
TagClass ::=
              DefinedEncodingClass |
              TagClassReference
 TagValue ::=
               "("number")"
```

16.2.2 «EncodingStructure» определяет класс кодирования на базе структуры с использованием нотации, описанной ниже. Эта нотация позволяет определять произвольные классы кодирования с помощью предопределенных классов кодирования и определенных классов кодирования (которые могут быть генерируемыми структурами кодирования) для битовых полей, конструкторов кодирования и классов «процедура кодирования» в категории «факультативные возможности». Все классы, которые определяет «EncodingStructure», относятся к категории «структура кодирования» (примеры назначения структуры кодирования, иллюстрирующие многие синтаксические структуры, приведены в D.2.8.4, а в D.2.2.3 приведен пример использования **#TAG**).

Примечание — Синтаксис препятствует спецификации тегированного класса немедленно после другого тегированного класса в определении структуры кодирования, а также такие структуры не могут вырабатываться с несколькими текстуальными тегами в определении типа ACH.1 [см. перечисление e) 11.3.4.1].

16.2.3 «DefinedEncodingClass» определен в 10.9.1; он должен быть классом в группе категорий «битовое поле».

16.2.4 «DefinedEncodingClass» в «TagClass» должен быть классом в категории «тег» (см. 16.1.3).

16.2.5 Элемент «number» в «TagValue» указывает номер тега, который связан с классом в категории «тег».

16.2.6 Продукцией «EncodingStructureField» является:

```
EncodingStructureField ::=
          #NUL
          | #BOOL
          | #INT                    Bounds?
          | #BITS                   Size?
          | #OCTETS                 Size?
          | #CHARS                  Size?
 | #PAD
 | #BIT-STRING                     Size?
 | #BOOLEAN
 | #CHARACTER-STRING
 | #EMBEDDED-PDV
 | #ENUMERATED                     Bounds?
 | #EXTERNAL
 | #INTEGER                        Bounds?
 | #NULL
 | #OBJECT-IDENTIFIER
 | #OCTET-STRING                   Size?
 | #OPEN-TYPE
 | #REAL
 | #RELATIVE-OID
 | #TIME
 | #DATE
 | #TIME-OF-DAY
 | #DATE-TIME
 | #DURATION
 | #GeneralizedTime
 | #UTCTime
 | #ObjectDescriptor               Size?
```

```
| #BMPString                Size?
| #GeneralString            Size?
| #GraphicString            Size?
| #IA5String                Size?
| #NumericString            Size?
| #PrintableString          Size?
| #TeletexString            Size?
| #UniversalString          Size?
| #UTF8String               Size?
| #VideotexString           Size?
| #VisibleString            Size?
```

16.2.7 «EncodingStructureField» представляет все возможные кодирования цепочки битов для соответствующих типов АСН.1 и может быть назначенными значениями этих типов при отображении значения (см. раздел 19).

16.2.8 Значениями АСН.1, которые могут быть связаны с каждым простейшим полем, являются следующие:

```
#NUL                      Значение НУЛЬ
#BOOL                     Булевы значения
#INT                      Целочисленные значения
#BITS                     Значения цепочки битов
#OCTETS                   Значения цепочки октетов
#CHARS                    Значения цепочки знаков
#PAD                      НЕТ
#OBJECT-IDENTIFIER        Значения идентификатора объекта
#OPEN-TYPE                Значения открытого типа
#REAL                     Значения действительного числа
#TIME                     Значения времени
#TAG                      Номера тегов
```

Примечание — Поле #PAD не может иметь связанных значений АСН.1, и его никогда не видно за пределами процедур кодирования и декодирования.

16.2.9 «Bounds» и «Size» определяют границы или ограничение на действующий размер соответственно в абстрактных значениях, которые могут отображаться в поле (см. раздел 19).

Примечание — Ограничения на действующий разрешенный алфавит не могут назначаться в определении структуры кодирования. Они могут назначаться только путем отображений значения согласно разделу 19.

16.2.10 Продукциями «Bounds» и «Size» являются:

```
Bounds ::= "(" EffectiveRange ")"
EffectiveRange ::=
            MinMax
          | Fixed
Size ::= "(" SIZE SizeEffectiveRange ")"
SizeEffectiveRange ::=
            "(" EffectiveRange ")"
 MinMax ::=
            ValueOrMin
            ".."
            ValueOrMax
 ValueOrMin ::=
            SignedNumber |
            MIN
 ValueOrMax ::=
```

```
                    SignedNumber |
                    MAX
            Fixed ::= SignedNumber
```

16.2.11 **MIN** и **MAX** указывают, что отсутствует нижняя или верхняя граница соответственно. **MIN** не используется в «Size». «Fixed» означает одиночное значение или одиночный размер. «SignedNumber» определен в ИСО/МЭК 8824-1, подраздел 19.1. Он должен быть неотрицательным при использовании в «Size». «ValueOrMin» и «ValueOrMax» указывают нижнюю и верхнюю границы соответственно.

16.2.12 Продукцией «EncodingStructureDefn» является:

```
EncodingStructureDefn ::=
                    AlternativesStructure
                  | RepetitionStructure
                  | ConcatenationStructure
```

16.2.13 Эти структуры кодирования определяются в следующих разделах:

AlternativesStructure 16.3;
RepetitionStructure 16.4;
ConcatenationStructure 16.5.

16.3 Структура кодирования альтернативы

16.3.1 Продукцией «AlternativesStructure» является:

```
AlternativesStructure ::=
        AlternativesClass
        "{"
        NamedFields
    "}"
AlternativesClass ::=
    DefinedEncodingClass |
    AlternativesClassReference
NamedFields ::= NamedField "," +
NamedField ::=
    identifier
    EncodingStructure
```

16.3.2 «AlternativesStructure» указывает на наличие при кодировании только одной структуры из возможных «EncodingStructure» в ее «NamedFields». В «DefinedEncodingClass» указывается класс в категории «альтернативы» (см. 16.1.8). Механизм, который определяет, какая именно «EncodingStructure» присутствует при кодировании, указывается объектом кодирования из «AlternativesClass».

16.3.3 «AlternativesStructure» является конструктором кодирования: когда набор объектов кодирования применен к этой структуре согласно 13.2, кодирование в «AlternativesClass» определяет выбор альтернатив, а точка приложения затем переходит к каждой «EncodingStructure» в ее «NamedFields».

16.4 Структура кодирования повторения

16.4.1 Продукцией «RepetitionStructure» является:

```
RepetitionStructure ::=
        RepetitionClass
        "{"
        identifier ?
        EncodingStructure
        "}"
        Size?
```

```
RepetitionClass ::=
        DefinedEncodingClass |
        RepetitionClassReference
```

16.4.2 «RepetitionStructure» указывает на наличие при кодировании повторяющихся случаев появления «EncodingStructure» в продукции. Факультативная конструкция «Size» (см. 16.2.9) указывает границы для числа повторений. Механизм, который определяет, сколько именно повторений «EncodingStructure» имеется в кодировании, указывается объектом кодирования класса «RepetitionClass». «DefinedEncodingClass» должен быть классом из категории «повторение» (см. 16.1.10).

16.4.3 «RepetitionStructure» является конструктором кодирования: когда объект кодирования применен к этой структуре согласно разделу 13.2, кодирование в «RepetitionClass» определяет механизмы для определения числа повторений, а точка приложения затем переходит к «EncodingStructure» в продукции.

Примечание — Знаки «{» и «}» используются в этой конструкции, но не присутствуют в соответствующей конструкции АСН.1 **SEQUENCE OF**. Это было сделано во избежание синтаксической неоднозначности в определении структуры.

16.5 Структура кодирования конкатенации

16.5.1 Продукцией «ConcatenationStructure» является:

```
ConcatenationStructure ::=
        ConcatenationClass
        "{"
        ConcatComponents
        "}"
ConcatenationClass ::=
        DefinedEncodingClass |
        ConcatenationClassReference
ConcatComponents ::=
        ConcatComponent "," *
ConcatComponent ::=
        NamedField
        ConcatComponentPresence ?
ConcatComponentPresence ::=
        OPTIONAL-ENCODING
        OptionalClass
OptionalClass ::=
        DefinedEncodingClass |
        OptionalityClassReference
```

16.5.2 «ConcatenationStructure» указывает на наличие при кодировании нуля или одного кодирования для каждой из «EncodingStructure» в ее «NamedField». «DefinedEncodingClass» в «ConcatenationClass» должен быть классом в категории «конкатенация» (см. 16.1.9), а «DefinedEncodingClass» в «OptionalClass» должен быть классом в категории «факультативные возможности» (см. 16.1.3).

16.5.3 Если «ConcatComponentPresence» отсутствует в «composant», то «EncodingStructure» в этом именованном поле должна появляться только один раз при кодировании.

16.5.4 Если «ConcatComponentPresence» присутствует, то механизм, который определяет, имеется ли кодирование соответствующей «EncodingStructure», указывается объектом кодирования, который кодирует «OptionalClass».

16.5.5 Порядок, в котором кодирования каждого «NamedField» появляются в кодировании конкатенации (и средства для указания, какой «NamedField» представляет кодирование), указывается объектом кодирования класса «ConcatenationClass».

16.5.6 «ConcatenationStructure» является конструктором кодирования: когда объект кодирования применен к этой структуре согласно разделу 13.2, кодирование в «ConcatenationClass» определяет процедуры конкатенации, а точка приложения затем переходит к каждой «EncodingStructure» в ее именованных полях.

17 Присвоения объектов кодирования

17.1 Общие положения

17.1.1 Продукцией «EncodingObjectAssignment» является:

```
EncodingObjectAssignment ::=
    encodingobjectreference
    DefinedOrBuiltinEncodingClass
    "::="
      EncodingObject
```

17.1.2 «EncodingObjectAssignment» определяет «encodingobjectreference» в качестве ссылки на объект кодирования «EncodingObject», который должен быть продукцией, генерирующей объект класса кодирования «DefinedOrBuiltinEncodingClass» (см. в D.1.2.2, D.1.7.3 и D.1.8.2 примеры присвоения объекта кодирования для разных синтаксических конструкций для описанного ниже «EncodingObject»).

17.1.3 «DefinedOrBuiltinEncodingClass» называется руководителем нотации «EncodingObject» в этой продукции.

Примечания

1 Всякий раз когда продукция «EncodingObject» появляется в ECN, имеется какой-либо руководитель, а синтаксис руководимой нотации зависит от класса кодирования руководителя.

2 Синтаксис руководимой нотации был разработан так, чтобы синтаксический анализатор мог находить его конец без сведений о руководителе.

17.1.4 Не должно быть рекурсивного определения (см. 3.2.39) для «encodingobjectreference» и не должно быть рекурсивной реализации (см. 3.2.40) для «encodingobjectreference».

17.1.5 Продукцией «EncodingObject» является:

```
EncodingObject ::=
        DefinedEncodingObject
      | DefinedSyntax
      | EncodeWith
      | EncodeByValueMapping
      | EncodeStructure
      | DifferentialEncodeDecodeObject
      | EncodingOptionsEncodingObject
      | NonECNEncodingObject
```

17.1.6 «DefinedEncodingObject» указывает на объект кодирования и определен в 10.9.2. «DefinedEncodingObject» должен быть того же класса кодирования, что и руководитель, или класса, который может быть получен из руководителя путем разыменования.

17.1.7 В настоящем стандарте выражения «тот же класс кодирования» и «тот же класс» следует понимать так: нотации, используемые для определения двух классов, должны иметь одно и то же справочное имя класса кодирования либо должны иметь справочные имена, которые разыменуют к одному и тому же имени класса кодирования.

17.1.8 Остальные продукции из «EncodingObject» определяются в последующих разделах и дают альтернативные средства определения объектов кодирования класса руководителя:

DefinedSyntax	17.2 с разделами 20—25
EncodeWith	17.3
EncodeByValueMapping	17.4
EncodeStructure	17.5
DifferentialEncodeDecodeObject	17.6
EncodingOptionsEncodingObject	17.7
NonECNEncodingObject	17.8

17.2 Кодирование с определенным синтаксисом

17.2.1 Продукция «DefinedSyntax» определена в ИСО/МЭК 8824-2, пункты 11.5 и 11.6 с учетом изменений из В.16; она используется для определения объектов кодирования для руководящего класса кодирования. Детальный синтаксис для этого выполнения определяется в разделах 23—25, а семантика конструкции определяется в разделе 22.

17.2.2 Эта нотация для определения объектов кодирования доступна только для руководящих классов кодирования в категориях (или в классе), перечисленных ниже, в таблице 3. Синтаксисом, который следует применять для каждого объекта кодирования, является «DefinedSyntax» для соответствующей категории или класса кодирования (см. определения в разделах 23—25).

Таблица 3 — Категории и классы, которые поддерживаются определенным синтаксисом

категория «вырожденная»
категория «булева»
категория «целочисленная»
категория «цепочка битов»
категория «цепочка октетов»
категория «цепочка знаков»
категория «pad»
категория «альтернативы»
категория «повторение»
категория «конкатенация»
категория «факультативные возможности»
класс **#CONDITIONAL-INT**
класс **#CONDITIONAL-REPETITION**
категория «тег»
класс **#TRANSFORM**
класс **#OUTER**

Примечания

1 Использование этого синтаксиса часто требует включения параметра для определителя. Параметризованные объекты кодирования с такими параметрами (возможно, включенные в виде части набора параметризованных объектов кодирования) будут полезны только для применения к структуре кодирования в EDM либо для включения в качестве объектов кодирования, применяемых как часть операции замены. Они не могут применяться в ELM.

2 Эта нотация дает возможность пользователю определять объекты кодирования, которые кодируют **#SET** в виде PER, который обычным образом кодирует **#SEQUENCE**, и наоборот. Ожидается, что пользователи будут ответственными за свое использование этой нотации.

17.2.3 Необходимая информация (и используемый синтаксис) для определения объекта кодирования в одной из этих категорий или классов с использованием «DefinedSyntax» описывается с помощью определений в разделах 23—25.

17.2.4 Если руководитель для значения одного из полей, появившихся в «DefinedSyntax», необходим для использования в списке фиктивных параметров, то должна быть применена нотация «EncodingClassFieldType» (определенная в В.17). Нотация «EncodingClassFieldType» не имеет других применений.

17.2.5 Когда синтаксис, определенный в разделе 23, требует наличия **REFERENCE**, это может быть обеспечено только в конструкции «DefinedSyntax» с помощью фиктивного параметра объекта кодирования, который определен, или в случае **«flag-to-be-used»** или **«flag-to-be-set»** с помощью использования справочного имени, которое представлено текстуально в определении структуры замены.

REFERENCE, что используется в качестве определителя, не должен быть названным компонентом повторения.

17.2.6 Нотация «DefinedSyntax» указывает, дается ли идентифицирующий описатель в определяемом объекте кодирования.

17.3 Кодирование с наборами объектов кодирования

17.3.1 Продукцией «EncodeWith» является:

```
EncodeWith ::=
    "{" ENCODE CombinedEncodings "}"
```

17.3.2 «CombinedEncodings» и его применение к классу кодирования определены в разделе 13.

17.3.3 Объект кодирования, определенный в «EncodeWith», является приложением «CombinedEncodings» к классу кодирования, который является руководителем (см. 17.1.3) нотации «EncodeWith».

17.3.4 Спецификация будет ошибочной, если это не выработает полную спецификацию кодирования для класса руководителя.

17.3.5 Если набор объектов кодирования в «CombinedEncodings» параметризован параметром, который является **REFERENCE**, то реальным параметром, представленным в этой конструкции, может быть только фиктивный параметр объекта кодирования, который определяется.

17.3.6 E — объект кодирования (в пределах «CombinedEncodings»), который применяется к классу руководителя. Если объект кодирования E показывает идентификационный описатель (с учетом данного набора значений описателя), то определяемый объект кодирования (см. 17.1.5) показывает тот же идентификационный описатель, как и E (с тем же набором значений описателя); иначе он не показывает описатель.

17.4 Кодирование с использованием отображений значения

17.4.1 Продукцией «EncodeByValueMapping» является:

```
EncodeByValueMapping ::=
    "{"
    USE
    DefinedOrBuiltinEncodingClass
    MAPPING
    ValueMapping
    WITH
    ValueMappingEncodingObjects
    "}"
ValueMappingEncodingObjects ::=
    EncodingObject |
    DefinedOrBuiltinEncodingObjectSet
```

17.4.2 Продукция «DefinedOrBuiltinEncodingClass» и ее семантика определены в 10.9.1. Она будет определенной пользователем структурой кодирования или предопределенным классом в группе категорий «битовое поле» (см. 16.1.7).

17.4.3 Продукция «ValueMapping» описана в 19.1.7; она будет отображением значений, связанных с руководящим классом кодирования, в класс, указанный в «DefinedOrBuiltinEncodingClass». Руководящим классом кодирования будет класс в группе категорий «битовое поле».

17.4.4 «ValueMappingEncodingObjects» описывает кодирование для «DefinedOrBuiltinEncodingClass». «EncodingObject» определяет объект кодирования с помощью нотации, руководимой этим классом, либо классом, к которому она может быть разыменована (см. 17.1.3). «DefinedOrBuiltinEncodingObject Set» может альтернативно использоваться для описания кодирования «DefinedOrBuiltinEncodingClass»; он должен содержать достаточно объектов кодирования для полного описания кодирования этого класса путем применения кодирований, определенных в разделе 13.

17.4.5 Синтаксис для «EncodingObject» позволяет как инлайновое определение объектов кодирования (рекурсивное применение этого компонента), так и использование справочных имен (в D.2.9.3

приведен пример инлайнового определения для выполнения отображения двух значений в одно присвоение).

17.4.6 Когда «EncodingObject» требует наличия **REFERENCE**, это может быть обеспечено только в этой конструкции с помощью фиктивного параметра объекта кодирования, который определен.

17.4.7 Если имеются границы или ограничения на реальный размер для полей в «DefinedOrBuilt inEncodingClass», а спецификации из раздела 19 требуют отобразить значения в такие поля, которые нарушают установленные границы или ограничения на реальный размер, то такие значения не отображаются, а кодирование таких значений будет невозможным. Если такие значения подаются для кодирования, то это будет ошибкой ECN или приложения.

17.4.8 E — объект кодирования, который применяется к «DefinedOrBuiltin-Encoding-Class». Если объект кодирования E показывает идентификационный описатель (с учетом данного набора значений описателя), то определяемый объект кодирования (см. 17.1.5) показывает тот же идентификационный описатель, как и E (с тем же набором значений описателя); иначе он не показывает описатель.

Примечание — Объект кодирования E может быть или «EncodingObject» в «ValueMappingEncodingObjects», или членом «DefinedOrBuiltinEncodingObjectSet».

17.5 Кодирование структуры кодирования

17.5.1 Продукцией «EncodeStructure» является:

```
EncodeStructure ::=
        "{"
        ENCODE STRUCTURE
        "{"
        ComponentEncodingList
        StructureEncoding ?
        "}"
        CombinedEncodings ?
        "}"
StructureEncoding ::=
        STRUCTURED WITH
        TagEncoding ?
            EncodingOrUseSet
TagEncoding ::= "[" EncodingOrUseSet "]"
EncodingOrUseSet ::=
            EncodingObject |
            USE-SET
```

17.5.2 «EncodeStructure» может использоваться для определения кодирования, если только управляющий класс кодирования разыменовал конструкцию, определенную с помощью конструктора кодирования в категориях «альтернативы», «конкатенация» или «повторение», либо конструкцию, определенную с помощью одной из этих категорий, которой предшествует класс в категории «тег». Такой конструктор кодирования называется управляющим конструктором кодирования.

17.5.3 Продукция «StructureEncoding», если она присутствует, будет определять кодирование для управляющего конструктора кодирования и для любого предшествующего класса в категории «тег», который находится перед управляющим конструктором кодирования. Если эта продукция отсутствует, то «CombinedEncodings» должен присутствовать и содержать объекты кодирования, которые могут кодировать управляющий конструктор кодирования и любой предшествующий класс в категории «тег»; в противном случае спецификация ECN будет ошибочной.

Примечание — «CombinedEncodings» должен присутствовать, если «StructureEncoding» отсутствует, так как должно выполняться полное кодирование. Если желательно отложить спецификацию части кодирования, то следует применить фиктивный параметр.

17.5.4 Если «ComponentEncodingList» не пуст, то объект кодирования, приложенный к управляющему конструктору кодирования (либо от «StructureEncoding», либо от «CombinedEncodings»), не определяет каких-либо действий замены.

17.5.5 Если «EncodingOrUseSet» в «StructureEncoding» имеет значение «EncodingObject», то он будет управляться управляющим конструктором кодирования.

17.5.6 Если в каком-либо «EncodingOrUseSet» указан **USE-SET**, то кодирование соответствующего класса достигается путем применения «CombinedEncodings», который должен присутствовать и должен быть достаточным для кодирования соответствующего класса; в противном случае спецификация ECN будет ошибочной.

17.5.7 Продукцией «ComponentEncodingList» является:

```
ComponentEncodingList ::=
    ComponentEncoding "," *

ComponentEncoding ::=
        NonOptionalComponentEncodingSpec |
        OptionalComponentEncodingSpec
```

17.5.8 Должен быть один самый большой «ComponentEncoding» для каждого компонента управляющего конструктора кодирования. «ComponentEncodings» должен быть в том же текстуальном порядке.

Примечание — Отсутствие «ComponentEncoding» может быть обнаружено по последующим именованным полям либо по концу «ComponentEncodingList».

17.5.9 «OptionalComponentEncodingSpec» используется, если, и только если, компонент является факультативным (то есть содержит класс кодирования в категории «факультативные возможности»).

17.5.10 Если «ComponentEncoding» для какого-либо компонента отсутствует в «ComponentEncodingList», то должен присутствовать «CombinedEncodings» (но см. также 17.5.6), который требуется при приложении к компоненту (см. 13.2) для обеспечения полного кодирования этого компонента (возможно, включая использование фиктивных параметров); в противном случае спецификация ECN будет ошибочной.

```
NonOptionalComponentEncodingSpec ::=
            identifier ?
            TagAndElementEncoding
OptionalComponentEncodingSpec ::=
            identifier
            TagAndElementEncoding
            OPTIONAL-ENCODING
            OptionalEncoding
TagAndElementEncoding ::=
        TagEncoding ?
        EncodingOrUseSet
OptionalEncoding ::= EncodingOrUseSet
```

17.5.11 Элементом «identifier» здесь должен быть «identifier» компонента управляющего конструктора кодирования. Элемент «identifier» в «NonOptionalComponentEncodingSpec» должен отсутствовать, если, и только если, управляющий конструктор кодирования является классом в категории «повторение», для которого нет идентификатора в повторяемом элементе.

17.5.12 «TagAndElementEncoding» в «ComponentEncoding» должен обеспечить полное кодирование для компонента (включая любой класс в категории «тег», который предшествует элементу, но исключая любой класс в категории «факультативные возможности», который следует за элементом).

17.5.13 «EncodingObject» из «EncodingOrUseSet» в «TagAndElementEncoding» будет руководиться соответствующим классом кодирования в компоненте. Если «EncodingOrUseSet» будет **USE-SET**, то кодирование достигается путем применения «CombinedEncodings» (который должен присутствовать).

17.5.14 «EncodingOrUseSet» в «OptionalEncoding» должен полностью кодировать класс в категории «факультативные возможности» компонента. Если «EncodingOrUseSet» будет **USE-SET**, то кодирование класса в категории «факультативные возможности» достигается путем применения «CombinedEncodings» (который должен присутствовать).

17.5.15 Если **REFERENCE** необходим в качестве реального параметра какого-либо объекта кодирования или набора объектов кодирования, используемого в этой продукции, то он может поставляться либо в виде фиктивного параметра определяемого объекта кодирования, либо в виде «ComponentIdList» (см. 15.3.1 для синтаксиса «ComponentIdList», значение «ComponentIdList» в данном контексте описано ниже).

17.5.16 Если руководитель не является конструктором в категории «повторение», то первый (или единственный) «identifier» в «ComponentIdList» должен быть «identifier» текстуально присутствующего «NamedType» (на определенном уровне вложенности — см. 17.5.17) конструкции, достигаемой путем разыменования руководителя. Это идентифицирует все определение этого «NamedType» компонента независимо от того, присутствует ли определение текстуально.

17.5.17 Если существует более одного такого подходящего идентификатора, то выбирается первый подходящий идентификатор в процессе сканирования (в текстовом порядке) идентификаторов внешнего уровня, затем — идентификаторов второго уровня, затем — третьего уровня и т. д.

17.5.18 Каждый последующий «identifier» из «ComponentIdList» (если таковые имеются) должен быть «identifier» в «NamedType» структуры, идентифицированной по предыдущей части «ComponentIdList», и идентифицирует все определение этого «NamedType» компонента независимо от того, присутствует ли определение структуры, идентифицированной по предыдущей части «ComponentIdList», текстуально.

17.5.19 Если руководитель является конструктором в категории «повторение», то реальный параметр для **REFERENCE** должен быть «ComponentIdList», чей первый «identifier» идентифицирует компонент, который текстуально присутствует в «EncodingStructure» в «RepetitionStructure», достигаемой путем разыменования повторения (см. 17.5.17). Затем применяются 17.5.17 и 17.5.18.

17.5.20 Если **REFERENCE** затребован для идентификации контейнера, то он может быть также подан как:

a) **STRUCTURE** (при условии, что конструктор для кодируемой структуры не находится в категории «альтернативы»), когда он указывает на эту структуру;

b) **OUTER**, когда он указывает на контейнер полного кодирования.

Примечание — «EncodeStructure» является единственной продукцией, в которой элементы **REFERENCE** могут быть представлены, за исключением случаев использования фиктивных параметров или использования **OUTER** или случаев, когда имеются ссылки для поддержки элемента **flag-to-be-used**» или **flag-to-be-set** в определении объекта кодирования для класса в категории «повторение», которая использует замену.

17.5.21 Распознавать, что определяемый объект кодирования (см. 17.1.5) показывает идентификационный описатель, следует следующим образом:

a) если «TagEncoding» присутствует в «StructureEncoding», E — объект кодирования, который применяется для класса кодирования в категории «тег»; или

b) если «TagEncoding» не присутствует в «StructureEncoding», E — объект кодирования, который применяется для управляющего конструктора кодирования (это может быть или «EncodingObject» в «EncodingOrUseSet», в «StructureEncoding», или может быть членом «CombinedEncodings»).

Если объект кодирования E показывает идентификационный описатель (с данным набором значений описателя), то определяемый объект кодирования показывает такой же идентификационный описатель, как и E (с тем же набором значений описателя); иначе он не показывает описатель.

17.6 Дифференциальное кодирование-декодирование

17.6.1 Продукцией «DifferentialEncodeDecodeObject» является:

```
DifferentialEncodeDecodeObject ::=
        "{"
        ENCODE-DECODE
        SpecForEncoding
        DECODE AS IF
        SpecForDecoders
        "}"
SpecForEncoding ::= EncodingObject
SpecForDecoders ::= EncodingObject
```

17.6.2 «DifferentialEncodingObject» определяет правила кодирования абстрактных значений, связанных с классом руководителя этой нотации, и (отдельно) правила, используемые декодерами для восстановления абстрактных значений из кодовых комбинаций, которые, как предполагается, созданы объектами кодирования с классом руководителя.

17.6.3 «SpecForEncoding» применяется кодерами. Декодеры будут декодировать так, как будто бы кодер применил «SpecForDecoders».

Примечания

1 «SpecForDecoders» является, тем не менее, спецификацией кодирования. Он предлагает декодерам предполагать, что кодеры использовали эту спецификацию.

2 Поведение декодеров, которые декодируют в предположении, что кодер использовал «SpecForDecoders», но обнаруживают ошибки кодирования, не стандартизовано.

17.6.4 Объекты кодирования «SpecForEncoding» и «SpecForDecoders» не должны определяться с помощью **ENCODE-DECODE**, и любые объекты кодирования, использованные в этом определении, не должны определяться с помощью **ENCODE-DECODE**.

Примечание — Это ограничение сделано потому, что в противном случае спецификация смысла конструкции кодирования/декодирования стала бы более сложной, не давая дополнительных функциональных возможностей.

17.6.5 Если «SpecForEncoding» и «SpecForDecoders» показывают одинаковый идентификационный описатель с одинаковым набором значений описателя, то определяемый объект кодирования (см. 17.1.5) показывает этот идентификационный описатель (с тем же набором значений описателя); иначе он не показывает описатель.

17.7 Факультативные возможности кодирования

17.7.1 Продукцией «EncodingOptionsEncodingObject» является:

```
EncodingOptionsEncodingObject ::=
    "{"
    OPTIONS
    EncodingOptionsList
    WITH AlternativesEncodingObject
    "}"
EncodingOptionsList ::= OrderedEncodingObjectList
AlternativesEncodingObject ::= EncodingObject
```

17.7.2 «EncodingOptionsEncodingObject» указывает, что кодер может кодировать (согласно 17.7.6) с помощью какого-либо «EncodingObject» из «EncodingOptionsList». Все эти «EncodingObject» должны быть объектами кодирования из управляющего класса.

Примечание — Новые реализации настоятельно рекомендуется кодировать с использованием такого самого раннего объекта «EncodingObject» из упорядоченного списка, который способен кодировать абстрактное значение, подлежащее кодированию (см. 17.7.6). Спецификация факультативных возможностей кодирования дается лишь потому, что необходимо отразить факультативные возможности, которые обеспечиваются в традиционных протоколах, и поддержать различные формы кодирования длины цепочек. Все факультативные возможности могут, конечно, появляться при декодировании.

17.7.3 «AlternativesEncodingObject» должен быть объектом кодирования любого класса в категории «альтернативы», а кодеры и декодеры должны использовать кодовые последовательности и процедуры, указанные этим объектом кодирования, так, как будто бы факультативные возможности кодирования были кодированы для компонентов экземпляра этого класса. «AlternativesEncodingObject» не должен содержать спецификации **REPLACE** (см. 23.1.1). Параметр **DETERMINED BY** устанавливается в **handle**, а также указывается идентификационный описатель.

Примечание — Если «AlternativesEncodingObject» параметризован параметром «поле ссылки», то определяемый «encodingobjectreference» должен быть параметризован параметром «фиктивное поле ссылки», который используется в качестве реального параметра для «AlternativesEncodingObject».

17.7.4 Все «EncodingObject» в «EncodingOptionsList» должны показывать такой идентификационный описатель, и их наборы значений описателей должны быть непересекающимися.

17.7.5 Если «AlternativesEncodingObject» показывает идентификационный описатель (с данным набором значений описателя), то определяемый объект кодирования (см. 17.1.5) показывает такой же идентификационный описатель (с тем же набором значений описателя), иначе он не показывает описатель.

Примечание — Идентификационный описатель, показываемый «AlternativesEncodingObject» (если есть), не имеет отношения к идентификационному описателю, показываемому «EncodingObject» в «EncodingOptionsList», даже если они имеют одинаковое имя.

17.7.6 Кодер должен ограничивать свой выбор «EncodingObject» из «EncodingOptionsList» такими объектами, которые обеспечивают кодирования для кодируемого реального абстрактного значения. Если отсутствует по крайней мере один такой «EncodingOptionsList» для какого-либо абстрактного значения, подлежащего кодированию, то спецификация ECN или приложение будут ошибочными.

Примечания

1 Возможно, что наборы абстрактных значений, кодируемых с помощью объектов «EncodingObject» из «EncodingOptionsList», разъединены. Это не является ошибкой; возможен удобный путь для спецификации разных структур кодирования разных зон абстрактных значений в управляющем классе, например, короткая форма и длинная форма кодирований, где короткая форма обязательна для небольших значений.

2 Возможно использовать объекты кодирования опций кодирования, такие как «SpecForDecoders» (см. 17.6) («SpecForEncoding» является объектом кодирования опций кодирования, который содержит точно одну опцию из «SpecForDecoders»). Это является другим подходом к расширяемости.

17.8 Не-ECN определение объектов кодирования

17.8.1 Продукцией «NonECNEncodingObject» является:

```
NonECNEncodingObject::=
    NON-ECN-BEGIN
    AssignedIdentifier
    anystringexceptnonecnend
    NON-ECN-END
```

17.8.2 «NonECNEncodingObject» определяет объект кодирования класса руководителя (см. 17.1.3). Нотация для выполнения этого содержится в «anystringexceptnonecnend» и не стандартизована.

17.8.3 Продукция «AssignedIdentifier» и ее семантика определены в ИСО/МЭК 8824-1, подраздел 13.1 с учетом изменений А.1. Она указывает нотацию, используемую в «anystringexceptuserdefinedend» для описания кодирования.

17.8.4 Если использована альтернатива «empty» из «AssignedIdentifier», то нотация определяется с помощью средств, не рассматриваемых в настоящем стандарте.

17.8.5 Присвоение идентификаторов объекта для любой нотации, используемой в «anystringexceptnonecnend», производится по обычным правилам присвоения идентификаторов объекта, определенным в серии ИСО/МЭК 9834.

17.8.6 Идентификационный описатель (с данным набором значений описателя) показывается определяемым объектом кодирования (см. 17.1.5), если, и только если, «anystringexceptnonecnend» указывает, что это так. Средства для такой спецификации не определяются в настоящем стандарте.

18 Назначения наборов объектов кодирования

18.1 Общие сведения

18.1.1 Продукцией «EncodingObjectSetAssignment» является:

```
EncodingObjectSetAssignment ::=
    encodingobjectsetreference
    #ENCODINGS
    "::="
    EncodingObjectSet
    CompletionClause ?
```

```
EncodingObjectSet ::=
        DefinedOrBuiltinEncodingObjectSet |
        EncodingObjectSetSpec
```

18.1.2 Нотация «EncodingObjectSet» управляется зарезервированным словом **#ENCODINGS** и должна удовлетворять условиям, приведенным ниже.

18.1.3 Не должно быть рекурсивного определения (см. 3.2.39) элемента «encodingclassreference» и не должно быть рекурсивной реализации (см. 3.2.40) элемента «encodingclassreference».

18.1.4 «DefinedOrBuiltinEncodingObjectSet» определен в 10.9.3.

18.1.5 Продукцией «EncodingObjectSetSpec» является:

```
EncodingObjectSetSpec ::=
        "{"
        EncodingObjects UnionMark *
        "}"
EncodingObjects ::=
        DefinedEncodingObject |
        DefinedEncodingObjectSet
UnionMark ::=
        "|" |
        UNION
```

18.1.6 «EncodingObjectSetSpec» определяет набор объектов кодирования, используя один или несколько объектов кодирования или наборов объектов кодирования.

18.1.7 Все объекты кодирования, формирующие набор объектов кодирования, должны иметь разные классы кодирования; они не должны быть классами из группы категорий «процедура кодирования», кроме случаев, когда они имеют класс **#OUTER** (см. 16.1.13).

Примечание — Набор объектов кодирования используется для определения других наборов объектов кодирования, для определения объектов кодирования в EDM и для импорта в ELM при применениях кодирований.

18.1.8 Если присутствует «CompletionClause», то набор объектов кодирования, определенный в «EncodingObjectSetSpec», рассматривается в качестве «PrimaryEncodings» (см. 13.2), а набор объектов кодирования, назначенный для «encodingobjectsetreference», является комбинированным набором объектов кодирования, сформированным согласно 13.2.

18.2 Предопределяемые наборы объектов кодирования

18.2.1 Продукцией «BuiltinEncodingObjectSetReference» является:

```
BuiltinEncodingObjectSetReference ::=
              PER-BASIC-ALIGNED
            | PER-BASIC-UNALIGNED
            | PER-CANONICAL-ALIGNED
            | PER-CANONICAL-UNALIGNED
            | BER
            | CER
            | DER
```

18.2.2 Эти имена наборов объектов кодирования указывают наборы объектов кодирования, определенные в ИСО /МЭК 8825-1 и ИСО /МЭК 8825-2. Идентификаторы объектов для правил кодирования, обеспечивающих эти наборы объектов кодирования, приводятся в таблице 4.

Примечание — Эти стандарты были написаны до настоящего стандарта об ECN и не используют терминологию объектов кодирования. Они определяют, например, способ кодирования типа ACH.1 **INTEGER** или **BOOLEAN**. Это следует интерпретировать как определение объекта кодирования класса **#INTEGER** или класса **#BOOLEAN**.

Таблица 4 — Имена предопределенных наборов объектов кодирования и связанные идентификаторы объектов

Имя	Идентификатор
PER-BASIC-ALIGNED	**{joint-iso-itu-t(2) asn1(1) packed-encoding(3) basic(0) aligned(0)}**
PER-BASIC-UNALIGNED	**{joint-iso-itu-t(2) asn1(1) packed-encoding(3) basic(0) unaligned(1)}**
PER-CANONICAL-ALIGNED	**{joint-iso-itu-t(2) asn1(1) packed-encoding(3) canonical(1) aligned(0)}**
PER-CANONICAL-UNALIGNED	**{joint-iso-itu-t(2) packed-encoding(3) canonical(1) unaligned(1)}**
BER	**{joint-iso-itu-t(2) asn1(1) basic-encoding(1)}**
CER	**{joint-iso-itu-t(2) asn1(1) ber-derived(2) canonical-encoding(0)}**
DER	**{joint-iso-itu-t(2) asn1(1) ber-derived(2) distinguished-encoding(1)}**

18.2.3 Каждый из этих наборов объектов кодирования является полным набором объектов кодирования, который может быть применен к любой структуре кодирования (либо неявно генерируемой из типа ACH.1, либо определяемой пользователем), с соответствующим разыменованием, чтобы определить соответствующие кодирования BER или PER.

Примечание — Объект кодирования для определяемого пользователем или неявно генерируемого класса кодирования может быть добавлен к такому набору и будет иметь приоритет перед любым кодированием, которое могло бы получиться с помощью разыменования.

18.2.4 Все вышеприведенные наборы содержат объекты кодирования для классов, используемых в неявно генерируемых структурах кодирования (см. 11.2), которые отличаются для каждого набора правил кодирования. Каждый из них содержит также идентичные объекты кодирования для классов **#INT**, **#BOOL**, **#NUL**, **#CHARS**, **#OCTETS**, **#BITS**, **#CONCATENATION**. Они не содержат объектов кодирования для **#ALTERNATIVES**, **#REPETITION** и **#PAD**.

18.2.5 Эти классы кодирования представляют базовые строительные блоки кодирований и просто кодируются всеми вышеприведенными предопределенными наборами объектов кодирования. Объекты кодирования для этих классов определяют кодирования следующим образом.

18.2.5.1 **#INT** кодируется в виде кодирования **PER-BASIC-UNALIGNED #INTEGER** при условии, что он имеет границы. Если **#INT** не имеет нижней и верхней границ, когда этот объект кодирования применяется к **#INT**, то это будет ошибкой разработки ECN.

18.2.5.2 **#BOOL** и **#NUL** кодируются в виде **PER-BASIC-UNALIGNED #BOOLEAN** и **#NULL** соответственно.

18.2.5.3 **#CHARS**, **#OCTETS** и **#BITS** кодируются в виде **PER-BASIC-UNALIGNED UTF8String**, **#OCTETSTRING** и **#BIT-STRING** соответственно при условии, что они имеют единый размер. Если **#CHARS**, **#OCTETS** или **#BITS** не имеют реального указания размера, ограничивающего их до единого размера, то это будет ошибкой разработки ECN.

18.2.5.4 **#CONCATENATION** кодируется в виде **PER-BASIC-UNALIGNED** кодирования **#SEQUENCE** без факультативных компонентов. Если эти объекты кодирования применены к **#CONCATENATION** с факультативными компонентами, то это будет ошибкой спецификации ECN.

18.2.6 Объекты кодирования **#OPEN-TYPE** в предопределенных наборах объектов кодирования BER, CER и DER не вырабатывают дополнительного кодирования для класса **#OPEN-TYPE**. Когда эти объекты кодирования применяются к классу в категории «открытый тип», появится ошибка спецификации ECN, если кодирования значений выборочного типа (в экземпляре связи) для использования с классом **#OPEN-TYPE** не являются саморазграничивающими.

Примечание — Комбинированный набор объектов кодирования, примененный этими объектами кодирования к выборочному типу для использования с классом **#OPEN-TYPE**, всегда будет тем же, что и комбинированный набор объектов кодирования, примененный к классу **#OPEN-TYPE**, так как эти объекты кодирования не содержат **ENCODED WITH** [см. перечисление d) 13.2.10.5 и 13.2.9].

19 Отображение значений

19.1 Общие положения

19.1.1 В настоящем разделе описывается синтаксис для отображения значений (и номеров тегов), которые кодируются полями одной структуры кодирования (она может быть генерируемой структурой кодирования или любой другой структурой кодирования) в поля другой структуры кодирования.

Примечание — Возможности, которые обеспечиваются при отдельном использовании этой нотации, ограничены (во избежание усложнения). Более сложные отображения могут быть достигнуты с помощью нескольких экземпляров «EncodeByValueMapping» (см. 17.4 и пример в D.1.10.2). Эти механизмы отображения могут быть расширены и обобщены, но это не будет делаться, пока не определены дальнейшие требования пользователей.

19.1.2 В спецификации нотации «EncodeByValueMapping» (см. 17.4.1) структура, к которой разыменует «DefinedOrBuiltinEncodingClass» в «EncodingObjectAssignment» (см. 17.1.1), частью которого она является, называется руководителем источника или классом кодирования источника (в зависимости от контекста). Структура, к которой «DefinedOrBuiltinEncodingClass» в «EncodeByValueMapping» сам разыменуется, называется руководителем цели или классом кодирования цели (в зависимости от контекста).

19.1.3 Если руководитель источника имеет начальный класс в категории «тег», то руководитель цели должен иметь начальный класс в категории «тег», а номер тега класса в руководителе источника отображается в номер тега класса в категории «тег» в руководителе цели. Если класс в категории «тег» в руководителе цели имеет связанный номер тега, то будет ошибка спецификации ECN, когда он отличается от номера тега, отображенного из руководителя источника.

19.1.4 Если руководитель источника не имеет начального класса в категории «тег», то руководитель цели не должен иметь начального класса в категории «тег», но если он его имеет, то в определении руководителя цели должен быть номер тега, связанный с этим тегом.

19.1.5 Влияние наличия начального класса в категории «тег» в руководителях источника или цели полностью определяется в 19.1.3 и 19.1.4, а в последующем тексте возможное присутствие таких классов игнорируется.

19.1.6 Кодирования, определенные для значений, отображенных в класс кодирования цели, становятся кодированиями таких значений в классе кодирования источника.

Примечания

1 Если полная спецификация ECN отображает только некоторые из значений типа ACH.1 в кодирования, то это не будет ошибкой. Это является ограничением, наложенным ECN на значения, которые могут использоваться приложением. Такие ограничения обычно указываются в комментариях или в спецификации ACH.1 или в спецификации ECN (см. 17.4.7).

2 Если полная спецификация ECN отображает два значения в одно и тоже кодирование, образованное единичным объектом кодирования, то это будет ошибкой спецификации ECN. Такие ошибки могут обнаруживаться средствами ECN, но правила их избежания не завершены в настоящем стандарте, а ответственность остается за пользователем ECN.

19.1.7 Продукцией «ValueMapping» является:

```
ValueMapping ::=
        MappingByExplicitValues
      | MappingByMatchingFields
      | MappingByTransformEncodingObjects
      | MappingByAbstractValueOrdering
      | MappingByValueDistribution
      | MappingIntToBits
```

Примечание — Перед любым местоположением этого синтаксиса помещается зарезервированное слово **MAPPING** (в D.1.2.2, D.1.4.2, D.1.10.2, D.2.1.3 и в приложении А приведены примеры определения кодирований с использованием каждого из этих отображений значения).

19.1.8 Продукции «ValueMapping» определяются в следующих разделах:

MappingByExplicitValues	19.2;
MappingByMatchingFields	19.3;
MappingByTransformEncodingObjects	19.4;
MappingByAbstractValueOrdering	19.5;

MappingByValueDistribution 19.6;
MappingIntToBits 19.7.

Примечание — Часто случается, что несколько отображений значения могут использоваться для определения одного и того же кодирования, но некоторые будут давать более ясную и менее многословную спецификацию, чем другие. Разработчикам ECN следует осторожно выбирать форму отображения значений для использования.

19.2 Отображение с помощью явных значений

19.2.1 В настоящем разделе приведена нотация для спецификации отображения значений между различными простейшими классами кодирования «битовое поле» (в D.1.10.2 приведен пример).

19.2.2 В этом разделе используется нотация для значений ACH.1 (нотация значения ACH.1), описанная в ИСО/МЭК 8824-1 для типа, который соответствует классу кодирования.

19.2.3 В таблице 5 описывается нотация значения ACH.1, которую следует использовать с каждым управляющим классом кодирования. В каждом случае класс может иметь или не иметь связанное ограничение на размер или на значение диапазона.

19.2.4 ECN поддерживает отображение явных значений (в класс кодирования или из него) для всех классов кодирования в категориях, перечисленных в столбце 1 таблицы 5. В столбце 2 таблицы 5 показаны нотации значения (в виде продукции ACH.1, или в виде ссылки на раздел ИСО/МЭК 8824-1, или в обоих видах), которые должны применяться, когда класс кодирования, приведенный в столбце 1, указан в качестве управляющего для нотации. Он определяет также раздел ИСО/МЭК 8824-1, описывающий нотацию значения.

Примечание — Никакая из нижеуказанных нотаций значения ACH.1 не может использовать «DefinedValue» (как определено в ИСО/МЭК 8824-1, подраздел 14.1), так как «valuereference» не может быть импортирован или определен в модуле EDM или ELM.

Таблица 5 — Категории классов кодирования и нотации значения, используемые при отображении явных значений

Категория управляющего класса кодирования	Нотация значения ACH 1
bitstring	**«bstring» или «hstring»** (ИСО/МЭК 8824-1, подразделы 12.10 и.12.12)
boolean	**«BooleanValue»** (ИСО/МЭК 8824-1, пункт 18.3)
characterstring	**«RestrictedCharacterStringValue»** (ИСО/МЭК 8824-1, пункт 41.8)
enteger	**«SignedNumber»** (ИСО/МЭК 8824-1, пункт 19.1)
null	**«NullValue»** (см. ИСО/МЭК 8824-1, пункт 24.3)
objectidentifier	**«DefinitiveIdentifier»** (см. А.1)
octetstring	**«bstring» или «hstring»** (см. ИСО/МЭК 8824-1, подразделы 12.10 и 12.12)
real	**«RealValue»** (см. ИСО/МЭК 8824-1, пункт 21.6)
time	**«TimeValue»** (см. ИСО/МЭК 8824-1, подпункт 38.3.2)

19.2.5 Продукцией «MappingByExplicitValues» является:

MappingByExplicitValues ::=
VALUES

```
        "{"
        MappedValues "," +
        "}"
    MappedValues ::=
        MappedValue1
        TO
        MappedValue2
    MappedValue1 ::= Value
    MappedValue2 ::= Value
```

19.2.6 «MappedValue1» — это нотация значения, управляемая управляющим источника, а «MappedValue2» — это нотация значения, управляемая управляющим цели (см. 19.1.2). Значение в источнике, определяемое «MappedValue1», отображается в значение в цели, определенное «MappedValue2».

19.2.7 Спецификация ECN будет ошибочной, если «MappedValue2» является значением, которое нарушает ограничение на границу или на размер в цели.

19.3 Отображение с помощью полей сопоставления

19.3.1 Это отображение предусмотрено первоначально для создания возможности кодирования типа ACH.1, определяемого как кодирование структуры кодирования, которая имеет поля, соответствующие компонентам этого типа, но также имеет поля для определителей.

19.3.2 Продукцией «MappingByMatchingFields» является:

```
MappingByMatchingFields ::=
    FIELDS
```

19.3.3 Если классы кодирования источника или цели являются определяемыми пользователем структурами кодирования (см. 9.2.2.3) либо генерируемыми структурами кодирования, то эти ссылки различаются до того, как источник и цель стартуют с конструктором кодирования. Если этот конструктор кодирования в цели будет в категории «повторение», то разыменование компонента этого конструктора кодирования «повторение» выполняется до того, как этот компонент стартует с конструктором кодирования. Ссылки внутри результирующих структур не различаются.

19.3.4 Влияние возможного наличия классов в категории «тег» при начальном разыменовании имен «DefinedOrBuiltinEncodingClass» в источнике и цели было полностью описано в 19.1.3—19.1.5. Спецификация ECN будет ошибочной, если дальнейшие начальные классы в категории «тег» вводятся путем применения 19.3.3.

19.3.5 После применения 19.3.3 классы кодирования источника и цели стартуют с одним и тем же конструктором кодирования. Это должен быть либо конструктор кодирования в категории «конкатенация», либо конструктор кодирования в категории «повторение». Если этот конструктор кодирования будет в категории «повторение», то его компоненты в цели должны быть классом в категории «конкатенация». В настоящем пункте результирующие структуры кодирования называются структурами кодирования источника и цели соответственно.

19.3.6 Имена полей компонентов (верхнего уровня) конструктора кодирования, образованного путем применения 19.3.3 к источнику, называются полями источника.

Примечание — Поля источника ограничены до полей верхнего уровня конкатенации или компонентов повторения. Это ограничение накладывается на каждую реализацию ECN и может быть ослаблено в будущем.

19.3.7 Имена полей компонентов конструктора кодирования в категории «конкатенация», образованного путем применения 19.3.3 к цели, называются потенциальными полями цели.

Примечание — Потенциальные поля цели могут быть либо компонентами конкатенации верхнего уровня, либо компонентами «конкатенации», которые являются компонентами «повторения».

19.3.8 Для каждого поля источника будет потенциальное поле цели с тем же именем поля (сопоставляющее поле цели).

Примечание — Компонент класса «повторение» может быть отображен только в случае, если в него включен идентификатор (сопоставляющий индикатор в цели). Использование отображения с помощью полей сопоставления было бы неправомерным при отсутствии опознавателя.

19.3.9 Сопоставляющее поле цели будет факультативным элементом в конкатенации, если, и только если, его поле источника является факультативным элементом в конкатенации, а наличие или отсутствие поля источника в абстрактном значении, связанном со структурой кодирования источника, определяет наличие или отсутствие поля цели в структуре кодирования цели.

19.3.10 Если поле источника имеет начальный класс в категории «тег», то сопоставляющее поле цели должно иметь начальный класс в категории «тег», а номер тега класса в поле источника отображается в номер тега класса в категории «тег» в сопоставляющем поле цели. Если класс в категории «тег» в сопоставляющем поле цели имеет связанный номер тега, то спецификация ECN будет ошибочной, когда он отличается от номера тега, отображаемого из поля источника.

19.3.11 Если поле источника не имеет начального класса в категории «тег», то отображающее поле цели не обязано иметь начальный класс в категории «тег», но если оно его имеет, то должен быть номер тега, связанный с этим тегом в определении сопоставляющего поля цели.

19.3.12 Не считая наличия или отсутствия класса в категории «тег» и факультативных категорий (как указано в 19.3.9—19.3.11), сопоставляющее поле цели и поле источника будут иметь один и тот же класс кодирования (см. 17.1.7) или будут определены с помощью одной и той же последовательности текстуальных элементов, игнорируя комментарий и пробел, а также спецификации границ.

19.3.13 Все абстрактные значения отображаются из каждого поля источника в сопоставляющие поля цели. Дополнительные поля в структуре кодирования цели не получают абстрактных значений. В правильных спецификациях ECN значения таких полей должны определяться с помощью ссылки в виде определителя.

19.3.14 Если конструкторы кодирования источника и цели являются классами в категории «повторение», то число повторений в абстрактном значении, связанном со структурой кодирования источника, отображается в число повторений в структуре кодирования цели.

19.3.15 Когда поле источника имеет ограничение на связанное содержимое, оно отображается в сопоставляющее поле цели в виде ограничения как связанное содержимое.

19.3.16 Если из-за наличия ограничений на границы или размер имеются значения в поле источника, которые не присутствуют в сопоставляющем поле цели, то следует применять 17.4.7.

19.4 Отображение с помощью объектов кодирования #TRANSFORM

19.4.1 Это отображение позволяет применять один или несколько объектов кодирования **#TRANSFORM** для выполнения отображения.

19.4.2 Класс кодирования **#TRANSFORM** определяется в разделе 24. Он дает возможность описывать объекты кодирования, которые будут преобразовывать абстрактные значения источника в абстрактные значения результата. Правила формирования упорядоченного списка преобразований (для «OrderedTransformList») описываются в разделе 24. Полный список определен для преобразования из источника в результат.

Примечание — Примеры отображений, определенных с этими преобразованиями, приведены в D.1.2.2 и D.2.4.2. Пример в D.1.6.3 показывает использование этой продукции для определения двоично-десятичного кодирования (BCD) целого числа ACH.1.

19.4.3 Продукцией «MappingByTransformEncodingObjects» является:

```
MappingByTransformEncodingObjects ::=
    TRANSFORMS
    "{"
    OrderedTransformList
    "}"
OrderedTransformList ::= Transform "," +
Transform ::= EncodingObject
```

19.4.4 Все «EncodingObject» в «OrderedTransformList» должны управляться классом кодирования **#TRANSFORM**.

19.4.5 Классы цели и источника для этого отображения (см. 19.1.2) должны быть категории «цепочка битов», «булева», «цепочка знаков», «целочисленная» или «цепочка октетов». Источник первого преобразования в списке и результат последнего преобразования в списке должны быть согласованы с категориями источника и цели, описанными в 24.2.7.

19.4.6 Спецификация или применение ECN будут ошибочными, если любой «Transform» в «OrderedTransformList» не является обратимым для отображаемого абстрактного значения.

Примечание — В разделе 24 для каждого преобразования определены абстрактные значения, для которых указана их обратимость.

19.4.7 Если имеются ограничения на границы и на реальный размер для класса кодирования цели, то применяется 17.4.7.

19.5 Отображение с помощью упорядочения абстрактных значений

19.5.1 Это отображение позволяет распределять абстрактные значения, связанные с простыми классами кодирования, в поля сложных структур кодирования и отображать абстрактные значения, связанные со сложными структурами кодирования, в простые классы кодирования, такие как **#INT**. Оно допускает также уплотнение целочисленных значений или перечислений в непрерывный набор целочисленных значений (см. D.1.4).

Примечание — Номера тегов, связанные с классами в категории «тег», не являются абстрактными значениями.

19.5.2 Продукцией «MappingByAbstractValueOrdering» является:

MappingByAbstractValueOrdering ::=
ORDERED VALUES

19.5.3 При этом отображении все имена классов кодирования являются разыменованными (рекурсивно), а результат должен быть классом в категории «нуль», «булева», «целочисленная» или «действительное число», либо конструкцией, использующей класс в категории «альтернативы», либо классом в категории «конкатенация», который имеет одиночный нефакультативный компонент.

19.5.4 Упорядоченный набор значений может быть конечным или бесконечным.

19.5.4.1 Конечный набор упорядоченных абстрактных значений определяется для классов кодирования в следующих категориях:

a) «вырожденная»;

b) «булева»;

c) ограниченная «целочисленная»;

d) «действительное число», ограниченное до конечного числа значений;

e) «структура кодирования», определенная с использованием категории «альтернативы», при условии, что все альтернативы имеют определенное конечное упорядочение;

f) «структура кодирования», определенная с использованием категории «конкатенация», которая имеет одиночный нефакультативный компонент, при условии, что этот компонент имеет определенное конечное упорядочение.

19.5.4.2 Бесконечный набор упорядоченных абстрактных значений определяется для классов кодирования в следующих категориях:

a) «целочисленная», ограниченная наличием конечной нижней границы;

b) «структура кодирования», определенная с использованием категории «альтернативы», при условии, что все альтернативы, за исключением последней, определяются как имеющие конечный набор упорядоченных значений, а последняя альтернатива определяется как имеющая бесконечный набор упорядоченных значений;

c) «структура кодирования», определенная с использованием категории «конкатенация», которая имеет одиночный нефакультативный компонент, при условии, что этот компонент определяется как имеющий бесконечный набор упорядоченных значений.

19.5.5 Классы в категории «нуль» имеют одиночное абстрактное значение. Классы в категории «булева» определяются как имеющие **TRUE** перед **FALSE**. Классы в «целочисленной» категории определяются как имеющие более высокие значения целого числа, следующие за низкими значениями целого числа. Классы в категории «действительное число» определяются как имеющие более высокие значения, следующие за низкими значениями.

Примечание — Число абстрактных значений, связанных с классом в «целочисленной» категории, не обязательно конечно.

19.5.6 Любые границы, присутствующие в источнике или в пункте назначения, должны полностью учитываться при определении упорядоченного набора абстрактных значений.

19.5.7 Упорядочение абстрактных значений, связанных с классом в категории «альтернативы» (у которого все альтернативы имеют определенное упорядочение абстрактных значений), определяется как абстрактные значения (упорядоченные) из первой альтернативы в тексте, за которыми следуют значения из второй альтернативы в тексте, и т. д. до последней альтернативы в тексте.

19.5.8 Упорядочение абстрактных значений, связанных с классом в категории «конкатенация», который имеет одиночный нефакультативный компонент, должно быть порядком, определенным упорядочением абстрактных значений его единственного компонента.

19.5.9 Отображение определяется из абстрактных значений первого класса кодирования в абстрактные значения второго класса кодирования с помощью их позиций в вышеуказанном упорядочении.

19.5.10 Заметим, что приведенные выше правила гарантируют, что имеется определенное первое значение в каждом упорядочении и определенное следующее значение. Не требуется, чтобы было определенное последнее значение (каждый набор или оба могут быть бесконечными).

19.5.11 Если число абстрактных значений в упорядочении пункта назначения меньше, чем число абстрактных значений в упорядочении источника, то это не будет ошибкой. Однако спецификация ECN будет неспособна кодировать некоторые абстрактные значения из спецификации ACH.1; это должно быть указано комментарием в спецификации ACH.1 или в спецификации ECN.

19.5.12 Если число абстрактных значений в упорядочении пункта назначения превышает такое число в упорядочении источника, то могут быть некоторые определенные в ECN кодирования, которые не имеют абстрактного значения ACH.1 и которые не будут генерироваться.

19.5.13 Это отображение может применяться также во всех случаях, когда единственное абстрактное значение в структуре цели является тем, которое связано с одиночным экземпляром того же класса, что и структура источника.

Примечание — Этот случай мог бы появиться, если бы структура цели была такой же, что и структура источника, перед которой имеются один или несколько экземпляров классов в категории «тег».

19.5.14 Классы в категории «тег» могут присутствовать в структуре цели, но запрашиваются они для того, чтобы иметь связанный номер тега, указанный в определении этой структуры.

19.6 Отображение с помощью распределения значений

19.6.1 Это отображение берет диапазоны значений из класса кодирования в «целочисленной» категории и отображает каждый диапазон в другое поле целых чисел в более сложной структуре кодирования. Поля, не получившие абстрактных значений, должны иметь свои значения, указанные путем применения определителей.

19.6.2 Все имена структур кодирования разыменуются (рекурсивно) перед применением этого отображения.

19.6.3 Класс кодирования источника должен быть затем классом в «целочисленной» категории, возможно, с предшествующим классом в категории «тег», который отображается согласно 19.1.3—19.1.5.

19.6.4 Класс кодирования цели может быть любой структурой кодирования и может содержать классы в категории «тег», но все имена полей в полной структуре кодирования должны быть разными, а все классы в категории «тег» в цели (за исключением тех, которые отображаются по 19.6.3) должны иметь номер тега в своем определении и будут в его отсутствие игнорироваться при отображении.

19.6.5 Значения должны отображаться только в поля в структуре цели, которая является классами в «целочисленной» категории, перед которыми, возможно, имеются классы в категории «тег» (см. 19.6.4) и, возможно, с границами.

19.6.6 Продукцией «MappingByValueDistribution» является:

```
MappingByValueDistribution ::=
        DISTRIBUTION
        "{"
        Distribution "," +
        "}"
Distribution ::=
        SelectedValues
```

```
            TO
            identifier
SelectedValues ::=
            SelectedValue
          | DistributionRange
          | REMAINDER
DistributionRange ::=
            DistributionRangeValue1
            ".."
            DistributionRangeValue2
SelectedValue ::= SignedNumber
DistributionRangeValue1 ::= SignedNumber
DistributionRangeValue2 ::= SignedNumber
```

19.6.7 «SignedNumber» определен в ИСО/МЭК 8824-1, подраздел 19.1.

19.6.8 «DistributionRangeValue1» должен быть меньше, чем «DistributionRangeValue2».

19.6.9 Значение, указанное «SelectedValue» в «SelectedValues», или набор значений, больших или равных «DistributionRangeValue1» и меньших или равных «DistributionRangeValue2», отображается в поле, указанное «identifier».

19.6.10 Зарезервированное слово **REMAINDER** должно использоваться только один раз для последнего «SelectedValues»; оно определяет все абстрактные значения в классе кодирования источника, который не был распределен более ранним «SelectedValues».

19.6.11 Значение не должно отображаться в более чем одно поле цели, но несколько «SelectedValues» могут иметь один и тот же пункт назначения.

19.6.12 Если в поле цели имеются границы, то применяется 17.4.7.

19.6.13 Если значение из источника отображается в поле цели, чье присутствие зависит от факультативной возможности или выбора альтернативы, или от того и другого, то это не будет ошибкой, но факультативная возможность и выбор альтернативы в цели (когда кодируются такие значения) должны быть такими, чтобы кодирование цели охватывало это поле цели.

19.7 Отображение целочисленных значений в биты

19.7.1 Это отображение берет одиночные значения или диапазоны значений из класса кодирования в «целочисленной» категории (которому, возможно, предшествуют классы категории «тег», указанные в 19.1.3—19.1.5) и отображает каждое целочисленное значение в значение цепочки битов (которому, возможно, предшествуют классы категории «тег»).

Примечание — Это отображение предназначено для поддержки саморазграничивающих кодирований целых чисел, таких как кодовые последовательности Хаффмана (см. приложение Е с дальнейшим обсуждением и примерами кодовых последовательностей Хаффмана).

19.7.2 Класс кодирования источника должен быть классом в «целочисленной» категории, которому, возможно, предшествуют классы в категории «тег».

19.7.3 Класс кодирования пункта назначения должен быть классом в категории «цепочка битов», которому, возможно, предшествуют классы в категории «тег».

19.7.4 Классы в категории «тег» отображаются согласно 19.1.3—19.1.5.

19.7.5 Продукцией «MappingIntToBits» является:

```
MappingIntToBits ::=
      TO BITS
      "{"
      MappedIntToBits "," +
      "}"
MappedIntToBits ::=
      SingleIntValMap |
      IntValRangeMap
```

19.7.6 Каждый «SingleIntValMap» отображает одиночное целочисленное значение в одиночное значение цепочки битов.

19.7.7 Каждый «IntValRangeMap» отображает диапазон непрерывных и возрастающих целочисленных значений в диапазон непрерывных и возрастающих значений цепочек битов.

19.7.8 Значения цепочек битов определяются как непрерывные:

a) если все они имеют одинаковую длину в битах;

b) когда считаются положительным целочисленным значением, то соответствующие целочисленные значения являются непрерывными и возрастающими целочисленными значениями.

19.7.9 Только значения, указанные в отображении, пригодны для кодирования. Другие абстрактные значения источника не отображаются и не могут быть кодированы объектом кодирования, определенным при присвоении объектов кодирования с использованием этой конструкции. ECN или применение будет ошибочным, если такие значения подаются к кодеру.

Примечание — Это лимитирование кодирования должно отражаться ограничениями на тип ACH.1, к которому оно применяется, либо комментарием в спецификации ACH.1.

19.7.10 Продукцией «SingleIntValMap» является:

```
SingleIntValMap ::=
        IntValue
        TO
        BitValue
IntValue ::= SignedNumber
BitValue ::=
        bstring |
        hstring
```

19.7.11 «SignedNumber», «bstring» и «hstring» определены в ИСО/МЭК 8824-1, подразделы 19.1, 12.10 и 12.12 соответственно.

19.7.12 «SingleIntValMap» отображает описанное целочисленное значение в описанное значение цепочки битов.

19.7.13 Продукцией «IntValRangeMap» является:

```
IntValRangeMap ::=
    IntRange
    TO
    BitRange
IntRange ::=
    IntRangeValue1
    ".."
    IntRangeValue2
BitRange ::=
BitRangeValue1
    ".."
    BitRangeValue2
IntRangeValue1 ::= SignedNumber
IntRangeValue2 ::= SignedNumber
BitRangeValue1 ::=
    bstring |
    hstring
BitRangeValue2 ::=
    bstring |
    hstring
```

19.7.14 Цепочки битов «BitRangeValue1» и «BitRangeValue2» должны иметь одно и то же число битов.

19.7.15 Значение «IntRangeValue2» должно превышать значение «IntRangeValue1».

19.7.16 Когда считается кодированием положительное целое число (см. ИСО/МЭК 8825-1, пункт 8.3.3), «BitRangeValue2» должен представлять целочисленное значение (обозначим его «B»), превышающее значение, представленное «BitRangeValue1» (обозначим его «A»), а разность между

этими целочисленными значениями, соответствующими «BitRangeValue2» и «BitRangeValue1» («B» — «A»), должна равняться разности между значениями «IntRangeValue2» и «IntRangeValue1».

19.7.17 «BitRange» представляет упорядоченный набор цепочек битов, соответствующих целочисленным значениям между «A» и «B».

19.7.18 «IntValRangeMap» отображает каждое целое число определенного диапазона в соответствующее значение цепочки битов «BitRange» (в приложении E приведены примеры «IntValRangeMap»).

19.7.19 Спецификация ECN будет ошибочной, если какой-либо «BitRange» содержит значение, которое нарушает ограничение на размер в цели.

20 Определение объектов кодирования с использованием определенного синтаксиса

20.1 В разделах 21—25 описывается информация, необходимая для определения объектов кодирования для каждой категории классов кодирования, и синтаксис, который следует использовать. Этот синтаксис называется определенным синтаксисом и описывается с помощью нотации класса информационных объектов из ИСО/МЭК 8824-2 с учетом изменений из приложения B.

20.2 Определенный синтаксис для каждой категории может использоваться также для определения объектов кодирования структур, которые являются классами такой категории, которой предшествует один или несколько экземпляров класса в категории «тег». Когда в нижеследующих текстах требуется, чтобы класс был в какой-либо указанной категории, тогда это охватывает и случай, когда этому классу предшествует класс в категории «тег».

20.3 Модифицированная нотация класса информационного объекта используется только в настоящем стандарте.

20.4 Использование нотации определенного синтаксиса для описания объектов кодирования рассмотрено в 17.2. Определенным синтаксисом для описания объектов кодирования будет синтаксис, указанный командами **WITH SYNTAX** в разделах 23—25.

20.5 Команды **WITH SYNTAX** налагают ограничения на значения некоторых признаков кодирования в увязке со значениями других признаков кодирования, чтобы навязать некоторые (но не все) ограничения на семантику. Другие ограничения на использование оператора **WITH SYNTAX** указываются в тексте.

20.6 Определенный синтаксис для каждого класса кодирования указывает число признаков кодирования, которое может подаваться со значениями типов ACH.1, определенных в разделе 21 (или в некоторых случаях с другими классами кодирования и объектами кодирования), для того чтобы подать информацию, необходимую для описания объекта кодирования этого класса. Информацией, необходимой для определения объекта кодирования, обычно является комбинация значений признаков кодирования вместе с конкретным экземпляром определенного синтаксиса, используемого для описания таких значений.

Примечание — Это отличается от использования оператора **WITH SYNTAX** в нормальном определении информационного объекта, где семантика, связанная с информационным объектом, зависит только от набора значений для полей класса информационного объекта, а не от формы оператора **WITH SYNTAX**, используемой для установки таких значений (см. B.15).

20.7 Признаки кодирования, описанные в разделах 23—25, работают вместе в группах признаков кодирования и используют значения типов ACH.1 для своего описания. В разделе 21 описывается смысл значений типов, широко используемых в спецификациях этих признаков кодирования.

20.8 Некоторые полные тексты разделов 21 и 22 копируются в разделах 22—25. Когда это производится, скопированный текст печатается серым цветом, а также приводится ссылка на полный текст.

20.9 В разделе 25 определяется число преобразователей, которые могут быть применены к абстрактным значениям. Некоторые группы признаков кодирования требуют упорядоченного списка преобразователей, которые будут применяться кодером. Чтобы декодирование было возможно, преобразователи, примененные кодером, должны быть обратимыми для декодера при восстановлении исходных абстрактных значений. В разделах 23 и 24 указывается, когда преобразователи должны быть обратимыми, а в разделе 25 указываются абстрактные значения, для которых любой заданный преобразователь является обратимым.

21 Типы, используемые в спецификации определенного синтаксиса

Примечание — Предполагается, что все приводимые здесь определения типов АСН.1 имеют автоматические теги и не имеют растяжимости.

21.1 Тип Unit

21.1.1 Продукцией типа «**Unit**» является:

```
Unit ::= INTEGER
    {repetitions (0), bit (1), nibble (4), octet (8), word16 (16),
    dword32 (32)} (0..256)
```

21.1.2 Безусловным значением (по умолчанию) для этого типа всегда является **bit**.

21.1.3 Признак кодирования этого типа указывает единицу, в которой будут подсчитываться другие признаки кодирования или поля определителя.

21.1.4 Значение признака кодирования этого типа ограничено во всех случаях, кроме одного, до ненулевых значений. В этих случаях признак кодирования указывает число битов. Это число битов определяет единицу, в которой подсчитываются другие признаки кодирования или поля определителя.

21.1.5 При использовании в определении объекта кодирования класса в категории «повторение» разрешается также значение **repetitions**, указывающее, что связанный подсчет дает число повторений в кодировании.

21.2 Тип EncodingSpaceSize

21.2.1 Продукцией типа «**EncodingSpaceSize**» является:

```
EncodingSpaceSize ::= INTEGER
        { encoder-option-with-determinant (–3),
        variable-with-determinant (–2),
        self-delimiting-values (–1),
        fixed-to-max (0)} (–3..MAX)
```

21.2.2 Безусловным значением (по умолчанию) для этого типа всегда является **self-delimiting-values**.

21.2.3 Признак кодирования этого типа указывает размер пространства кодирования (см. 9.21.5).

21.2.4 Положительные (не нулевые) значения указывают фиксированный размер пространства кодирования в виде значения типа «**Unit**», умноженного на значение типа «**EncodingSpaceSize**» в битах. Если значением типа «**Unit**» является «**repetitions**», то размер пространства кодирования может изменяться (так как пространство кодирования, необходимое для каждого компонента, может быть разным), но оно всегда составляет фиксированное число повторений, а если должно кодироваться абстрактное значение, которое не имеет такого числа повторений, то спецификация или применение ECN будут ошибочными.

21.2.5 Значение «**encoder-option-with-determinant**» указывает, что размер пространства кодирования может меняться согласно кодируемому абстрактному значению и что кодер должен выбирать размер пространства кодирования, записывая выбранный размер в связанный определитель. В этом случае требуется значение типа «**EncodingSpaceDetermination**» (см. 21.3) или «**RepetitionSpaceDetermination**» (см. 21.7).

Примечание — Значение типа «**EncodingSpaceDetermination**» или «**RepetitionSpaceDetermination**» (для определения размера пространства кодирования) требуется в этом случае (и в случае из 21.2.6), но обеспечение определителя разрешается во всех других случаях для поддержки кодирований (аналогичных BER), которые используют определители длины даже тогда, когда они избыточны. Любое различие между двумя определениями будет ошибкой. Не всегда, однако, может быть возможность определения, является ли она ошибкой спецификации ECN или ошибкой применения, но кодеры, удовлетворяющие техническим требованиям, не должны передавать таких кодирований.

21.2.6 Значение «**variable-with-determinant**» указывает, что размер пространства кодирования может меняться согласно кодируемому абстрактному значению. В этом случае требуется значение

типа «**EncodingSpaceDetermination**» (см. 21.3) или «**RepetitionSpaceDetermination**» (см. 21.7) (для того чтобы обеспечить точные средства определения размера пространства кодирования).

21.2.7 Значение «**self-delimiting-values**» указывает, что кодирование значения является саморазграничивающим, то есть каждое значение кодируется в несколько указанных значений типа «**Unit**». Это не должно быть парой абстрактных значений, в которой кодирование одного абстрактного значения является первой частью кодирования другого абстрактного значения.

Примечание — Кодер может (после возможного определения неиспользуемых битов и выравнивания) обнаруживать конец пространства кодирования путем сопоставления кодирования каждого возможного абстрактного значения с кодированием, которое рассматривается. В кодированиях, которые выданы кодером, соответствующим техническим требованиям, будет точно одно совпадение. В декодерах могут проявляться более эффективные, но эквивалентные подходы.

21.2.8 Значение «**fixed-to-max**» указывает, что пространство кодирования должно быть одинаковым для кодирования всех абстрактных значений. Оно указывает, что размер пространства кодирования должен быть самым малым произведением с множителем «**Unit**», которое может содержать указанное кодирование любого одного (всех) абстрактного(ых) значения(й). Это значение не должно использоваться, если абстрактное значение, которое должно быть кодировано в пространство кодирования, является абстрактным значением, связанным с классом в категории «конкатенация» (см. 23.14.2.5) или «повторение» (см. 23.13.2.5).

Примечания

1 Специальным случаем является наличие одиночного абстрактного значения, у которого кодированием значения является нуль битов. Это вызовет пустое пространство кодирования (нуль битов).

2 Спецификация ECN будет ошибочной, если она применяется в случае, когда максимальный размер не может быть определен (например, при кодировании неограниченных целых чисел), но кодеры, соответствующие техническим требованиям, должны отказываться генерировать кодирования в таких случаях.

21.3 Тип EncodingSpaceDetermination

21.3.1 Продукцией типа «**EncodingSpaceDetermination**» является:

```
EncodingSpaceDetermination ::= ENUMERATED
        {field-to-be-set, field-to-be-used, container}
```

21.3.2 Безусловным значением (по умолчанию) для этого типа всегда является «**field-to-be-set**».

21.3.3 Признак кодирования этого типа указывает способ определения пространства кодирования, когда признак кодирования типа «**EncodingSpaceSize**» (см. 21.2) установлен в «**variable-with-determinant**» или «**encoderoption-with-determinant**».

21.3.4 Значение «**field-to-be-set**» требует спецификацию **REFERENCE** для поля, которое будет устанавливаться кодером для переноса информации о длине и будет использоваться декодером. Эта спецификация кодирования указывает, как кодер должен устанавливать значение этого поля в пределах размера пространства кодирования (в единицах пространства кодирования). Если поле устанавливается более одного раза с помощью «**field-to-be-set**» или «**flag-to-be-set**» (см. 21.7), то спецификация ECN или применение будут ошибочными, когда разные значения выдаются разными процедурами кодирования; в этом случае кодеры не должны генерировать кодирования.

21.3.5 Значение «**field-to-be-used**» требует спецификацию **REFERENCE** для поля, значение которого может быть установлено из абстрактного синтаксиса (то есть соответствующее поле появляется в пределах спецификации ACH.1) либо может быть установлено некоторыми другими действиями кодера, вызванными с помощью «**field-tobe-set**» или «**flag-to-be-set**». Эта спецификация кодирования указывает, как декодер должен извлекать размер пространства кодирования из значения этого поля. Кодер, соответствующий техническим требованиям, не должен вырабатывать кодирования, при которых преобразования этого поля в декодере не будут указывать правильно конец пространства кодирования.

21.3.6 Значение «container» требует либо спецификацию **REFERENCE** для другого поля, у которого класс кодирования (контейнер) имеет определитель длины, а содержимое включает в себя это пространство кодирования, либо спецификацию того, что конец PDU определяет конец пространства кодирования (с помощью **OUTER**). Пространство кодирования заканчивается, когда заканчивается указанный контейнер или когда подсчитан конец PDU. Эта спецификация может применяться только в

случаях, когда пространство кодирования кодируемого элемента является последним кодированием, подлежащим размещению в контейнере.

Примечание — Если в контейнер помещаются дополнительные кодирования, то это будет ошибкой кодера ECN (возможно, вызванной ошибкой в спецификации ECN или в применении).

21.4 Тип UnusedBitsDetermination

21.4.1 Продукцией типа «**UnusedBitsDetermination**» является:

```
UnusedBitsDetermination ::= ENUMERATED
        {field-to-be-set, field-to-be-used, not-needed}
```

21.4.2 Безусловным значением (по умолчанию) для этого типа всегда является «**field-to-be-set**».

21.4.3 Признак кодирования этого типа указывает способ, которым декодер может определить неиспользуемые биты, когда кодирование значения выровнено влево или вправо в пространстве кодирования.

21.4.4 Значение «**field-to-be-set**» требует спецификацию **REFERENCE** для поля, которое будет устанавливаться кодером для переноса информации о неиспользуемых битах и использоваться декодером. Эта спецификация кодирования указывает, как кодер должен определять число неиспользуемых битов и как устанавливать значение этого поля с этим числом неиспользуемых битов. Если поле устанавливается более одного раза с помощью «**field-to-beset**» или «**flag-to-be-set**» (см. 21.7), то спецификация ECN или применение будут ошибочными, когда разные значения выдаются разными процедурами кодирования; в этом случае кодеры не должны генерировать кодирования.

21.4.5 Значение «**field-to-be-used**» требует спецификацию **REFERENCE** для поля, значение которого может быть установлено из абстрактного синтаксиса (то есть соответствующее поле появляется в пределах спецификации ACH.1) либо может быть установлено некоторыми другими действиями кодера, вызванными с помощью «**field-tobe-set**» или «**flag-to-be-set**». Эта спецификация кодирования указывает, как кодер должен определять число неиспользуемых битов из значения этого поля. Кодер, соответствующий техническим требованиям, не должен вырабатывать кодирования, при которых преобразования этого поля в декодере не будут правильно указывать число неиспользуемых битов.

21.4.6 Значение «**not-needed**» указывает, что декодер не требует явного определителя, чтобы обнаруживать число неиспользуемых битов. Число неиспользуемых битов будет возможно выводить из спецификации кодирования без знания реального абстрактного значения, которое кодировано. Этот способ определения описывается для каждого кодирования значения.

21.5 Тип OptionalityDetermination

21.5.1 Продукцией типа «**OptionalityDetermination**» является:

```
OptionalityDetermination ::= ENUMERATED
        {field-to-be-set, field-to-be-used, container, handle, pointer}
```

21.5.2 Безусловным значением (по умолчанию) для этого типа всегда является «**field-to-be-set**».

21.5.3 Признак кодирования этого типа указывает способ определения наличия или отсутствия факультативного компонента.

21.5.4 Значение «**field-to-be-set**» требует спецификацию **REFERENCE** для поля, которое будет устанавливаться кодером для переноса информации о факультативных возможностях и будет использоваться декодером. Спецификация ECN будет также содержать признак кодирования, который указывает, как кодер должен устанавливать значение этого поля со смысловым булевым значением, которое будет TRUE, когда факультативный компонент присутствует, или FALSE, когда факультативный компонент отсутствует. Если поле устанавливается более одного раза с помощью «**field-to-be-set**» или «**flag-to-be-set**» (см. 21.7), то спецификация ECN или применение будут ошибочными, когда разные значения выдаются разными процедурами кодирования; в этом случае кодеры не должны генерировать кодирования.

21.5.5 Значение «**field-to-be-used**» требует спецификацию **REFERENCE** для поля, значение которого может быть установлено из абстрактного синтаксиса (то есть соответствующее поле появляется в пределах спецификации ACH.1) либо может быть установлено некоторыми другими действиями

кодера, вызванными с помощью «**field-tobe-set**» или «**flag-to-be-set**». Эта спецификация будет также содержать признак кодирования, который указывает, как декодер должен определять наличие или отсутствие факультативного компонента по значению этого поля. Кодер, соответствующий техническим требованиям, должен гарантировать, что значение этого поля правильно указывает наличие или отсутствие факультативного поля.

21.5.6 Значение «**container**» требует либо спецификацию **REFERENCE** для другого поля, у которого класс кодирования (контейнер) имеет определитель длины, а содержимое включает в себя этот факультативный компонент, либо спецификацию того, что контейнер является концом PDU (с помощью **OUTER**). Если конец контейнера появился тогда, когда декодер ищет начало этого факультативного компонента, то декодер будет считать, что этот факультативный компонент отсутствует.

Примечание — Эта спецификация может использоваться только в случае, когда кодируемые абстрактные значения таковы, что никакие другие кодирования не должны помещаться в контейнер. Это может потребовать ограничений, налагаемых на абстрактные значения типа ACH.1, например, запрета на введение позднего факультативного компонента, пока не имеются все ранние факультативные компоненты. Спецификация ECN или применение будут ошибочными, когда в контейнер должны быть помещены дополнительные кодирования после компонента, факультативные возможности которого определены этим способом; кодер, соответствующий требованиям, не должен генерировать такие кодирования.

21.5.7 Значение «**handle**» требует, чтобы был указан идентификационный описатель. Этот идентификационный описатель должен выдаваться и объектом кодирования для факультативного компонента, и объектом кодирования, применяемым к любому возможному альтернативному классу кодирования, который может следовать, если факультативный компонент отсутствует. Любой возможный альтернативный класс кодирования может быть компонентом конкатенации, содержащей факультативный компонент, или может быть классом кодирования, следующим за конкатенацией. Наборы значений описателей, указанные всеми участвующими объектами кодирования (показывающими один и тот же идентификационный описатель), должны быть непересекающимися.

Примечание — Каждое абстрактное значение данного компонента требуется для того, чтобы значение описателя соответствовало заданному набору значений описателя (см. 22.9.2.2).

21.5.8 Если конец любого открытого контейнера (или конец PDU) был обнаружен, когда декодер пытается обнаружить присутствие или отсутствие факультативного компонента, то декодер должен определить, что факультативный компонент отсутствует. В противном случае декодер должен определить, что компонент присутствует, если, и только если, декодирование оставшихся частей кодирования даст значение указанного идентификационного описателя, которое соответствует набору значений описателя факультативного компонента. Спецификация ECN будет ошибочной, если это будет не результатом правильной идентификации наличия или отсутствия кодирования факультативного компонента; кодеры, соответствующие техническим требованиям, не должны генерировать такие кодирования.

21.5.9 Значение «**pointer**» требует для другого поля спецификацию **REFERENCE** о начале кодирования. Если данное поле является нулем, то этот компонент отсутствует. Если оно не является нулем, то применяются правила для указателя начала кодирования (см. 22.3).

21.6 Тип AlternativeDetermination

21.6.1 Продукцией типа «**AlternativeDetermination**» является:

```
AlternativeDetermination ::=
        ENUMERATED {field-to-be-set, field-to-be-used, handle}
```

21.6.2 Безусловным значением (по умолчанию) для этого типа всегда является «**field-to-be-set**».

21.6.3 Признак кодирования этого типа указывает способ, которым декодер определяет, какая альтернатива присутствует в кодировании класса категории «альтернативы».

21.6.4 Значение «**field-to-be-set**» требует спецификацию **REFERENCE** для поля, которое будет устанавливаться кодером для переноса информации, указывающей альтернативу, и использоваться декодером. Эта спецификация будет также содержать признак кодирования, который указывает, как кодер должен устанавливать значение этого поля со смысловым целочисленным значением, которое указывает каждую альтернативу (используя порядок следования, указанный в других признаках кодирования). Если поле устанавливается более одного раза с помощью «**field-to-beset**» или «**flag-to-be-set**»

(см. 21.7), то спецификация ECN или применение будут ошибочными, когда разные значения выдаются разными процедурами кодирования; в этом случае кодеры не должны генерировать кодирования.

21.6.5 Значение «**field-to-be-used**» требует спецификацию **REFERENCE** для поля, значение которого может быть установлено из абстрактного синтаксиса (то есть соответствующее поле появляется в пределах спецификации ACH.1) либо некоторыми другими действиями кодера, вызванными с помощью «**field-tobe-set**» или «**flag-to-be-set**». Эта спецификация будет также содержать признак кодирования, который указывает, как декодер должен определять (из значения упомянутого поля) смысловое целочисленное значение, которое указывает альтернативу (используя порядок следования, указанный в других признаках кодирования).

21.6.6 Значение «**handle**» требует, чтобы был указан идентификационный описатель. Этот идентификационный описатель должен выдаваться объектами кодирования всем альтернативами в конструкции, определенной классом в категории «альтернативы». Наборы значений описателей, заданные этими объектами кодирования, должны быть непересекающимися (нарушение этого правила даст ошибку спецификации ECN, и кодеры, соответствующие техническим требованиям, не должны генерировать кодирований, когда это правило нарушается).

21.6.7 Декодер должен определять альтернативу, которая присутствует за счет декодирования оставшихся частей кодирования, чтобы произвести значение для указанного идентификационного описателя. Альтернатива, набор значений описателя которой совпадает с этим значением, является той альтернативой, которая присутствует. Если конец любого открытого контейнера (или конец PDU) достигается до того, как идентификационный описатель может быть декодирован, или если значение идентификационного описателя не соответствует набору значений описателя любой из альтернатив, то это является ошибкой кодирования.

Примечание — Каждое абстрактное значение данной альтернативы должно иметь значение описателя, соответствующее набору значений описателя альтернативы (см. 22.9.2.2).

21.7 Тип RepetitionSpaceDetermination

21.7.1 Продукцией типа «**RepetitionSpaceDetermination**» является:

RepetitionSpaceDetermination ::= ENUMERATED
{field-to-be-set, field-to-be-used, flag-to-be-set, flag-to-be-used,
container, pattern, handle, not-needed}

21.7.2 Безусловным значением (по умолчанию) для этого типа всегда является «**field-to-be-set**».

21.7.3 Признак кодирования этого типа указывает способ, которым декодер определяет конец пространства кодирования в кодировании класса категории «повторение». Оно заменяет использование признака кодирования типа «**EncodingSpaceDetermination**» при кодировании повторений.

21.7.4 Значение «**field-to-be-set**» требует спецификацию **REFERENCE** для поля, которое будет устанавливаться кодером для переноса информации, указывающей размер пространства повторения. Эта спецификация кодирования определяет, как кодер должен устанавливать значение этого поля в пределах размера (в единицах пространства повторения) поля повторения. Если поле устанавливается более одного раза с помощью «**field-to-be-set**» или «**flag-to-be-set**», то спецификация ECN или приложение будут ошибочными, когда разные значения выдаются разными процедурами кодирования; в этом случае кодеры не должны генерировать кодирования.

21.7.5 Значение «**field-to-be-used**» требует спецификацию **REFERENCE** для поля, значение которого может быть установлено из абстрактного синтаксиса (то есть соответствующее поле появляется в пределах спецификации ACH.1) либо может быть установлено некоторыми другими действиями кодера, вызванными с помощью «**field-tobe-set**» или «**flag-to-be-set**». Эта спецификация кодирования определяет, как декодер должен получать размер (в единицах пространства повторения) пространства кодирования из значения этого поля. Кодер, соответствующий техническим требованиям, не должен выдавать кодирования, при которых преобразования этого поля в декодере не указывают правильно конец пространства кодирования.

21.7.6 Значение «**flag-to-be-set**» требует спецификацию **REFERENCE** для поля, которое является частью повторяющегося элемента и которое будет устанавливаться кодером для указания последнего элемента в повторении. Эта спецификация кодирования определяет, как кодер должен устанавливать значение этого поля с булевым значением, которое будет FALSE, когда элемент является последним

в повторении, или TRUE в противном случае. Если поле устанавливается более одного раза с помощью «**flag-to-be-set**» или «**field-to-be-set**», то спецификация ECN или применение будут ошибочными, когда разные значения выдаются разными процедурами кодирования; в этом случае кодеры не должны генерировать кодирования.

21.7.7 Значение «**flag-to-be-used**» требует спецификацию **REFERENCE** для поля, которое является частью повторяющегося элемента и чье значение может быть установлено из абстрактного синтаксиса (то есть соответствующее поле появляется в пределах спецификации ACH.1) либо может быть установлено некоторыми другими действиями кодера, вызванными с помощью «**flag-to-be-set**» или «**field-to-be-set**». Эта спецификация кодирования определяет, как декодер должен получать булево значение из значения этого поля. Булево значение будет FALSE, когда элемент является последним элементом в повторении, или TRUE в противном случае.

Кодер, соответствующий техническим требованиям, не должен выдавать кодирования, при которых преобразования этого поля в декодере не указывают правильно последний элемент повторения.

21.7.8 Значение «**container**» требует либо спецификацию **REFERENCE** для другого поля, класс кодирования (контейнер) которого имеет определитель длины, а содержимое которого включает в себя класс кодирования в категории «повторение», либо спецификацию (с помощью **OUTER**) того, что конец PDU определяет конец повторений.

Повторения кончаются, когда указанный контейнер заканчивается либо когда после завершения кодирования одного повторения подсчитан конец PDU.

Примечание — Эта спецификация может использоваться только в случаях, когда кодирование класса (категории «повторение») является последним кодированием, помещаемым в контейнер. Спецификация ECN будет ошибочной, если в контейнер помещаются дополнительные кодирования; кодеры, соответствующие техническим требованиям, не должны генерировать такие кодирования.

21.7.9 Значение «**pattern**» указывает, что некоторая определенная комбинация битов (см. 21.10) будет заканчивать повторения. В этом случае дополнительные признаки кодирования будут требовать введения кодером определенной комбинации и обнаружения этой комбинации декодером. Спецификация ECN будет ошибочной, если кодирование комбинации может стать начальной частью кодирования абстрактного значения в повторении. Кодер, соответствующий техническим требованиям, должен обнаруживать такие ошибки и не должен генерировать кодирования, нарушающие это правило.

Примечание — Примером является цепочка знаков с нулем в конце, содержимое которой не допускает применения знака нуль.

21.7.10 Значение «**handle**» требует, чтобы был определен идентификационный описатель. Идентификационный описатель должен выставляться элементом, который повторяется, и всеми возможными последующими элементами (с учетом факультативных возможностей). Значение идентификационного описателя для элемента, который повторяется, должно отличаться от значения для всех возможных последующих элементов.

Примечание — Каждое абстрактное значение данного компонента должно иметь значение описателя, соответствующее набору значений описателя компонента (см. 22.9.2.2).

21.7.11 Значение «**not-needed**» указывает, что число повторений фиксировано в абстрактном синтаксисе.

Примечание — Спецификация ECN будет иметь ошибку (которая должна обнаруживаться и блокироваться кодерами), когда это кодирование указано, а число повторений не ограничено таким образом, либо когда применение нарушает это ограничение.

21.8 Тип Justification

21.8.1 Продукцией типа «**Justification**» является:

```
Justification ::= CHOICE
                  { left INTEGER (0..MAX),
                  right INTEGER (0..MAX)}
```

21.8.2 Безусловным значением (по умолчанию) для этого типа всегда является «**right:0**».

21.8.3 Признак кодирования этого типа указывает на правое или левое выравнивание кодирования значения в пределах пространства кодирования, используя смещение битов от конца пространства кодирования.

21.8.4 Альтернатива «**left**» указывает, что начальный бит кодирования значения располагается относительно начального края пространства кодирования. Целочисленное значение указывает число битов между начальным краем пространства кодирования и начальным битом кодирования значения.

Примечание — Если кодирование значения имеет нефиксированную длину или является саморазграничивающим, то использование заполнения значения, в контейнере с фиксированным размером может в некоторых обстоятельствах сделать невозможным для декодера восстановление исходных абстрактных значений. Это будет ошибкой спецификации ECN.

21.8.5 Альтернатива «**right**» указывает, что заканчивающий бит кодирования значения располагается относительно заканчивающего края пространства кодирования. Целочисленное значение указывает число битов между заканчивающим битом кодирования значения и заканчивающим краем пространства кодирования.

21.8.6 Установка битов (если она имеется) перед или после кодирования значения определяется признаками кодирования типа «**Padding**» и «**Pattern**» (см. 21.9 и 21.10).

21.9 Тип Padding

21.9.1 Продукцией типа «**Padding**» является:

```
Padding ::= ENUMERATED {zero, one, pattern, encoder-option}
```

21.9.2 Безусловным значением (по умолчанию) для признака кодирования этого типа всегда является «**zero**».

21.9.3 Признак кодирования этого типа определяет детали заполнения при предварительном заполнении для классов в категории «pad» [заполнение] и при последующем заполнении для PDU, указанного в классе кодирования **#OUTER**.

21.9.4 Если значением является «**zero**», то заполнение содержит нуль битов.

21.9.5 Если значением является «**one**», то заполнение содержит один бит.

21.9.6 Если значением является «**pattern**», то биты устанавливаются согласно свойству кодирования типа «**Pattern**» (см. 21.10).

21.9.7 Если значением является «**encoder-option**», то кодер свободно выбирает значения битов.

21.10 Типы Pattern и Non-Null-Pattern

21.10.1 Продукцией типа «**Pattern**» является:

```
Pattern ::= CHOICE
        {bits BIT STRING,
        octets OCTET STRING,
        char8 IA5String,
        char16 BMPString,
        char32 UniversalString,
        any-of-length INTEGER (1..MAX),
        different ENUMERATED {any} }
```

21.10.2 Продукцией типа «**Non-Null-Pattern**» является:

```
Non-Null-Pattern ::= Pattern
        (ALL EXCEPT (bits:"B | octets:"H | char8:"" | char16:"" |
                                char32:""))
```

21.10.3 Безусловным значением (по умолчанию) для признака кодирования этого типа всегда является «**bits:'0'B**».

21.10.4 Альтернатива «**bits**» или «**octets**» определяет комбинацию длины и значение, равное заданной цепочке битов или цепочке октетов соответственно.

21.10.5 Альтернатива «**char8**» определяет комбинацию (кратную 8 битам), в которой каждый знак из заданной цепочки преобразован в его значение по ИСО/МЭК 10646 в виде 8-битового значения.

21.10.6 Альтернатива «**char16**» определяет комбинацию (кратную 16 битам), в которой каждый знак из заданной цепочки преобразован в его значение по ИСО/МЭК 10646 в виде 16-битового значения.

21.10.7 Альтернатива «**char32**» определяет комбинацию (кратную 32 битам), в которой каждый знак из заданной цепочки преобразован в его значение по ИСО/МЭК 10646 в виде 32-битового значения.

21.10.8 Альтернатива «**any-of-length**» определяет размер комбинации. Реальное значение комбинации выбирается кодером.

21.10.9 Значение «**different:any**» разрешается только в случаях, когда в той же группе признаков кодирования имеется другой признак кодирования типа «**Pattern**». В этом случае любой из двух признаков кодирования (но не оба) типа «**Pattern**» может быть установлен в «**different:any**». Значение «**different:any**» указывает, что длина комбинации должна быть такой же, как длина комбинации, установленная для другого признака кодирования. Оно указывает также, что его значение выбирается кодером при условии, что это значение отличается от значения комбинации, указанного для другого признака кодирования.

21.10.10 «**Non-Null-Pattern**» используется для предварительного заполнения или для выравнивания (но не для обеих целей), причем эта комбинация укорачивается и/или копируется, как требуется, чтобы обеспечить достаточное число битов для предварительного заполнения, предварительного заполнения значения и последующего заполнения значения.

21.10.11 Значение «**different:any**» типа «**Pattern**» исключено из большинства применений этого типа. Когда параметр типа «**Pattern**» использован с целью указания комбинации для булева значения (например, **TRUE**), значение «**different:any**» может использоваться с целью указания комбинации для другого булева значения (**FALSE** в этом случае). При использовании этого метода «**different:any**» означает выбор кодера для комбинации. Кодер может использовать любую комбинацию, которую он выбирает, но она должна иметь ту же длину, что и другая комбинация, и должна отличаться от нее по крайней мере на одну битовую позицию.

21.11 Тип RangeCondition

21.11.1 Продукцией типа «**RangeCondition**» является:

```
RangeCondition ::= ENUMERATED
        { unbounded-or-no-lower-bound,
        semi-bounded-with-negatives,
        bounded-with-negatives,
        semi-bounded-without-negatives,
        bounded-without-negatives
        test-lower-bound,
        test-upper-bound,
        test-range}
```

21.11.2 Безусловным значением (по умолчанию) для признака кодирования этого типа всегда является «**unboundedor-no-lower-bound**».

21.11.3 Признак кодирования типа «**RangeCondition**» используется в спецификации предиката (логического условия), который проверяет наличие и природу границ целочисленных значений, связанных с классом кодирования в «целочисленной» категории.

21.11.4 Предикат удовлетворяет каждому из первых пяти перечисленных значений из 21.11.1, если, и только если, следующие условия удовлетворяются границами класса кодирования в «целочисленной» категории:

a) **unbounded-or-no-lower-bound**: либо нет границ, либо имеется только верхняя граница, но нет нижней границы;

b) **semi-bounded-with-negatives**: имеется нижняя граница, которая меньше нуля, но нет верхней границы;

c) **bounded-with-negatives**: имеется нижняя граница, которая меньше нуля, и имеется верхняя граница;

d) **semi-bounded-without-negatives**: имеется нижняя граница, которая больше нуля или равна нулю, но нет верхней границы;

e) **bounded-without-negatives**: имеется нижняя граница, которая больше нуля или равна нулю, и имеется верхняя граница.

Примечание — Для любого заданного набора границ будет удовлетворяться только один предикат.

21.11.5 Если используются последние три enumeration значения из 21.11.1, значение «**Comparison**» типа (см. 21.12) должно быть предоставлено вместе с целочисленным значением **comparator**. Если используются остальные значения, данные не должны предоставляться.

21.12 Тип Comparison

21.12.1 Продукцией типа «**Comparison**» является:

```
Comparison ::= ENUMERATED
        {equal-to,
        not-equal-to,
        greater-than,
        less-than,
        greater-than-or-equal-to,
        less-than-or-equal-to}
```

21.12.2 Значения по умолчанию для признака кодирования этого типа не существуют.

21.12.3 Признак кодирования типа «**Comparison**» используется для сверки выявленного признака класса с целочисленным значением (**comparator**).

21.12.4 Предикат, использующий «**Comparison**», удовлетворяет каждому enumeration значению, если, и только если, идентифицируемый признак удовлетворяет следующим условиям:

a) **equal-to**: его значение равно таковому указанному целочисленному значению **comparator;**

b) **not-equal-to**: его значение отличается от указанного целочисленного значения **comparator;**

c) **greater-than**: его значение больше указанного целочисленного значения **comparator;**

d) **less-than**: его значение меньше указанного целочисленного значения **comparator;**

e) **greater-than-or-equal-to**: его значение не менее указанного целочисленного значения **comparator;**

f) **less-than-or-equal-to**: его значение не более указанного целочисленного значения **comparator**.

21.13 Тип SizeRangeCondition

21.13.1 Продукцией типа «**SizeRangeCondition**» является:

```
SizeRangeCondition ::= ENUMERATED
        { no-ub-with-zero-lb,
          ub-with-zero-lb,
          no-ub-with-non-zero-lb,
          ub-with-non-zero-lb,
          fixed-size
          test-lower-bound,
          test-upper-bound,
          test-range}
```

21.13.2 Безусловным значением (по умолчанию) для признака кодирования этого типа всегда является «**no-ubwith-zero-lb**».

21.13.3 Признак кодирования типа «**SizeRangeCondition**» используется для проверки признаков границ в ограничении реального размера, связанного с классом в категории «повторение» или «цепочка знаков».

21.13.4 Предикат удовлетворяет каждому из первых пяти enumeration значений из 21.13.1, если, и только если, ограничение реального размера удовлетворяет следующим условиям:

a) **no-ub-with-zero-lb**: нет верхней границы размера, а нижняя граница равна нулю;

b) **ub-with-zero-lb**: имеется верхняя граница размера, а нижняя граница равна нулю;

c) **no-ub-with-non-zero-lb**: нет верхней границы размера, а нижняя граница не равна нулю;
d) **ub-with-non-zero-lb**: имеется верхняя граница размера, а нижняя граница не равна нулю,
e) **fixed-size**: нижняя граница и верхняя граница размера имеют одно и то же значение.

Примечание — Только случай «**fixed-size**» пересекается с другими предикатами.

21.14 Тип ReversalSpecification

21.14.1 Продукцией типа «**ReversalSpecification**» является:

```
ReversalSpecification ::= ENUMERATED
        {no-reversal,
        reverse-bits-in-units,
        reverse-half-units,
        reverse-bits-in-half-units}
```

21.14.2 Безусловным значением (по умолчанию) для признака кодирования этого типа всегда является «**noreversal**».

21.14.3 Признак кодирования типа «**ReversalSpecification**» используется при окончательном преобразовании битов из пространства кодирования в выходной буферный накопитель для передачи (с реверсивным преобразованием, применяемым для декодирования).

Примечание — Биты, введенные в результате предварительного заполнения, указанного объектом кодирования, не образуют часть кодирования, для которой этот объект кодирования указал реверсирование битов, но могут быть предметом реверсирования битов, указанного объектом кодирования для контейнера, в который встроено полное кодирование.

21.14.4 Значения этого типа всегда используются совместно с признаком кодирования типа «**Unit**», который указывает размер единицы в битах (см. 21.1).

21.14.5 Спецификация ECN будет ошибочной, если используются значения «**reverse-half-units**» и «**reversebits-in-half-units**», когда признак кодирования типа «**Unit**» имеет нечетное число битов.

21.14.6 Перечисления указываются в следующих случаях (в порядке перечислений, показанном ниже):

a) нет реверсирования битов, либо

b) реверсирование порядка следования половин единиц (без изменения порядка следования битов в каждой половине единицы), либо

c) реверсирование порядка следования битов в каждой половине единицы, но без реверсирования порядка следования половин единиц, либо

d) реверсирование порядка следования битов в каждой единице.

21.14.7 Спецификация ECN будет ошибочной, если число битов в кодировании, к которому применяется реверсирование битов, не является целым, кратным «**Unit**».

21.14.8 Реверсирование битов может быть указано для кодирования всех классов, которые появляются в виде полей структур кодирования, кроме класса кодирования в категории «альтернативы», который не использует понятие пространства кодирования.

21.15 Тип ResultSize

21.15.1 Продукцией типа «**ResultSize**» является:

```
ResultSize ::= INTEGER {variable(-1), fixed-to-max(0)} (-1..MAX)
```

21.15.2 Безусловным значением (по умолчанию) для признака кодирования этого типа всегда является «**variable**».

21.15.3 Признак кодирования этого типа указывает размер результата в классе **#TRANSFORM**.

21.15.4 Значение «**variable**» указывает, что размер результата **#TRANSFORM** будет изменяться для разных абстрактных значений и определяться детальными спецификациями преобразования.

21.15.5 Значение «**fixed-to-max**» указывает, что размер результата **#TRANSFORM** должен быть одинаковым для преобразований всех абстрактных значений. Значение указывает, что размер цели должен быть самым малым размером, который может содержать указанное кодирование любого одного

абстрактного значения (то есть всех). Точные детали этой спецификации определяются для каждого преобразования, в котором используются значения этого типа.

21.15.6 Положительное значение типа «**ResultSize**» указывает, что размер результата **#TRANSFORM** фиксирован. Это значение используется в спецификации реального преобразователя.

21.16 Тип HandleValueSet

21.16.1 Продукцией типа «HandleValueSet» является:

```
HandleValue ::= CHOICE {
        bits BIT STRING,
        octets OCTET STRING,
          number INTEGER (0..MAX),
          tag ENUMERATED {any},
          range SEQUENCE {
                low INTEGER(0..MAX),
                high INTEGER(0..MAX) },
          ranges SET (SIZE(1..MAX)) OF SEQUENCE {
                low INTEGER(0..MAX),
                high INTEGER(0..MAX) }}
```

21.16.2 «**HandleValueSet**» используется для определения набора битовый паттернов (набора значений описателей), характеризующего кодирования, производимые объектами кодирования, которые показывают идентификационный описатель.

21.16.3 Значение идентификационного описателя может использоваться для указания на наличие или отсутствие факультативных компонентов, на выбор альтернативы, на размещение наборов или конец повторения. В этих случаях наборы значений описателей объектов кодирования, применяемых для разных альтернатив или компонентов, должны быть непересекающимися (см. 21.5.7, 21.6.6, 21.7.10 и 22.10.2.1), и все возможные значения идентификационного описателя, появляющиеся при кодированиях любых заданных альтернатив или компонентов, должны соответствовать указанному набору значений описателя объекта кодирования, применяемого к такой альтернативе или компоненту (см. 22.9.2.2).

Примечание — Разработчик ECN должен определять набор значений описателя во всех случаях, кроме тех, когда (при кодировании класса «тег») набор значений описателя состоит из единственного значения и зависит от номера тега, связанного с этим классом «тег», либо прямо с помощью неявной генерации из тега ACH.1, либо с помощью отображения из неявно генерируемой структуры.

21.16.4 Альтернативы «**bits**», «**octets**» и «**number**» указывают значения описателя в виде значений цепочки битов, цепочки октетов и целочисленного значения соответственно. Спецификация ECN будет ошибочной, если это значение не может быть кодировано в пределах числа битов, указанного до идентификационного описателя (см. 22.9).

21.16.5 Альтернатива «**tag:any**» указывает, что значение описателя определяется числом, определенным в структуре кодирования ECN для классов в категории «тег», либо номером тега, отображенным из теговой конструкции ACH.1. Она используется только при описании идентификации описателя для кодирования класса в категории «тег».

21.16.6 Альтернатива «**range**» указывает диапазон целочисленных значений, где **high** не менее **low**.

21.16.7 Альтернатива «**ranges**» указывает набор диапазонов целочисленных значений, где **high** не менее **low**. Один или несколько таких диапазонов могут быть определены, и они не должны пересекаться.

21.17 Тип IntegerMapping

21.17.1 Продукцией типа «**IntegerMapping**» является:

```
IntegerMapping ::= SET OF SEQUENCE {
        source SET OF INTEGER,
        result INTEGER} (CONSTRAINED BY {/* the intersection of the
                source components shall be empty */})
```

21.17.2 «**IntegerMapping**» используется для явного указания ints-to-ints преобразования.

22 Обычно используемые группы признаков кодирования

В настоящем разделе описываются группы признаков кодирования, которые обычно используются в определенном синтаксисе (см. раздел 20). Описываются также цели каждой группы, ограничения на значения признаков кодирования и на синтаксис, который может использоваться, а также действия кодера и декодера для каждой группы.

22.1 Спецификация замены

Имеются три варианта спецификации замены:

a) спецификация полной замены: она используется для классов в категории «конкатенация», где замена возможна для целой структуры либо возможна селективная замена факультативных и нефакультативных компонентов;

b) спецификация замены структуры или компонента: она используется для классов в категории «альтернативы» и для класса кодирования **#CONDITIONAL-REPETITION**, где замена возможна для полной структуры или компонента.

Примечание — Когда объект кодирования класса **#CONDITIONAL-REPETITION** используется при определении кодирований для класса в категориях «цепочка битов», «цепочка знаков» или «цепочка октетов», он может выполнять только замену исключительно структуры;

c) спецификация замены исключительно структуры: она используется для классов, которые не имеют компонентов.

22.1.1 Признаки кодирования, синтаксис и цель

22.1.1.1 В спецификации полной замены используются следующие признаки кодирования:

```
        &#Replacement-structure          OPTIONAL,
        &#Replacement-structure2         OPTIONAL,
&replacement-structure-encoding-object &#Replacement-structure          OPTIONAL,
&replacement-structure-encoding-object2 &#Replacement-structure2         OPTIONAL,
&#Head-end-structure          OPTIONAL,
&#Head-end-structure2         OPTIONAL
```

22.1.1.2 Для спецификации полной замены должен использоваться следующий синтаксис:

```
[REPLACE
        [STRUCTURE]
        [COMPONENT]
        [ALL COMPONENTS]
        [OPTIONALS]
        [NON-OPTIONALS]
        WITH &#Replacement-structure
                [ENCODED BY &replacement-structure-encoding-object
                        [INSERT AT HEAD &#Head-end-structure]]
                [AND OPTIONALS WITH &#Replacement-structure2
                        [ENCODED BY &replacement-structure-encoding-object2
                                [INSERT AT HEAD &#Head-end-structure2]]] ]
```

22.1.1.3 В спецификации замены структуры или компонента используются следующие признаки кодирования:

```
&#Replacement-structure
        OPTIONAL,
        &replacement-structure-encoding-object &#Replacement-structure OPTIONAL,
        &#Head-end-structure          OPTIONAL
```

22.1.1.4 Для спецификации замены структуры или компонента должен использоваться следующий синтаксис:

```
[REPLACE
        [STRUCTURE]
        [COMPONENT]
        [ALL COMPONENTS]
        WITH &Replacement-structure
        [ENCODED BY &replacement-structure-encoding-object
                [INSERT AT HEAD &#Head-end-structure]]]
```

22.1.1.5 В спецификации замены исключительно структуры используются следующие признаки кодирования:

```
&#Replacement-structure
        OPTIONAL,
&replacement-structure-encoding-object &#Replacement-structure OPTIONAL
```

22.1.1.6 Для спецификации замены исключительно структуры должен использоваться следующий синтаксис:

```
[REPLACE
        [STRUCTURE]
        WITH &#Replacement-structure
                [ENCODED BY &replacement-structure-encoding-object]]
```

22.1.1.7 Использование «**WITH SYNTAX**» для этих групп признаков кодирования указывает, что:

a) класс кодирования, к которому применяется этот объект кодирования, должен заменяться полностью («**REPLACE STRUCTURE**»); в случае класса кодирования в категории «факультативные возможности» заменяется весь компонент; в случае объекта кодирования **#CONDITIONAL-REPETITION**, используемого при определении объекта кодирования для класса в категориях «цепочка битов», «цепочка знаков», «цепочка октетов» или «повторение», заменяется (если условие по диапазону удовлетворяется) полная структура цепочки битов, цепочки знаков, цепочки октетов или повторения, либо

b) все его компоненты (кроме используемых для спецификации исключительно структуры) должны заменяться (одним и тем же действием замены для всех компонентов) («**REPLACE COMPONENT**» или «**REPLACE ALL COMPONENTS**»), либо

c) все его факультативные компоненты (только для спецификации полной замены) должны заменяться («**REPLACE OPTIONALS**»), либо

d) все его нефакультативные компоненты (только для спецификации полной замены) должны заменяться («**REPLACE NON-OPTIONALS**»), либо

e) все его компоненты (только для спецификации полной замены) должны заменяться разными действиями замены для факультативных и нефакультативных компонентов («**REPLACE NON-OPTIONALS AND OPTIONALS**»).

22.1.1.8 «**REPLACE COMPONENT**» является синонимом для «**REPLACE ALL COMPONENTS**». Это было бы нормально, но это не требуется применять, если имеется только единственный компонент.

22.1.1.9 Факультативный «**ENCODED BY**» указывает объект кодирования для структуры замены.

22.1.1.10 Факультативный «**INSERT AT HEAD**» определяет структуру кодирования (вставку головного узла), вводимую перед всеми компонентами класса (конструктора), выполняющего замену. Имеется одна вставка головного узла для каждого компонента, который заменяется, причем они вводятся в порядке следования исходных компонентов.

22.1.1.11 В спецификации полной замены, если объект кодирования, применяемый к структуре замены, показывает идентификационный описатель (с заданным набором значений описателя), то объект кодирования, описанный синтаксис которого содержит спецификацию полной замены, показывает такой же идентификационный описатель (с таким же набором значений описателя); в противном случае он не показывает описатель.

22.1.2 Ограничения на спецификацию

22.1.2.1 Должен использоваться только один из разрешенных синтаксисов между «**REPLACE**» и «**WITH**».

22.1.2.2 Структуры замены «**WITH**» будут являться параметризованными структурами кодирования с одним параметром класса кодирования. Когда они определяются в вышеопределенном синтаксисе, дается только справочное имя класса структуры. При таком использовании имен не требуется иметь список параметров.

22.1.2.3 Эти параметризованные структуры реализуются во время действия замены с реальным параметром, как указывается в 22.1.3. Использование фиктивного параметра в параметризованной структуре замены должно быть согласовано с классом реального параметра, который будет подан при действии замены.

Примечание — В частности, если «**REPLACE STRUCTURE**» используется для класса кодирования в категории «тег», то фиктивный параметр может появляться в структуре замены только тогда, когда разрешен класс кодирования в категории «тег».

22.1.2.4 Объектами кодирования «**ENCODED BY**» должны быть параметризованные объекты кодирования со структурами кодирования «**WITH**». Они должны иметь фиктивный параметр (например, **#D**), который является классом кодирования, и должны быть определены в назначении параметризованного объекта кодирования, в котором руководителем является соответствующая параметризованная структура кодирования «**WITH**», созданная с **#D**. Когда они описываются в вышеуказанном определенном синтаксисе, должно даваться только справочное имя объекта кодирования. При таком использовании имен они не должны иметь списка параметров.

22.1.2.5 Они создаются во время действия замены с реальным параметром, который имеет то же значение, что и реальный параметр, использованный для создания соответствующих структур кодирования замены «**WITH**». Они могут также иметь.

- другой (но только один) фиктивный параметр (факультативный), который является набором объектов кодирования; когда они создаются во время действия замены, тогда реальным параметром для этого фиктивного параметра является текущий комбинированный набор объектов кодирования;

- другой (но только один) фиктивный параметр (условный), который является параметром **REFERENCE**. Этот параметр присутствует, если, и только если, указан «**INSERT AT HEAD**». Когда объекты кодирования создаются во время действия замены, тогда реальным параметром для этого фиктивного параметра является ссылка на соответствующую структуру «**INSERT AT HEAD**».

22.1.2.6 Все поля структуры замены, которые не являются частью параметра класса кодирования, являются вспомогательными полями и устанавливаются путем кодирования структуры замены.

22.1.2.7 Структуры кодирования «**INSERT AT HEAD**» не имеют фиктивных параметров. Все их поля являются вспомогательными полями и устанавливаются объектом кодирования «**ENCODED BY**» с помощью параметра **REFERENCE**.

22.1.2.8 Если объект кодирования имеет раздел «**REPLACE STRUCTURE**», то он не должен иметь раздел «**INSERT AT HEAD**» и должен иметь раздел «**ENCODED BY**».

22.1.3 Действия кодера

22.1.3.1 Если объект кодирования класса, входящего в группу категорий «базовое поле» или в категорию «тег», указывает «**REPLACE STRUCTURE**», то кодер заменяет эту структуру на реализацию структуры замены, используя имя исходной структуры в качестве реального параметра.

22.1.3.2 Если объект кодирования класса в группе категорий «конструктор кодирования» указывает «**REPLACE STRUCTURE**», то кодер заменяет полную конструкцию на реализацию структуры замены, используя полную исходную конструкцию в качестве реального параметра.

22.1.3.3 Если объект кодирования класса в категории «факультативные возможности» указывает «**REPLACE STRUCTURE**», то кодер заменяет полный факультативный компонент на нефакультативную реализацию структуры замены. Реальным параметром должно быть имя невидимой структуры (которая не соответствует никакой другой структуре и не может иметь объектов кодирования). Это имя невидимой структуры должно разыменовывать в полный исходный факультативный компонент (включая любой класс в категории «тег»), кроме класса в категории «факультативные возможности».

22.1.3.4 Если объект кодирования любого класса указывает «**REPLACE COMPONENT**», «**REPLACE ALL COMPONENTS**», «**REPLACE OPTIONAL COMPONENTS**» или «**REPLACE NON-OPTIONAL COMPONENTS**», то кодер заменяет полный(ые) указанный(ые) компонент(ы) на нефакультативную реализацию структуры замены. Реальным параметром должно быть имя невидимой структуры (которая

не соответствует никакой другой структуре и не может иметь объектов кодирования). Это имя невидимой структуры должно разыменовывать в полный исходный факультативный компонент (включая любой класс в категории «тег»), кроме класса в категории «факультативные возможности».

22.1.3.5 Все абстрактные значения и номера тегов исходной структуры или исходного компонента должны отображаться в соответствующие абстрактные значения и номера тегов в реальном параметре структуры замены. Значения других полей в структуре устанавливаются согласно спецификации в объекте кодирования структуры замены.

22.1.3.6 Если указана вставка головного узла, то кодер вставляет структуру головного узла перед всеми компонентами структуры, объект кодирования которой выполняет замену. Вставка головного узла должна вставляться в том же текстуальном порядке, в каком следуют заменяемые компоненты. Значения полей в этой структуре устанавливаются согласно спецификации в объекте кодирования структуры замены.

Примечание — Этими структурами обычно будут простые целочисленные поля, обеспечивающие местный определитель для заменяемого поля.

22.1.3.7 Кодер создает объект (объекты) кодирования структуры замены с реальными параметрами следующим образом:

a) фиктивный параметр, являющийся классом кодирования, должен означать реальный параметр, имеющий то же значение, что и реальный параметр реализации структуры замены;

b) фиктивный параметр (если он есть), являющийся параметром **REFERENCE,** должен означать реальный параметр, являющийся ссылкой на вставленную структуру головного узла;

c) фиктивный параметр (если он есть), являющийся набором объектов кодирования (чьим руководителем является **#ENCODINGS**), должен означать реальный параметр, являющийся текущим комбинированным набором объектов кодирования.

22.1.3.8 Затем кодер использует этот реализованный объект кодирования для кодирования соответствующей структуры замены вместо комбинированного набора объектов кодирования.

Примечание — Кодирование вставок головного узла определяется применением текущего комбинированного набора объектов кодирования.

22.1.4 Действия декодера

Декодер генерирует (для некоторого применения) абстрактные значения исходной структуры, которая была кодирована, делая невидимыми любые действия замены (даже выполненные путем повторяющегося применения замен).

22.2 Спецификация предварительного выравнивания и заполнения

22.2.1 Признаки кодирования, синтаксис и цель

22.2.1.1 В спецификации предварительного выравнивания и заполнения используются следующие признаки кодирования:

```
&encoding-space-pre-alignment-unit    Unit (ALL EXCEPT repetitions) DEFAULT bit,
&encoding-space-pre-padding           Padding DEFAULT zero,
&encoding-space-pre-pattern           Non-Null-Pattern (ALL EXCEPT different:any)
                                      DEFAULT bits:'0'B
```

22.2.1.2 Для спецификации предварительного выравнивания и заполнения должен использоваться следующий синтаксис:

```
[ALIGNED TO
        [NEXT]
        [ANY]
                &encoding-space-pre-alignment-unit
                [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
```

22.2.1.3 Определением типов, используемых в спецификации предварительного выравнивания и заполнения, является:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
      dword32(32)} (0..256) -- (см. 21.1)
Padding ::= ENUMERATED {zero, one, pattern, encoder-option} -- (см. 21.9)
Pattern ::= CHOICE
        {bits              BIT STRING,
         octets            OCTET STRING,
         char8             IA5String,
         char16            BMPString,
         char32            UniversalString,
         any-of-length     INTEGER (1..MAX),
         different         ENUMERATED {any} }
Non-Null-Pattern ::= Pattern
      (ALL EXCEPT (bits:"B | octets:"H | char8:"" | char16:"" |
            char32:"")) -- (см. 21.10)
```

22.2.1.4 Признаки кодирования предварительного выравнивания используют значение типа «**Unit**» для указания, что контейнер должен начинаться с целого числа битов, кратного «**Unit**», от точки выравнивания. Точкой выравнивания является начало кодирования типа, к которому ELM применяет кодирование, кроме случаев, когда применен сброс для кодирования вложенного типа путем использования объекта кодирования **#OUTER** (см. раздел 25). Признаки кодирования типов «**Padding**» и «**Pattern**» используются для управления битами, которые обеспечивают заполнение для затребованного выравнивания. Спецификация «**ALIGNED TO NEXT**» создает минимальное число вставляемых битов. Спецификация «**ALIGNED TO ANY**» оставляет реальное число вставляемых битов (подчиненное вышеуказанному ограничению по кратности с «**Unit**») на усмотрение кодеров и требует определения начального указателя.

22.2.2 Ограничения на спецификацию

22.2.2.1 Должен указываться один самый больший из «**NEXT**» и «**ANY**». Если не указывается, то подразумевается «NEXT».

22.2.2.2 Если указывается «**ALIGNED TO ANY**», то спецификация объекта кодирования должна содержать раздел «**START-POINTER**».

22.2.3 Действия кодера

22.2.3.1 Если указывается «**NEXT**» (или применяется по умолчанию), то кодер вставляет минимальное число битов, необходимое для гарантии, что общее число битов в кодировании (от точки выравнивания до начала контейнера, см. 22.2.1.4) будет кратно признаку кодирования типа «**Unit**».

22.2.3.2 Если указывается «**ANY**», то кодер вставляет независящее от кодера число битов при условии, что общее число битов в кодировании (от точки выравнивания) будет кратно признаку кодирования типа «**Unit**».

22.2.3.3 Вставляемые биты устанавливаются так, чтобы первый вставляемый бит был начальным битом «**Pattern**» и т. д. Если необходимо больше битов, чем имеется в признаке кодирования типа «**Pattern**», то эта комбинация используется повторно, помещая первым старший значащий бит.

22.2.4 Действия декодера

22.2.4.1 Декодер определяет число вставленных битов из действий кодера, если указан «**NEXT**».

22.2.4.2 Декодер определяет число вставленных битов из спецификации начального указателя, если указан «**ANY**».

22.2.4.3 Во всех случаях декодер сбрасывает вставленные биты незаметно для приложения. Он не должен распознавать ошибку кодера или спецификации, когда биты не согласуются с указанными действиями кодеров.

22.3 Спецификация начального указателя

22.3.1 Признаки кодирования, синтаксис и цель

22.3.1.1 В спецификации начального указателя используются следующие признаки кодирования:

```
&start-pointer                    REFERENCE OPTIONAL,
&start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
&Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL
```

22.3.1.2 Для спецификации начального указателя должен использоваться следующий синтаксис:

[START-POINTER &start-pointer
[MULTIPLE OF &start-pointer-unit]
[ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]

22.3.1.3 Определением типа, используемого в спецификации начального указателя, является:

Unit ::= INTEGER
{repetitions(0), bit(1), nibble(4), octet(8), word16(16),
dword32(32)} (0..256) — (см. 21.1)

22.3.1.4 Эта спецификация указывает начало пространства кодирования для элемента. Если начало пространства кодирования для элемента смещено на «n» единиц «**MULTIPLE OF**», то значением, которое помещено в поле, указанном признаком кодирования «**START-POINTER**», является значение, которое получено путем применения «**ENCODER-TRANSFORMS**» к «n».

Примечания

1 Если значением «**MULTIPLE OF**» является не «**bits**», то это означает, что смещение от начала поля, указанного признаком кодирования «**START-POINTER**», до начала пространства кодирования должно быть кратно единицам «**MULTIPLE OF**» целое число раз.

2 В общем случае могут быть кодирования других элементов и, возможно, другие начальные указатели между полем, указанным с помощью признака кодирования «**START-POINTER**», и началом кодирования этого элемента.

22.3.2 Спецификация ограничений

22.3.2.1 Если «**ENCODER-TRANSFORMS**» не присутствует, то «**START-POINTER**» будет классом в «целочисленной» категории.

22.3.2.2 Если «**ENCODER-TRANSFORMS**» присутствует, то «**START-POINTER**» будет началом в категории, которая может кодировать значение результата окончательного преобразования в «**ENCODER-TRANSFORMS**».

22.3.2.3 Спецификация ECN или применение будут ошибочными, когда какое-либо преобразование в «**ENCODER-TRANSFORMS**» необратимо для абстрактного значения, к которому оно применено. Первое преобразование должно иметь источник, являющийся целым числом.

22.3.3 Действия кодера

22.3.3.1 Кодер определяет число «n» единиц «**MULTIPLE OF**» от начала кодирования поля «**START-POINTER**» (после предварительного выравнивания этого поля) до начала кодирования элемента со спецификацией начального указателя (после предварительного выравнивания этого элемента). Спецификация ECN будет ошибочной, когда «n» не является целым числом. Если кодируемый элемент является факультативным и отсутствует, то «n» устанавливается в нуль.

22.3.3.2 Значение «n» будет преобразовано с помощью «**ENCODER-TRANSFORMS**» (если он имеется) для образования смыслового значения «m». Если это результирующее значение «m» не является абстрактным значением, которое может быть связано с классом кодирования «**START-POINTER**», то возникает ошибка спецификации ECN, а кодирование не будет продолжено. В остальных случаях значение «m» должно быть значением, кодированным в поле, указанном в «**START-POINTER**».

Примечание — Объект кодирования, приложенный к полю, указанному в «**START-POINTER**», будет определять кодирование значения «m».

22.3.4 Действия декодера

22.3.4.1 Декодер определяет смысловое значение «m» в поле, указанном «**START-POINTER**», и использует знание действий кодера для реверсирования преобразований (если они есть), чтобы образовать целочисленное значение «n».

22.3.4.2 Если «n» равно нулю, то декодер распознает ошибку кодера, если декодируемый элемент не является факультативным элементом со спецификацией факультативной возможности, определяющей факультативную возможность начальным указателем. Если «n» равно нулю, а декодируемый элемент является факультативным элементом со спецификацией факультативной возможности, определяющей факультативную возможность начальным указателем, то декодер определяет, что этот элемент отсутствует.

22.3.4.3 Значение «n» умножается с помощью «**MULTIPLE OF**», а начало кодирования из поля «**START-POINTER**» добавляется для получения позиции «p». Если «p» является позицией в кодировании, которая появляется ранее текущей точки декодирования, то декодер распознает ошибку кодирования.

22.3.4.4 Если «p» является позицией в кодировании, которая совпадает с текущей точкой декодирования или расположена за ней, то декодер молча игнорирует все биты до позиции «p» и продолжает декодирование этого элемента, начиная с позиции «p».

22.4 Спецификация пространства кодирования

22.4.1 Признаки кодирования, синтаксис и цель

22.4.1.1 В спецификации пространства кодирования используются следующие признаки кодирования:

```
&encoding-space-size            EncodingSpaceSize
                                DEFAULT self-delimiting-values,
&encoding-space-unit            Unit (ALL EXCEPT repetitions)
                                DEFAULT bit,
&encoding-space-determination   EncodingSpaceDetermination
                                DEFAULT field-to-be-set,
&encoding-space-reference       REFERENCE OPTIONAL,
&Encoder-transforms             #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms             #TRANSFORM ORDERED OPTIONAL
```

22.4.1.2 Для спецификации пространства кодирования должен использоваться следующий синтаксис:

```
ENCODING-SPACE
        [SIZE &encoding-space-size
                [MULTIPLE OF &encoding-space-unit]]
        [DETERMINED BY &encoding-space-determination]
        [USING &encoding-space-reference
                [ENCODER-TRANSFORMS &Encoder-transforms]
                [DECODER-TRANSFORMS &Decoder-transforms]]
```

22.4.1.3 Определением типов, используемых в этой спецификации, является:

```
EncodingSpaceSize ::= INTEGER
                { encoder-option-with-determinant(-3),
                  variable-with-determinant(-2),
                  self-delimiting-values(-1),
                  fixed-to-max(0)} (-3..MAX) -- (см. 21.2)
                Unit ::= INTEGER
                        {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
dword32(32)} (0..256) -- (см. 21.1)
EncodingSpaceDetermination ::= ENUMERATED
        {field-to-be-set, field-to-be-used, container} -- (см. 21.3)
```

22.4.1.4 Цель этой спецификации — определить действия кодера и декодера для гарантирования того, что декодер сможет правильно определить конец пространства кодирования.

Примечание — Реальное кодирование значения не обязательно заполнит все пространство кодирования, поэтому восстановление кодирования значения декодером будет обычно требовать действий, определяющих заполнение и выравнивание значения (см. 22.8).

22.4.1.5 Смысл признаков кодирования типов «**Unit**», «**EncodingSpaceSize**» и «**EncodingSpaceDetermination**» описан в 21.1—21.3. Они вместе указывают способ определения конца пространства кодирования для этого элемента.

Примечание — Даже при фиксированном размере пространства кодирования может быть указан «**variable-withdeterminant**», если спецификатор ECN требует, чтобы был введен определитель длины, даже когда он не нужен.

22.4.1.6 Спецификацией «**USING**» является ссылка, позволяющая декодеру определить конец пространства кодирования. Она является ссылкой на вспомогательное поле, или на поле, переносящее абстрактные значения, или на контейнер, в зависимости от значения «**DETERMINED BY**».

22.4.2 Ограничения на спецификацию

22.4.2.1 Если «**SIZE**» имеет значение «**variable-with-determinant**», а «**DETERMINED BY**» отсутствует, то по умолчанию предполагается безусловное значение («**field-to-be-set**»).

22.4.2.2 «**USING**» указывается, если, и только если, «**SIZE**» имеет значение «**variable-with-determinant**» или «**encoder-option-with-determinant**».

22.4.2.3 «**ENCODER-TRANSFORMS**» присутствует, если только «**DETERMINED BY**» установлен (возможно, по умолчанию) в «**field-to-be-set**». Ссылка «**USING**» в этом случае будет вспомогательным полем категорий «цепочка битов», «цепочка знаков» или «целочисленная».

22.4.2.4 Спецификация ECN или применение будут ошибочными, когда преобразование в «**ENCODER-TRANSFORMS**» не является обратимым для абстрактного значения, к которому оно применено. Первый преобразователь должен иметь источник, который является целым числом, а последний преобразователь должен иметь результат, который может быть кодирован классом поля, указанного в «**USING**».

22.4.2.5 «**DECODER-TRANSFORMS**» присутствует, если только «**DETERMINED BY**» установлен в «**field-to-beused**». Первый преобразователь должен иметь источник, который совпадает с категорией поля, указанного в «**USING**» и не являющегося вспомогательным полем. Последний преобразователь должен иметь результат, являющийся целым числом.

22.4.2.6 Признак кодирования «**USING**», если присутствует, должен быть ссылкой на поле, появившееся в кодировании раньше, чем кодируемое поле. Применение или спецификация ECN будут ошибочными, когда в экземпляре кодирования поле, подлежащее кодированию, присутствует, но поле, указанное признаком кодирования «**USING**», отсутствует (из-за применения факультативной возможности).

22.4.2.7 Если «**DETERMINED BY**» дает «**container**», то ссылка «**USING**» должна быть на конкатенацию или повторение (либо на цепочку битов или цепочку октетов с вложенным типом), в которых кодируемый элемент является компонентом (или компонентом компонента на любую глубину). Применение или спецификация ECN будут ошибочными, когда в экземпляре кодирования должны кодироваться последние элементы внутри одной и той же конкатенации или повторения.

22.4.2.8 Эта спецификация считается установленной, если использовано ключевое слово «**ENCODING-SPACE**», кроме того, для нее обязательно, чтобы она была установлена во всех местах определенного синтаксиса, где она разрешена. Применение безусловных значений (по умолчанию) для всех признаков кодирования этой группы (например, использование одиночного «**ENCODING-SPACE**») может не удовлетворять вышеуказанным ограничениям.

22.4.3 Действия кодера

22.4.3.1 Кодеры не будут генерировать кодовые последовательности, если не удовлетворяются условия 22.4.2.

22.4.3.2 Если «**SIZE**» является положительным значением, то пространство кодирования будет кратно единицам «**MULTIPLE OF**», а последующие действия кодера отсутствуют.

22.4.3.3 Если «**SIZE**» не установлен в положительное значение, то кодер определяет размер (например, «s») пространства кодирования в единицах «**MULTIPLE OF**» из спецификации кодирования значения. Это определение описано в разделах о спецификации кодирования значения.

22.4.3.4 Если «**SIZE**» является «**encoder-option-with-determinant**», то кодер может (по своему выбору) увеличить размер «s» (указанный в 22.4.3.3) на единицы «**MULTIPLE OF**» по сравнению с размером, определенным спецификацией кодирования значения, до значения, которое может быть кодировано в соответствующем определителе.

22.4.3.5 Если «**SIZE**» равен «**fixed-to-max**» или «**self-delimiting-values**», то последующие действия кодера отсутствуют.

22.4.3.6 Если «**SIZE**» равен «**variable-with-determinant**», а «**DETERMINED BY**» равен «**container**», то последующие действия кодера отсутствуют.

22.4.3.7 Если «**DETERMINED BY**» равен «**field-to-be-set**», то кодер применяет преобразователи, указанные в «**ENCODER-TRANSFORMS**» (если он имеется), к значению «s» для получения значения, которое будет кодироваться в ссылке «**USING**».

Примечание — Кодирование ссылки «**USING**» (например, битового поля «A») в этом случае появляется в кодировании раньше, чем кодирование этого поля (например, битового поля «B»), а кодер должен будет отложить кодирование битового поля «A» до определения значения, которое будет кодироваться, с помощью кодирования битового поля «B».

22.4.3.8 Если «**DETERMINED BY**» равен «**field-to-be-used»,** то кодер проверяет, что значение в ссылке **«USING»** при преобразовании с помощью **«DECODER-TRANSFORMS»** (если он есть) равно «s». Применение будет ошибочным, когда это условие не удовлетворяется, а кодирование не продолжается.

22.4.4 Действия декодера

22.4.4.1 Если «**SIZE**» является положительным значением, то декодер определяет пространство кодирования как кратное единицам **«MULTIPLE OF».**

22.4.4.2 Если «**SIZE**» равен «**fixed-to-max**» или «**self-delimiting-values**», то декодер определяет конец пространства кодирования согласно спецификации кодирования значения. Это определение описано в разделах о спецификации кодирования значения.

22.4.4.3 Если «**SIZE**» равен «**variable-with-determinant**», а «**DETERMINED BY**» установлен в «**container**», то декодер использует конец контейнера, указанного в «**USING**», в качестве конца пространства кодирования.

22.4.4.4 Если «**SIZE**» равен «variable-with-determinant», а «**DETERMINED BY**» установлен (возможно, по умолчанию) в «**field-to-be-set**», то декодер восстанавливает значение «s» путем применения реверсирования «**ENCODERTRANSFORMS**» (если он имеется) к значению ссылки «**USING**».

22.4.4.5 Если «**DETERMINED BY**» равен «**field-to-be-used**», то декодер восстанавливает значение «s» путем применения «**DECODER-TRANSFORMS**» (если он имеется) к значению этого поля.

22.5 Определение факультативных возможностей

22.5.1 Признаки кодирования, синтаксис и цель

22.5.1.1 При определении факультативных возможностей используются следующие признаки кодирования:

```
&optionality-determination   OptionalityDetermination
                             DEFAULT field-to-be-set,
&optionality-reference       REFERENCE OPTIONAL,
&Encoder-transforms          #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms          #TRANSFORM ORDERED OPTIONAL,
&handle-id                   PrintableString
                             DEFAULT "default-handle"
```

22.5.1.2 Для определения факультативных возможностей должен использоваться следующий синтаксис:

```
PRESENCE
        [DETERMINED BY &optionality-determination
                [HANDLE &handle-id]]
        [USING &optionality-reference
                [ENCODER-TRANSFORMS &Encoder-transforms]
                [DECODER-TRANSFORMS &Decoder-transforms]]
```

22.5.1.3 Определением типов, используемых в описании факультативных возможностей, является:

```
OptionalityDetermination ::= ENUMERATED
{field-to-be-set, field-to-be-used, container, handle, pointer} -- (см. 21.5)
```

22.5.1.4 Цель этой спецификации — описать правила, обеспечивающие декодеру возможность правильно определить, что кодер закодировал значение факультативного компонента. Когда для определения факультативной возможности используется какой-либо указатель, необходима также спецификация предварительного выравнивания и начального указателя.

22.5.1.5 Кодер будет кодировать значение факультативного компонента, если этого требует приложение, при условии, что такое кодирование не нарушит правил, руководящих присутствием факультативных компонентов.

Примечание — Примером нарушения такого правила будет случай, когда наличие (отсутствие) факультативного компонента было определено концом контейнера, а приложение позже запросило, чтобы в том же контейнере были кодированы факультативные компоненты.

22.5.1.6 Эта спецификация считается установленной, если использовано ключевое слово «**PRESENCE**», которое обязательно должно быть установлено во всех местах в определенном синтаксисе, где оно разрешено. Установка по умолчанию всех остальных частей этого определенного синтаксиса (например, использование одиночного «**PRESENCE**») может не удовлетворять вышеприведенным ограничениям.

22.5.2 Ограничения на спецификацию

22.5.2.1 Если «**DETERMINED BY**» не присутствует, то подразумевается безусловное значение (по умолчанию) («**field-to-be-set**»).

22.5.2.2 «**HANDLE**» не указывается, если «**DETERMINED BY**» не равен «**handle**».

22.5.2.3 «**USING**» не указывается, если «**DETERMINED BY**» имеет значение «**handle**» или «**pointer**».

22.5.2.4 Если «**DETERMINED BY**» равен «**pointer**», то в том же объекте кодирования должна быть спецификация «**START-POINTER**» (см. 22.3).

Примечание — Спецификация начального указателя обычно требует также спецификации предварительного выравнивания с «**ALIGNED TO ANY**» (см. 22.2).

22.5.2.5 Если «**DETERMINED BY**» является «**handle**», применяется 21.5.7.

22.5.2.6 **«ENCODING-TRANSFORMS»** присутствует, если только **«DETERMINED BY»** установлен (возможно, по умолчанию) в «**field-to-be-set**». Ссылка «**USING**» в этом случае является вспомогательным полем категорий «цепочка битов», «булева», «цепочка знаков» или «целочисленная».

22.5.2.7 Спецификация ECN или применение будут ошибочными, когда преобразователь в «**ENCODER-TRANSFORMS**» не является обратимым для абстрактного значения, к которому он применен. Первый преобразователь должен иметь источник с булевым значением, а последний преобразователь должен иметь результат, который может быть кодирован классом поля, указанного в «**USING**».

22.5.2.8 «**DECODER-TRANSFORMS**» присутствует, если только «**DETERMINED BY**» установлен в «**field-to-beused**». Первый преобразователь должен иметь источник той же категории, что и поле, указанное в «**USING**» и не являющееся вспомогательным полем. Последний преобразователь должен иметь результат с булевым значением.

22.5.2.9 Признак кодирования «**USING**», если имеется, будет ссылкой на поле, которое присутствует в кодировании раньше, чем поле, чье присутствие определяется. Применение или спецификация ECN будут ошибочными, когда в экземпляре кодирования поле, указанное признаком кодирования «**USING**», затребовано декодером, но отсутствует (из-за применения факультативной возможности).

22.5.2.10 Если «**DETERMINED BY**» равен «**container**», то ссылка «**USING**» будет на конкатенацию или на повторение (либо на цепочку битов или цепочку октетов с вложенным типом), в которых кодируемый элемент является компонентом (или компонентом компонента на любую глубину). Применение или спецификация ECN будут ошибочными, когда в экземпляре кодирования должны кодироваться последние элементы внутри одной и той же конкатенации или повторения, а компонент, факультативное значение которого определяется, отсутствует.

22.5.2.11 Если «**DETERMINED BY**» равен «**container**», то возникнет ошибка спецификации ECN, когда какое-либо абстрактное значение факультативного компонента имеет кодовую последовательность длиной нуль битов.

22.5.3 Действия кодера

22.5.3.1 Кодеры не будут генерировать кодовые последовательности, если не удовлетворяются условия 22.5.2.

22.5.3.2 Кодер определяет, желает ли приложение кодировать факультативный компонент, и создает смысловое булево значение «**element-is-present**», установленное в «**TRUE**», когда значение компонента должно кодироваться, или в «**FALSE**» в противном случае.

22.5.3.3 Если «**DETERMINED BY**» равен «**field-to-be-set**», то кодер применяет преобразователи, указанные в «**ENCODER-TRANSFORMS**» (если он имеется), к смысловому булеву значению «**element-is-present**» для создания значения, которое будет кодироваться в ссылке «**USING**».

Примечание — Кодирование ссылки «**USING**» в этом случае появляется в кодировании раньше, чем кодирование этого поля, а кодер должен будет отложить кодирование этого поля до определения значения, которое будет кодироваться, с помощью кодирования этого поля.

22.5.3.4 Если «**DETERMINED BY**» равен «**field-to-be-used**», то кодер проверяет, является ли значение в ссылке «**USING**» при преобразовании с помощью «**DECODER-TRANSFORMS**» (если он есть) булевым значением, равным смысловому значению «**element-is-present**». Применение будет ошибочным, когда это условие не удовлетворяется, и кодирование не будет продолжаться.

22.5.3.5 Если «**DETERMINED BY**» равен «**container**», то от кодера не требуется дальнейших действий, кроме обнаружения ошибки и прекращения кодирования, если приложение запросило кодирование следующих компонентов в контейнер «**USING**», когда смысловое значение «**element-is-present**» для этого факультативного компонента равно «**FALSE**».

22.5.3.6 Если «**DETERMINED BY**» равен «**handle**», то от кодера не требуется дальнейших действий.

22.5.3.7 Если «**DETERMINED BY**» равен «**pointer**», то от кодера не требуется дальнейших действий, кроме тех, которые сопровождают предварительное выравнивание (если оно имеется) и спецификации начальных указателей.

22.5.4 Действия декодера

22.5.4.1 Если «**DETERMINED BY**» установлен (возможно, по умолчанию) в «**field-to-be-set**», то декодер восстанавливает значение «**element-is-present**» путем применения реверсии «**ENCODER-TRANSFORMS**» (если он есть) к значению ссылки «**USING**».

22.5.4.2 Если «**DETERMINED BY**» равен «**field-to-be-used**», то декодер восстанавливает смысловое значение «**element-is-present**» путем применения «**DECODER-TRANSFORMS**» (если он есть) к значению этого поля.

22.5.4.3 Если «**DETERMINED BY**» равен «**container**», то декодер устанавливает смысловое значение «**element-ispresent**» в «**TRUE**», если, и только если, имеется по меньшей мере один бит, остающийся в контейнере «**USING**».

22.5.4.4 Если «**DETERMINED BY**» равен «**handle**», то декодер определяет значение указанного идентификационного описателя. Если это значение соответствует значению идентификационного описателя факультативного компонента, то декодер устанавливает смысловое значение «**element-is-present**» в **TRUE**, а в противном случае декодер устанавливает его в **FALSE**.

22.5.4.5 Если «**DETERMINED BY**» равен «**pointer**», то декодер продолжает работу согласно 22.3, чтобы определить смысловое значение «**element-is-present**».

22.5.4.6 Если декодер определит (одним из вышеописанных способов), что смысловое значение «**element-ispresent**» равно **FALSE**, то декодирование продолжается к следующему компоненту, в противном случае декодер ожидает кодирования значения факультативного компонента и обнаруживает ошибку кодирования, если его нет.

22.6 Определение альтернативы

22.6.1 Признаки кодирования, синтаксис и цель

22.6.1.1 При определении альтернативы используются следующие признаки кодирования:

```
&alternative-determination      AlternativeDetermination
                                DEFAULT field-to-be-set,
&alternative-reference          REFERENCE OPTIONAL,
&Encoder-transforms             #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms             #TRANSFORM ORDERED OPTIONAL,
&handle-id                      PrintableString
                                DEFAULT "default-handle",
&alternative-ordering           ENUMERATED {textual, tag}
                                DEFAULT textual
```

22.6.1.2 Для определения альтернативы должен использоваться следующий синтаксис:

```
ALTERNATIVE
        [DETERMINED BY &alternative-determination
                [HANDLE &handle-id]]
        [USING &alternative-reference
                [ORDER &alternative-ordering]
                [ENCODER-TRANSFORMS &Encoder-transforms]
                [DECODER-TRANSFORMS &Decoder-transforms]]
```

22.6.1.3 Определением типов, используемых для описания альтернативы, является.

```
AlternativeDetermination ::=
        ENUMERATED {field-to-be-set, field-to-be-used, handle} -- (см. 21.6)
```

22.6.1.4 Цель этой спецификации — описать правила, обеспечивающие декодеру возможность правильно определить, какой именно компонент класса кодирования в категории «альтернативы» был кодирован.

22.6.2 Ограничения на спецификацию

22.6.2.1 Если «**DETERMINED BY**» не присутствует, то подразумевается безусловное значение (по умолчанию) («**field-to-be-set**»).

22.6.2.2 «**HANDLE**» не указывается, если «**DETERMINED BY**» не равен «**handle**».

22.6.2.3 «**USING**» не указывается, если «**DETERMINED BY**» имеет значение «**handle**».

22.6.2.4 Если «**DETERMINED BY**» является «**handle**», применяется 21.6.6.

22.6.2.5 «**ENCODER-TRANSFORMS**» присутствует, если только «**DETERMINED BY**» установлен (возможно, по умолчанию) в «**field-to-be-set**». Первый преобразователь должен иметь источник, являющийся целым числом, а последний преобразователь должен иметь результат, который может кодироваться классом поля, указанного в «**USING**».

22.6.2.6 Спецификация ECN или приложение будут ошибочными, когда какой-либо преобразователь в «**ENCODER-TRANSFORMS**» не является обратимым для абстрактного значения, к которому он применен.

22.6.2.7 «**DECODER-TRANSFORMS**» присутствует, если только «**DETERMINED BY**» установлен в «**field-to-be-used**». Первый преобразователь должен иметь источник той же категории, что и поле, указанное в «**USING**» и не являющееся вспомогательным полем. Последний преобразователь должен иметь результат, являющийся целым числом.

22.6.2.8 Признак кодирования «**USING**», если имеется, будет ссылкой на поле, которое присутствует в кодировании раньше, чем кодирование альтернативы. Применение или спецификация ECN будут ошибочными, когда в экземпляре кодирования поле, указанное признаком кодирования «**USING**», затребовано декодером, но отсутствует (из-за применения факультативной возможности).

22.6.2.9 Эта спецификация считается установленной, если использовано ключевое слово «**ALTERNATIVE**», которое обязательно должно быть установлено во всех местах в определенном синтаксисе, где оно разрешено. Установка по умолчанию всех остальных частей этого определенного синтаксиса (например, использование одиночного «**ALTERNATIVE**») может не удовлетворять вышеприведенным ограничениям.

22.6.2.10 Если «**ORDER**» равен «**tag**», то каждая альтернатива начинается с класса кодирования в категории «тег». Номер тега, связанный с этим классом, называется компонент-тегом.

22.6.2.11 Компонент-теги всех альтернатив должны быть разными.

22.6.3 Действия кодера

22.6.3.1 Кодеры не будут генерировать кодовые последовательности, если не удовлетворяются условия 22.6.2.

22.6.3.2 Кодер определяет, какую альтернативу желает кодировать приложение, и создает смысловое значение целого числа «**alternative-index**» для указания на эту альтернативу.

22.6.3.3 Значение «**alternative-index**» будет равно нулю для первой альтернативы, единица — для следующей и т. д., причем порядок следования альтернатив определяет «**ORDER**».

22.6.3.4 Если «**ORDER**» равен «**textual**», то используется текстуальный порядок в спецификации типа ACH.1 или в определении структуры ECN. Если «**ORDER**» равен «**tag**», то порядок следования будет соответствовать номерам тегов в компонент-тегах (наименьший номер тега будет первым).

22.6.3.5 Если «**DETERMINED BY**» равен «**field-to-be-set**», то кодер применяет преобразователь, указанный в «**ENCODER-TRANSFORMS**» (если он есть), к смысловому значению «**alternative-index**» для создания значения, которое будет кодироваться в ссылке «**USING**».

Примечание — Кодирование ссылки «**USING**» в этом случае появляется в кодировании раньше, чем кодирование альтернативы, а кодер должен будет отложить кодирование этого поля до определения альтернативы, которая будет кодироваться.

22.6.3.6 Если «**DETERMINED BY**» равен «**field-to-be-used**», то кодер проверяет является ли значение в ссылке «**USING**», преобразованное с помощью «**DECODER-TRANSFORMS**» (если он есть), целочисленным значением, равным смысловому значению «**alternative-index**». Применение будет ошибочным, когда это условие не удовлетворяется, и кодирование не будет продолжаться.

22.6.3.7 Если «**DETERMINED BY**» равен «**handle**», то от кодера не требуется дальнейших действий.

22.6.4 Действия декодера

22.6.4.1 Декодер использует «**ORDER**», описанный в действиях кодера, для определения значения «**alternativeindex**», связанного с каждой альтернативой, и предполагает наличие кодирования соответствующей альтернативы, когда определено смысловое значение «**alternative-index**».

22.6.4.2 Если «**DETERMINED BY**» установлен (возможно, по умолчанию) в «**field-to-be-set**», то кодер восстанавливает значение «**alternative-index**» путем применения реверсии «**ENCODER-TRANSFORMS**» (если он есть) к значению ссылки «**USING**».

22.6.4.3 Если «**DETERMINED BY**» равен «**field-to-be-used**», то декодер восстанавливает смысловое значение «**alternative-index**» путем применения «**DECODER-TRANSFORMS**» (если он есть) к значению этого поля.

22.6.4.4 Если «**DETERMINED BY**» равен «**handle**», то декодер определяет значение идентификационного описателя. Это значение сравнивается со значением идентификационного описателя каждой из альтернатив. Если согласования нет, то декодер объявляет ошибку кодера. В остальных случаях смысловое значение «**alternative-index**» устанавливается в значение согласующейся альтернативы.

22.7 Спецификация пространства повторения

22.7.1 Признаки кодирования, синтаксис и цель

22.7.1.1 При спецификации пространства повторения используются следующие признаки кодирования:

```
        &repetition-space-size          EncodingSpaceSize
                                        DEFAULT self-delimiting-values,
        &repetition-space-unit Unit
                                        DEFAULT bit,
        &repetition-space-determination RepetitionSpaceDetermination
                                        DEFAULT field-to-be-set,
        &main-reference                 REFERENCE OPTIONAL,
        &Encoder-transforms             #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms                     #TRANSFORM ORDERED OPTIONAL,
            &handle-id PrintableString
                                        DEFAULT "default-handle",
            &termination-pattern        Non-Null-Pattern (ALL EXCEPT
different:any)                          DEFAULT bits '0'B
```

22.7.1.2 Для спецификации пространства повторения должен использоваться следующий синтаксис:

```
REPETITION-SPACE
    [SIZE &repetition-space-size
            [MULTIPLE OF &repetition-space-unit]]
    [DETERMINED BY &repetition-space-determination
            [HANDLE &handle-id]]
    [USING &main-reference
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
    [PATTERN &termination-pattern]
```

22.7.1.3 Определением типов, используемых в этой спецификации, является:

```
        EncodingSpaceSize ::= INTEGER
{encoder-option-with-determinant(-3),
                variable-with-determinant(-2),
                self-delimiting-values(-1),
                fixed-to-max(0)} (-3..MAX) -- (см. 21.2)
        Unit ::= INTEGER
                { repetitions(0), bit(1), nibble(4), octet(8), word16(16),
                  dword32(32)} (0..256) -- (см. 21.1)
        RepetitionSpaceDetermination ::= ENUMERATED
                { field-to-be-set, field-to-be-used, flag-to-be-set, flag-to-be-used,
                container, pattern, handle, not-needed} -- (см. 21.7)
```

Non-Null-Pattern ::= Pattern
(ALL EXCEPT (bits:"B | octets:"H | char8:"" | char16:"" |
char32:"")) -- (см. 21.10.2)

22.7.1.4 Цель этой спецификации — описать действия кодера и декодера, которые обеспечат декодеру возможность правильно определять конец пространства кодирования, занятого повторением.

Примечание — Реальное кодирование повторения не обязательно заполнит все пространство кодирования, поэтому восстановление кодирования повторения декодером будет обычно требовать действий, определяющих заполнение и выравнивание значения (см. 22.8).

22.7.1.5 Смысл признаков кодирования типа «**Unit**», «**EncodingSpaceSize**» и «**RepetitionSpace-Determination**» описан в 21.1, 21.2 и 21.7. Они вместе указывают способ определения конца пространства кодирования для повторений.

Примечание — Если спецификатор ECN требует, чтобы был введен определитель длины, то значение «**variable-withdeterminant**» в «**SIZE**» может быть указано даже при фиксированном размере пространства повторения.

22.7.1.6 Спецификация «**USING**» является ссылкой на вспомогательное поле, или на поле, переносящее абстрактные значения, или на контейнер в зависимости от значения «**DETERMINED BY**».

22.7.2 Ограничения на спецификацию

22.7.2.1 Если «**SIZE**» равен «**variable-with-determinant**», а «**DETERMINED BY**» не присутствует, то предполагается безусловное значение (по умолчанию) (**field-to-be-set**).

22.7.2.2 «**USING**» указывается, если, и только если, «**SIZE**» равен «**variable-with-determinant**», а «**DETERMINED BY**» равен «**field-to-be-set**», или «**field-to-be-used**», или «**flag-to-be-set**», или «**flag-to-be-used**», или «**container**».

22.7.2.3 «**ENCODER-TRANSFORMS**» присутствует, если только «**DETERMINED BY**» установлен (возможно, по умолчанию) в «**field-to-be-set**» или «**flag-to-be-set**». Первый преобразователь должен иметь источник, который является целым числом, если «**DETERMINED BY**» равен «**field-to-be-set**», или является булевым значением, если «**DETERMINED BY**» равен «**flag-to-be-set**». Последний преобразователь должен иметь результат, который может быть кодирован классом поля, указанного в «**USING**».

22.7.2.4 Спецификация ECN или применение будут ошибочными, когда какой-либо преобразователь в «**ENCODER-TRANSFORMS**» не является обратимым для абстрактного значения, к которому он применен.

22.7.2.5 «**DECODER-TRANSFORMS**» присутствует, если только «**DETERMINED BY**» установлен в «**field-to-beused**» или «**flag-to-be-used**». Первый преобразователь должен иметь источник той же категории, что и поле, указанное в «**USING**». Последний преобразователь должен иметь результат, который является целым числом, если «**DETERMINED BY**» равен «**field-to-be-used**», или является булевым значением, если «**DETERMINED BY**» равен «**flag-to-be-used**».

22.7.2.6 Признак кодирования «**USING**», если присутствует, при «**field-to-be-set**» или «**field-to-be-used**» должен быть ссылкой на поле, появившееся в кодировании раньше, чем кодируемое поле. Применение или спецификация ECN будут ошибочными, когда в экземпляре кодирования присутствует закодированное повторение, а поле, указанное признаком кодирования «**USING**», отсутствует (из-за применения факультативной возможности).

22.7.2.7 Признак кодирования «**USING**», если присутствует, при «**flag-to-be-set**» или «**flag-to-be-used**» должен быть ссылкой на поле, присутствующее в повторенном элементе повторения. Применение или спецификация ECN будут ошибочными, когда в экземпляре кодирования поле, указанное признаком кодирования «**USING**», отсутствует (из-за применения факультативной возможности) в каком-либо повторенном элементе.

Примечание — Требование присутствия указанного поля в элементе повторения будет удовлетворено, если имеется идентификатор, видимый согласно 17.5 (структура кодирования), 19.3 (отображение путем сопоставления полей), 19.6 (отображение путем распределения значений), или если он представлен текстуально в определении структуры замены, когда «**REPLACE COMPONENT**» используется объектом кодирования с классом в категории «повторение».

22.7.2.8 Если «**DETERMINED BY**» равен «**container**», то ссылка «**USING**» будет на конкатенацию или на повторение (либо на цепочку битов или цепочку октетов с вложенным типом), в которых кодируемое повторение является компонентом (или компонентом компонента на любую глубину). Применение

или спецификация ECN будут ошибочными, когда в экземпляре кодирования должны кодироваться последние элементы внутри одной и той же конкатенации или повторения.

22.7.2.9 «**HANDLE**» указывается, если только «**SIZE**» равен «**variable-with-determinant**», а «**DETERMINED BY**» равен «**handle**».

22.7.2.10 Если «**DETERMINED BY**» является «**handle**», применяется 21.7.10.

22.7.2.11 «**PATTERN**» указывается, если только «**SIZE**» равен «**variable-with-determinant**», а «**DETERMINED BY**» равен «**pattern**».

22.7.2.12 «**PATTERN**» не должен быть начальной субцепочкой в кодировании значения повторяемого элемента.

Примечание — Нет запрета на появление «**PATTERN**» внутри кодирования повторяемого элемента, который не является началом кодирования.

22.7.2.13 Эта спецификация считается установленной, если использовано ключевое слово «**REPETITION-SPACE**», которое обязательно должно быть установлено во всех местах в определенном синтаксисе, где оно разрешено. Установка по умолчанию всех остальных частей этого определенного синтаксиса (например, использование одиночного «**REPETITION-SPACE**») не может удовлетворять вышеприведенным ограничениям.

22.7.3 Действия кодера

22.7.3.1 Кодеры не будут генерировать кодовые последовательности, если не удовлетворяются условия 22.7.2.

22.7.3.2 Если «**SIZE**» равен положительному значению, то пространство кодирования будет кратно единицам «**MULTIPLE OF**». Если «**MULTIPLE OF**» является повторениями, то кодер прекращает кодирование, когда абстрактное значение, подлежащее кодированию, не является повторениями, и распознает ошибку спецификации или применения.

22.7.3.3 Если «**SIZE**» не установлен в положительное значение, то кодер определяет размер «s» пространства повторения в единицах «**MULTIPLE OF**» из спецификации кодирования значения. Это определение описано в подразделах о спецификации кодирования значения.

22.7.3.4 Если «**SIZE**» равен «**encoder-option-with-determinant**», то кодер (по своему выбору) может увеличить размер «s» (определенный в 22.7.3.3) на единицы «**MULTIPLE OF**» по сравнению с размером, определенным из спецификации кодирования значения, до любого значения, которое может быть кодировано в соответствующем определителе.

22.7.3.5 Если «**SIZE**» равен «**fixed-to-max**» или «**self-delimiting-values**», то дальнейшие действия кодера отсутствуют.

22.7.3.6 Если «**SIZE**» равен «**variable-with-determinant**», а «**DETERMINED BY**» равен «**container**», то дальнейшие действия кодера отсутствуют.

22.7.3.7 Если «**DETERMINED BY**» равен «**field-to-be-set**», то кодер применяет преобразователи, указанные в «**ENCODER-TRANSFORMS**» (если он есть), к значению «s» для получения значения, которое будет кодировано в ссылке «**USING**».

Примечание — Кодирование ссылки «**USING**» в этом случае появляется в кодировании раньше, чем кодирование повторения, а кодер должен будет отложить кодирование этого поля до определения повторения, которое будет кодироваться.

22.7.3.8 Если «**DETERMINED BY**» равен «**field-to-be-used**», то кодер проверяет, что значение в ссылке «**USING**», преобразованное с помощью «**DECODER-TRANSFORMS**» (если он есть), равно «s». Применение будет ошибочным, когда это условие не удовлетворяется, и кодирование не будет продолжаться.

22.7.3.9 Если «**DETERMINED BY**» равен «**flag-to-be-set**», то кодер применяет (для каждого повторяемого элемента) преобразователи, указанные в «**ENCODER-TRANSFORMS**» (если он есть), к булеву значению, которое равно TRUE для всех элементов, кроме последнего, и равно **FALSE** для последнего элемента. Результат из «**ENCODER-TRANSFORMS**» кодируется в ссылке «**USING**».

22.7.3.10 Если «**DETERMINED BY**»равен «**flag-to-be-used**», то кодер проверяет (для каждого повторяемого элемента), что значение в ссылке «**USING**», преобразованное с помощью «**DECODER-TRANSFORMS**» (если он есть), является булевым значением, которое равно **TRUE** для всех элементов, кроме последнего, и равно **FALSE** для последнего элемента. Применение будет ошибочным, когда это условие не удовлетворяется, и кодирование не будет продолжаться.

22.7.3.11 Если «**DETERMINED BY**» равен «**handle**», то от кодера не требуется дальнейших действий.

22.7.3.12 Если «**DETERMINED BY**» равен «**pattern**», то кодер проверяет, что указанная комбинация не является начальной субцепочкой какого-либо кодирования повторяемого элемента, прекращает кодирование, если проверка оказалась неудачной, и распознает ошибку спецификации или применения. Кодер добавляет комбинацию «**PATTERN**» в конец кодирования повторения.

22.7.4 Действия декодера

22.7.4.1 Если «**SIZE**» является положительным значением, то декодер определяет пространство кодирования, умножив это значение на единицу «**MULTIPLE OF**». Если «**MULTIPLE OF**» является повторениями, то реальный конец пространства повторения определяется путем декодирования и подсчета повторений.

22.7.4.2 Если «**SIZE**» не установлен в положительное значение, то кодер определяет размер «s» пространства повторения в единицах «**MULTIPLE OF**» из спецификации кодирования значения. Это определение описано в подразделах о спецификации кодирования значения.

22.7.4.3 Если «**SIZE**» равен «**variable-with-determinant**», а «**DETERMINED BY**» установлен в «**container**», то декодер использует конец контейнера, указанный в «**USING**», в качестве конца пространства кодирования.

22.7.4.4 Если «**SIZE**» равен «**variable-with-determinant**», а «**DETERMINED BY**» установлен (возможно, по умолчанию) в «**field-to-be-set**», то декодер восстанавливает значение «s» путем применения реверсии «**ENCODER-TRANSFORMS**» (если он есть) к значению ссылки «**USING**».

22.7.4.5 Если «**DETERMINED BY**» равен «**field-to-be-used**», то декодер восстанавливает значение «s» путем применения «**DECODER-TRANSFORMS**» (если он есть) к значению ссылки «**USING**».

22.7.4.6 Если «**DETERMINED BY**» равен «**flag-to-be-set**», то кодер восстанавливает булево значение путем применения реверсии «**ENCODER-TRANSFORMS**» (если он есть) к значению ссылки «**USING**».

22.7.4.7 Если «**DETERMINED BY**» равен «**flag-to-be-used**», то декодер восстанавливает булево значение путем применения «**DECODER-TRANSFORMS**» (если он есть) к значению ссылки «**USING**». Элемент будет последним в повторении, если, и только если, это булево значение равно FALSE.

22.7.4.8 Если «**DETERMINED BY**» равен «**handle**», то декодер определяет значение идентификационного описателя и пытается декодировать следующий элемент (параллельно) либо как следующее повторение, либо как последующий элемент, используя значение идентификационного описателя для различения этих вариантов. Если декодирование даст более одного результата или не даст ни одного, это будет ошибкой кодирования или спецификации.

22.7.4.9 Если «**DETERMINED BY**» равен «**pattern**», то декодер в начале декодирования каждого повторения будет проверять, присутствует ли «**PATTERN**». Если «**PATTERN**» присутствует, то биты комбинации сбрасываются, а повторение заканчивается.

22.8 Заполнение и выравнивание значения

22.8.1 Признаки кодирования, синтаксис и цель

22.8.1.1 При заполнении и выравнивании значения используются следующие признаки кодирования:

```
&value-justification              Justification DEFAULT right:0,
&value-pre-padding                Padding DEFAULT zero,
&value-pre-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
&value-post-padding               Padding DEFAULT zero,
&value-post-pattern               Non-Null-Pattern DEFAULT bits:'0'B,
&unused-bits-determination        UnusedBitsDetermination
                                  DEFAULT field-to-be-set,
&unused-bits-reference            REFERENCE OPTIONAL,
&Unused-bits-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,
&Unused-bits-decoder-transforms   #TRANSFORM ORDERED OPTIONAL
```

22.8.1.2 Для заполнения и выравнивания значения должен использоваться следующий синтаксис:

```
[VALUE-PADDING
        [JUSTIFIED &value-justification]
        [PRE-PADDING &value-pre-padding
                [PATTERN &value-pre-pattern]]
```

```
[POST-PADDING &value-post-padding
        [PATTERN &value-post-pattern]]
[UNUSED BITS
        [DETERMINED BY &unused-bits-determination]
        [USING &unused-bits-reference
            [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
            [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]
```

22.8.1.3 Определением типов, используемых в выравнивании, является.

```
Justification ::= CHOICE
                { left          INTEGER (0..MAX),
                  right         INTEGER (0..MAX)} -- (см. 21.8)
Padding ::= ENUMERATED {zero, one, pattern, encoder-option} -- (см. 21.9)
Pattern ::= CHOICE
        {bits                   BIT STRING,
        octets                  OCTET STRING,
        char8                   IA5String,
        char16                  BMPString,
          char32                UniversalString,
          any-of-length         INTEGER (1..MAX),
          different             ENUMERATED {any} }
Non-Null-Pattern ::= Pattern
        (ALL EXCEPT (bits:"B | octets:"H | char8:"" | char16:"" |
                                char32:"")) -- (см. 21.10)
UnusedBitsDetermination ::= ENUMERATED
        {field-to-be-set, field-to-be-used, not-needed} -- (см. 21.4)
```

22.8.1.4 Цель этой спецификации — описать способ, с помощью которого кодер помещает кодирование значения в пространство кодирования, а декодер получает возможность определять позицию этого кодирования значения.

22.8.1.5 Точное число битов, добавляемых кодером, зависит как от спецификации пространства кодирования, так и от спецификации кодирования значения, и указывается для каждого экземпляра кодирования значения.

22.8.1.6 «**USING**» является ссылкой (с правкой), которая дает возможность декодеру определять число введенных битов заполнения. Он является ссылкой на вспомогательное поле или поле, переносящее абстрактные значения, в зависимости от «**DETERMINED BY**».

22.8.2 Ограничения на спецификацию

22.8.2.1 Число битов, указанных в выравнивании, должно быть не больше общего числа «b» битов заполнения (см. ниже).

22.8.2.2 «**USING**» указывается, если, и только если, «**DETERMINED BY**» не равен «**non-needed**».

22.8.2.3 «**DENCODER-TRANSFORMS**» присутствует, если только «**DETERMINED BY**» установлен (возможно, по умолчанию) в «**field-to-be-set**». Первый преобразователь должен иметь источник, который является целым числом, а последний преобразователь должен иметь результат, который может кодироваться классом поля, указанного в «**USING**».

22.8.2.4 Спецификация ECN или применение будут ошибочными, когда какой-либо преобразователь в «**DENCODER-TRANSFORMS**» не является обратимым для абстрактного значения, к которому он приложен.

22.8.2.5 «**DECODER-TRANSFORMS**» присутствует, если только «**DETERMINED BY**» установлен в «**field-to-be-used**». Первый преобразователь должен иметь источник той же категории, что и поле, указанное в «**USING**» и не являющееся вспомогательным полем. Последний преобразователь должен иметь результат, который является целым числом.

22.8.2.6 Признак кодирования «**USING**», если присутствует, должен быть ссылкой на поле, появившееся в кодировании раньше, чем кодируемое поле. Применение или спецификация будут ошибочными, когда в экземпляре кодирования кодируемое поле присутствует, а поле, указанное признаком кодирования «**USING**», отсутствует (из-за применения факультативной возможности).

22.8.2.7 Эта спецификация считается установленной, если использовано ключевое слово «**VALUE-PADDING**». Если она не установлена, то действия указываются во всех местах, где разрешает синтаксис.

22.8.3 Действия кодера

22.8.3.1 Кодеры не будут генерировать кодовые последовательности, если не удовлетворяются условия 22.8.2.

22.8.3.2 Эта спецификация применяется, если, и только если, спецификация кодирования пространства кодирования или пространства повторения, вместе со спецификацией кодирования значения определяют, что можно добавлять биты заполнения вокруг кодовой последовательности значения или повторения в пределах пространства кодирования или повторения. Примем, что определенное число добавленных битов заполнения в экземпляре кодирования равно «b» («b» — больше или равно 0).

22.8.3.3 Если «**JUSTIFIED**» равен «**right:n**», то «b»-«n» битов добавляются в качестве предварительного заполнения перед кодированием значения или повторения, а «n» битов добавляются в качестве последующего заполнения после него.

22.8.3.4 Если «**JUSTIFIED**» равно «**left:n**», то «n» битов добавляются в качестве предварительного заполнения перед кодированием значения или повторения, а «b»-«n» битов добавляются в качестве последующего заполнения после него.

22.8.3.5 Биты заполнения устанавливаются согласно спецификациям «**PRE-PADDING**» и «**POST-PADDING**», причем начальный бит комбинации будет в каждом случае первым вводимым битом.

22.8.3.6 Если «**DETERMINED BY**» равен «**not-needed**», то это завершает действия кодеров.

22.8.3.7 Если «**DETERMINED BY**» равен «**field-to-be-set**», то кодер применяет преобразователи, указанные в «**ENCODER-TRANSFORMS**» (если он есть), к значению «b», чтобы получить значение, которое будет кодироваться в ссылке «**USING**».

Примечание — Кодирование ссылки «**USING**» в этом случае появляется в кодировании раньше, чем кодирование этого поля, а кодер должен будет отложить кодирование этого поля до определения значения, которое будет кодироваться путем кодирования этого поля.

22.8.3.8 Если «**DETERMINED BY**» равен «**field-to-be-used**», то кодер проверяет, что значение ссылки «**USING**», преобразованное с помощью «**DECODER-TRANSFORMS**» (если он есть), равно «b». Применение будет ошибочным, когда это условие не удовлетворяется, и кодирование не будет продолжаться.

22.8.4 Действия декодера

22.8.4.1 Если «**DETERMINED BY**» равен «**not-needed**», то декодер определяет значение «b» согласно спецификации кодирования значения и определения пространства кодирования или повторения.

22.8.4.2 Если «**DETERMINED BY**» установлен (возможно, по умолчанию) в «**field-to-be-set**», то декодер восстанавливает значение «b», применив реверсию «**ENCODER-TRANSFORMS**» (если он есть) к значению ссылки «**USING**».

22.8.4.3 Если «**DETERMINED BY**» равен «**field-to-be-used**», то декодер восстанавливает значение «b» путем применения «**DECODER-TRANSFORMS**» (если он есть) к значению этого поля.

22.8.4.4 Декодер должен использовать «**JUSTIFIED**» и значение «b» для определения позиции кодирования значения в пределах пространства кодирования и должен игнорировать значения всех битов заполнения.

22.9 Спецификация идентификационного описателя

22.9.1 Признаки кодирования, синтаксис и цель

22.9.1.1 При спецификации идентификационного описателя используются следующие признаки кодирования:

```
&exhibited-handle PrintableString     DEFAULT "default-handle",
&Handle-positions                     INTEGER (0..MAX) OPTIONAL,
&handle-value-set                     HandleValueSet DEFAULT tag:any
```

22.9.1.2 Для спецификации идентификационного описателя должен использоваться следующий синтаксис:

```
[EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
        [AS &handle-value-set]]
```

22.9.1.3 Определением типов, используемых в спецификации идентификационного описателя, является:

```
HandleValue ::= CHOICE {
        bits        BIT STRING,
        octets      OCTET STRING,
        number      INTEGER (0..MAX),
        tag         ENUMERATED {any}}
        range SEQUENCE {
                low INTEGER(0..MAX),
                high INTEGER(0..MAX) },
        ranges SET (SIZE(1..MAX)) OF SEQUENCE {
                low INTEGER(0..MAX),
                high INTEGER(0..MAX) }} -- (см. 21.16)
```

22.9.1.4 Цель данной спецификации — объявить, что объект кодирования демонстрирует идентификационный описатель, и указать его свойства, которыми являются:

a) имя описателя;

b) позиции битов, образующих описателя, и

c) возможные битовые паттерны (для битовых позиций, образующих описателя), появляющиеся в кодированиях, производимых этим объектом кодирования (набором значений описателя).

22.9.1.5 Список позиций в «AT» является позициями битов, формирующих идентификационный описатель в окончательном кодировании после того, как применено предварительное выравнивание и выполнены действия кодера по реверсии битов, кроме тех реверсий битов, которые вызываются спецификацией объекта кодирования в классе **#OUTER**.

Примечание — Это означает, что декодер должен выполнять любые реверсии битов, указанные в **#OUTER**, для целого PDU, а в других случаях рассматривать позиции битов и их значения без учета возможных реверсий битов, которые могут быть указаны для конкретных объектов кодирования.

22.9.1.6 Список позиций в «AT» является набором целочисленных значений (не обязательно непрерывным и не обязательно в возрастающем порядке по спецификации ECN). Эти позиции упорядочиваются кодерами и декодерами от нулевой позиции (первого бита в той части кодовой последовательности, которая образует описатель) и далее вверх, а биты этих позиций формируют смысловое поле описателя.

22.9.1.7 При значении «**number**» из «**HandleValueSet**» или кодировании номера тега бит в смысловом поле описателя, ближайший к нулевой позиции, будет битом старшего порядка, а «**number**» или номер тега, указывающий «**HandleValueSet**», выравнивается вправо внутри этого поля. Если «**number**» или номер тега слишком велик для этого поля, то возникает ошибка спецификации ECN.

22.9.1.8 Если используются альтернативы «**bitstring**» или «**octetstring**» из «**HandleValueSet**», то их значения должны иметь то же число битов, что и указанное в «AT» для идентификационного описателя. Бит в смысловом поле описателя, ближайший к нулевой позиции, будет начальным битом в «**bitstring**» или «**octetstring**», которые указывают «**HandleValueSet**».

22.9.1.9 «**HandleValueSet**» не указывается в виде «**tag:any**», если спецификация не относится к объекту кодирования класса **#TAG**. В этом случае значение идентификационного описателя определяется либо номером тега в спецификации ECN, либо номером тега, отображенным из тега ACH.1 (как описано в разделе 19), и не должно указываться с помощью «**HandleValueSet**». Если, однако, значение указано в «**HandleValueSet**» и отличается от того, которое присвоено в спецификации ECN для класса тега или в теге ACH.1, который отображается в тег ECN, то возникает ошибка спецификации ECN.

22.9.2 Ограничения на спецификацию

22.9.2.1 В любом применении ECN все идентификационные описатели с одним и тем же именем должны указывать одинаковый набор битовых позиций.

Примечание — Отсутствует общее требование, чтобы наборы значений описателей разных объектов кодирования, определяемые в спецификации ECN, были непересекающимися, но непересекающиеся наборы значений описателей требуются, когда идентификационный описатель используется для различения факультативных возможностей выбора альтернатив, окончания повторения или упорядочивания набора (см. 21.5.7, 21.6.6, 21.7.10 и 22.10.2.1).

22.9.2.2 Для объекта кодирования, показывающего идентификационный описатель (с заданным набором значений описателя), значение идентификационного описателя, появляющееся при каждом возможном кодировании, произведенном объектом кодирования (для всех возможных абстрактных значений), должно быть членом заданного набора значений описателя.

22.9.2.3 Все объекты кодирования, которые показывают один и тот же идентификационный описатель, либо не должны иметь спецификации предварительного выравнивания, либо должны выравнивать на одинаковый блок предварительного выравнивания.

Примечание — Это ограничение введено для того, чтобы декодеры могли сдвигаться к позиции выравнивания перед поиском описателя, когда декодирование зависит от значения описателя.

22.9.2.4 Эта спецификация считается установленной, если использовано ключевое слово «**EXHIBITS-HANDLE**». Если она не установлена, то идентификационный описатель не будет показываться.

22.9.3 Действия кодера

22.9.3.1 Если объект кодирования показывает идентификационный описатель, то кодер проверяет, является ли значение идентификационного описателя, появляющееся в производимой кодировке, членом заданного набора значений описателя, а в противном случае распознает ошибку спецификации или применения.

22.9.4 Действия декодера

22.9.4.1 Отсутствуют действия декодера, прямо зависящие от наличия идентификационного описателя. Действия декодера зависят только от использования идентификационного описателя для обнаружения факультативных возможностей, конца повторения или выбора альтернатив.

22.10 Спецификация конкатенации

22.10.1 Признаки кодирования, синтаксис и цель

22.10.1.1 При спецификации конкатенации используются следующие признаки кодирования:

&concatenation-order	**ENUMERATED {textual, tag, random}** **DEFAULT textual,**
&concatenation-alignment	**ENUMERATED {none, aligned}** **DEFAULT aligned,**
&concatenation-handle	**PrintableString** **DEFAULT "default-handle"**

22.10.1.2 Для спецификации конкатенации должен использоваться следующий синтаксис:

[CONCATENATION
[ORDER &concatenation-order]
[ALIGNMENT &concatenation-alignment]
[HANDLE &concatenation-handle]]

22.10.1.3 Эта спецификация определяет порядок следования, в котором кодируются компоненты класса кодирования в категории «конкатенация», средства, используемые кодером для идентификации каждого компонента, и заполнение предварительного выравнивания, которое следует обеспечивать между компонентами.

22.10.2 Ограничения на спецификацию

22.10.2.1 Если «**ORDER**» является «**random**», то предполагается, что «**HANDLE**», когда не установлен, имеет значение по умолчанию «**default-handle**», а объекты кодирования, применяемые ко всем компонентам, должны показывать этот идентификационный описатель. Наборы значений описателей этих объектов кодирования должны быть непересекающимися.

22.10.2.2 Если «**ALIGNMENT**» равен «**aligned**», то предполагается, что спецификация предварительного выравнивания, когда не установлена, имеет безусловное значение (по умолчанию).

22.10.2.3 Если компонент имеет собственное явное предварительное выравнивание, то применяется после любого предварительного выравнивания компонента, вызванного установкой «**ALIGNMENT**» в классе кодирования категории «конкатенация».

Примечание — Эквивалентная функция не обеспечивается для повторений, так как она может быть достигнута проще, путем предварительного выравнивания одиночного компонента.

22.10.2.4 Если «**ORDER**» равен «**tag**», то каждый компонент должен начинаться с класса кодирования в категории «тег». Номер тега, связанный с этим классом, называется компонент-тегом.

22.10.2.5 Компонент-теги каждой альтернативы должны быть разными.

22.10.2.6 Эта спецификация считается установленной, если использовано ключевое слово «**CONCATENATION**». Если она не установлена, то кодеры и декодеры действуют так, как будто она была установлена с каждым признаком кодирования с учетом ее безусловного значения (по умолчанию).

22.10.2.7 Если (из-за применения факультативных возможностей) имеется по меньшей мере одно абстрактное значение конкатенации, которое не имеет битов в своем кодировании, то конкатенация не должна иметь предварительного выравнивания.

Примечание — Этот пункт будет применяться в случаях, когда конкатенация не имеет обязательных компонентов или когда все и обязательные компоненты не могут иметь (из-за применения факультативных возможностей) битов в своих кодированиях.

22.10.3 Действия кодера

22.10.3.1 Если «**ORDER**» равен «**textual**», то используется текстуальный порядок следования из спецификации типов ACH.1 или определения структуры ECN.

22.10.3.2 Если «**ORDER**» равен «**tag**», то порядок следования определяется номерами тегов в компонент-тегах (первым будет самый малый номер тега).

22.10.3.3 Если «**ORDER**» равен «**random**», то кодер определяет порядок следования конкатенации без ограничения.

22.10.3.4 Если «**ALIGNMENT**» равен «**none**», то кодер соединяет кодирования компонентов без вставляемых битов.

22.10.3.5 Если «**ALIGNMENT**» равен «**aligned**», то кодер применяет спецификацию предварительного выравнивания для класса в категории «конкатенация» до кодирования каждого компонента, кроме случая, когда спецификация предварительного выравнивания «**ALIGNED TO ANY**» интерпретируется как спецификация «**ALIGNED TO NEXT**» (см. 22.2).

Примечания

1 Это объясняется тем, что может быть только один начальный указатель для «**ALIGNED TO ANY**».

2 Любое предварительное выравнивание, указанное для компонента (включая «**ALIGNED TO ANY**»), применяется после вышеуказанных действий.

22.10.4 Действия декодера

22.10.4.1 При декодировании компонента декодер сначала выполняет действия декодера, связанные со спецификацией предварительного выравнивания для «**ALIGNMENT**», если он установлен в «**aligned**», обрабатывая «**ALIGNED TO ANY**», как «**ALIGNED TO NEXT**» (см. 22.2). Если «**ALIGNMENT**» установлен в «**none**», то декодер приступает прямо к декодированию компонента.

22.10.4.2 Декодер определяет порядок компонентов из определенного порядка для кодера, если «**ORDER**» равен «**textual**» или «**tag**».

22.10.4.3 Если «**ORDER**» равен «**random**», то декодер определяет порядок компонентов путем проверки значения идентификационного описателя.

22.10.4.4 Декодирование продолжается до получения абстрактного значения для каждого компонента, а декодер будет распознавать ошибку кодера, когда для компонента указано более одной кодовой последовательности или когда во время декодирования появились неожидаемые значения для идентификационных описателей.

Примечание — Неожидаемые значения могут появиться в виде части обеспечения расширяемости, но это не поддерживается условиями настоящего стандарта, поэтому такие случаи считаются ошибками кодера.

22.11 Спецификация кодирования вложенного типа

22.11.1 Признаки кодирования, синтаксис и цель

22.11.1.1 При спецификации кодирования вложенного типа используются следующие признаки кодирования:

```
&Primary-encoding-object-set      #ENCODINGS OPTIONAL,
&Secondary-encoding-object-set    #ENCODINGS OPTIONAL,
&over-ride-encoded-by             BOOLEAN DEFAULT FALSE
```

22.11.1.2 Для спецификации кодирования вложенного типа должен использоваться следующий синтаксис:

```
[CONTENTS-ENCODING &Primary-encoding-object-set
                  [COMPLETED BY &Secondary-encoding-object-set]
                  [OVERRIDE &over-ride-encoded-by]]
```

22.11.1.3 Цель спецификации — определить кодирования вложенного типа и необходимости отмены связанного с таким вложенным типом ограничения на содержимое **ENCODED BY** ACH.1.

22.11.1.4 Эта спецификация обеспечивает либо один набор объектов кодирования, либо два таких набора. Если обеспечиваются два, то они объединяются согласно 13.2, чтобы образовать комбинированный набор объектов кодирования.

22.11.1.5 Эта спецификация считается установленной, если использовано ключевое слово «**CONTENTS-ENCODING**».

22.11.2 Действия кодера

22.11.2.1 Если «**CONTENTS-ENCODING**» не установлен, то вложенный тип кодируется с использованием комбинированного набора объектов кодирования, примененного к контейнеру, когда **ENCODED BY** не присутствует в ограничении содержимого ACH.1, а в противном случае — с помощью правил кодирования, указанных оператором **ENCODED BY**.

22.11.2.2 Если «**CONTENTS-ENCODING**» установлен, то к вложенному типу применяется комбинированный набор объектов кодирования, сформированный из «**COMPLETED BY**», когда **ENCODED BY** не присутствует в ограничении содержимого ACH.1 либо когда **ENCODED BY** присутствует, а «**OVERIDE**» равен **TRUE**. В остальных случаях комбинированный набор кодирования, примененный к объемлющему типу, применяется к вложенному типу.

22.11.3 Действия декодера

22.11.3.1 Декодер декодирует вложенный тип согласно кодированию, примененному кодером, как описано выше.

22.12 Спецификация реверсии битов

22.12.1 Признаки кодирования, синтаксис и цель

22.12.1.1 При спецификации реверсии битов используется следующий признак кодирования:

```
&bit-reversal ReversalSpecification
              DEFAULT no-reversal
```

22.12.1.2 Для спецификации реверсии битов должен использоваться следующий синтаксис:

```
[BIT-REVERSAL &bit-reversal]
```

22.12.1.3 Определением типов, используемых в этой группе, является:

```
ReversalSpecification ::= ENUMERATED
        {no-reversal,
        reverse-bits-in-units,
        reverse-half-units,
        reverse-bits-in-half-units} -- (см. 21.14)
```

22.12.1.4 Цель этой спецификации — создать возможности, чтобы порядок битов в окончательной кодовой комбинации отличался от битов, генерированных как часть пространства кодирования или пространства повторения, либо от битов в полном кодировании PDU (см. раздел 25).

Примечания

1 Реверсия битов может указываться для кодирований индивидуальных битовых полей, а также для результатов конкатенации или повторения. Следует обращать внимание на обеспечение того, чтобы одна реверсия не была обратна другой.

2 Реверсия битов применяется к содержимому пространства кодирования или пространства повторения (включая любое предварительное выравнивание и последующее выравнивание значения), но не применяется к заполнению предварительного выравнивания.

22.12.2 Ограничения на спецификацию

22.12.2.1 Эта спецификация доступна только в случаях, когда затребовано кодирование пространства кодирования или пространства повторения, а также внутри **#OUTER**.

22.12.2.2 «**BIT-REVERSAL**» не должен быть «**reverse-half-units**» или «**reverse-bits-in-half-units**», если «**MULTIPLE OF**» не установлен в четное число битов для пространства кодирования или пространства повторения либо если реверсируется **#OUTER** (это требование означает, что значение «повторения» для «**MULTIPLE OF**» в таком случае не разрешается).

22.12.2.3 «**BIT-REVERSAL**» не должен устанавливаться, если «**MULTIPLE OF**» не является повторениями или не превышает одного бита.

22.12.2.4 Эта спецификация считается установленной, если использовано ключевое слово «**BIT-REVERSAL**». Если она не установлена, то кодеры и декодеры действуют так, как будто она была установлена с признаком кодирования, имеющим его безусловное значение (по умолчанию).

22.12.3 Действия кодера

22.12.3.1 За исключением выполнения действий **#OUTER** кодер разделяет содержимое пространства кодирования или пространства повторения на единицы «**MULTIPLE OF**», если «**MULTIPLE OF**» не является «**repetitions**». Если «**MULTIPLE OF**» равен «**repetitions**», то все пространство кодирования рассматривается как единый блок. Когда выполняется реверсия битов для **#OUTER**, все кодирование (после применения «**PADDING**») разделяется на единицы «**MULTIPLE OF**». Спецификация ECN будет ошибочной, когда полное кодирование не кратно единице «**MULTIPLE OF**» целое число раз.

22.12.3.2 Кодер не выполняет реверсии (значение по умолчанию), или реверсирует биты в каждой единице, или реверсирует половины единицы (не изменяя порядок следования битов в каждой половине единицы), или реверсирует биты внутри каждой половины единицы в зависимости от значения «**BIT-REVERSAL**».

22.12.4 Действия декодера

22.12.4.1 Декодер вначале определяет (см. спецификацию пространства кодирования или пространства повторения) конец пространства кодирования или пространства повторения, или (для спецификации реверсии битов внутри **#OUTER**) конец всего кодирования, а затем выполняет действия реверсии, указанные для кодера, перед продолжением декодирования.

Примечание — Выполнение той же реверсии восстановит исходный порядок следования битов.

23 Спецификация определенного синтаксиса для классов «битовое поле» и «конструктор»

В настоящем разделе приведен полный синтаксис для определения объектов кодирования каждого класса кодирования в различных категориях.

Примечание — Действия кодера и декодера описываются в последующих разделах при условии, что установлена группа признаков кодирования. Группа установлена, если, и только если, в спецификации объекта кодирования присутствует начальное ключевое слово этой группы.

23.1 Определение объектов кодирования для классов в категории «альтернативы»

23.1.1 Определенный синтаксис

Синтаксис для определения объектов кодирования для классов в категории «альтернативы» определяется следующим образом:

```
#ALTERNATIVES ::= ENCODING-CLASS {
    -- Спецификация замены структуры или компонента (см. 22.1)
    &#Replacement-structure                                    OPTIONAL,
    &replacement-structure-encoding-object &#Replacement-structure OPTIONAL,

    -- Спецификация предварительного выравнивания и заполнения (см. 22.2)
    &encoding-space-pre-alignment-unit    Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding           Padding DEFAULT zero,
    &encoding-space-pre-pattern           Non-Null-Pattern (ALL EXCEPT different:any)
                                          DEFAULT bits:'0'B,
```

```
        -- Спецификация начального указателя (см. 22.3)
        &start-pointer                          REFERENCE OPTIONAL,
        &start-pointer-unit Unit                (ALL EXCEPT repetitions) DEFAULT bit,
        &Start-pointer-encoder-transforms       #TRANSFORM ORDERED OPTIONAL,

        -- Определение альтернативы (см. 22.6)
        &alternative-determination              AlternativeDetermination
                                                DEFAULT field-to-be-set,
        &alternative-reference                  REFERENCE OPTIONAL,
        &Encoder-transforms                     #TRANSFORM ORDERED OPTIONAL,
        &Decoder-transforms                     #TRANSFORM ORDERED OPTIONAL,
        &handle-id PrintableString
                                                DEFAULT "default-handle",
        &alternative-ordering                   ENUMERATED {textual, tag}
                                                DEFAULT textual,

        -- Спецификация идентификационного описателя (см. 22.9)
        &exhibited-handle                       PrintableString DEFAULT "default-handle",
        &Handle-positions                       INTEGER (0..MAX) OPTIONAL,
        &handle-value-set                       HandleValueSet DEFAULT tag:any
    } WITH SYNTAX {
            [REPLACE
                [STRUCTURE]
                WITH &#Replacement-structure
                        [ENCODED BY &replacement-structure-encoding-object]]
      [ALIGNED TO
                [NEXT]
                [ANY]
                        &encoding-space-pre-alignment-unit
                        [PADDING &encoding-space-pre-padding
                                [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER &start-pointer
                [MULTIPLE OF &start-pointer-unit]
                [ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
      ALTERNATIVE
                [DETERMINED BY &alternative-determination
                        [HANDLE &handle-id]]
                [USING &alternative-reference
                        [ORDER &alternative-ordering]
                        [ENCODER-TRANSFORMS &Encoder-transforms]
                        [DECODER-TRANSFORMS &Decoder-transforms]]
      [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
                [AS &handle-value]]
}
```

23.1.2 Цель и ограничения

23.1.2.1 Этот синтаксис используется с целью определения начала пространства кодирования для класса кодирования в категории «альтернатива», определения кодированной альтернативы и факультативного объявления о том, что объект кодирования показывает определенный идентификационный описатель (с заданным набором значений описателя).

23.1.2.2 Если «**REPLACE STRUCTURE**» установлен, то другие группы признаков кодирования не устанавливаются. Если объект кодирования структуры замены показывает описатель (с заданным набором значений описателя), определяемый объект кодирования показывает тот же идентификационный описатель (с тем же набором значений описателя — см. 22.1.1.11).

23.1.2.3 Объект кодирования этого класса не показывает идентификационный описатель, если «**EXHIBITS HANDLE**» не установлен (даже если все компоненты описанной конструкции показывают

идентификационные описатели), или пока «**REPLACE STRUCTURE**» не установлен и объект кодирования структуры замены не показывает идентификационный описатель (см. 22.1.1.11).

23.1.2.4 Если «**EXHIBITS HANDLE**» установлен, то объект кодирования показывает заданный идентификационный описатель.

Примечание — При этом, как правило, потребуется, чтобы каждый компонент имел «**EXHIBITS HANDLE**», установленный в одно и то же значение, за исключением вставки головного узла, показывающей идентификационный описатель (см. 9.10.3).

23.1.3 Действия кодера

23.1.3.1 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) замена;

b) предварительное выравнивание и заполнение;

c) начальный указатель;

d) определение альтернативы;

e) идентификационный описатель.

23.1.4 Действия декодера

23.1.4.1 Для любой установленной группы признаков кодирования декодер выполняет декодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) определение альтернативы.

23.2 Определение объектов кодирования для классов в категории «цепочка битов»

23.2.1 Определенный синтаксис

Синтаксис для определения объектов кодирования для классов в категории «цепочка битов» определяется следующим образом:

```
#BITS ::= ENCODING-CLASS {
    -- Спецификация предварительного выравнивания и заполнения (см. 22.2)
    &encoding-space-pre-alignment-unit    Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding           Padding DEFAULT zero,
    &encoding-space-pre-pattern           Non-Null-Pattern (ALL EXCEPT different:any)
                                          DEFAULT bits:'0'B,

    -- Спецификация начального указателя (см. 22.3)
    &start-pointer                        REFERENCE OPTIONAL,
    &start-pointer-unit Unit              (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms     #TRANSFORM ORDERED OPTIONAL,

    -- кодирование значения битов
    &value-reversal                       BOOLEAN DEFAULT FALSE,
    &Transforms                           #TRANSFORM ORDERED OPTIONAL,
    &Bits-repetition-encodings            #CONDITIONAL-REPETITION ORDERED OPTIONAL,
    &bits-repetition-encoding             #CONDITIONAL-REPETITION OPTIONAL,

    -- Спецификация идентификационного описателя (см. 22.9)
    &exhibited-handle                     PrintableString DEFAULT "default-handle",
    &Handle-positions                     INTEGER (0..MAX) OPTIONAL,
    &handle-value-set                     HandleValueSet DEFAULT tag:any,

    -- Спецификация кодирования вложенного типа (см. 22.11)
    &Primary-encoding-object-set          #ENCODINGS OPTIONAL,
    &Secondary-encoding-object-set        #ENCODINGS OPTIONAL,
```

```
        &over-ride-encoded-by                    BOOLEAN DEFAULT FALSE
  } WITH SYNTAX {
        [ALIGNED TO
              [NEXT]
          [ANY]
              &encoding-space-pre-alignment-unit
              [PADDING &encoding-space-pre-padding
                    [PATTERN &encoding-space-pre-pattern]]]
  [START-POINTER &start-pointer
          [MULTIPLE OF &start-pointer-unit]
          [ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
  [VALUE-REVERSAL &value-reversal]
  [TRANSFORMS &Transforms]
  [REPETITION-ENCODINGS &Bits-repetition-encodings]
  [REPETITION-ENCODING &bits-repetition-encoding]
  [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
        [AS &handle-value]]
  [CONTENTS-ENCODING &Primary-encoding-object-set
                    [COMPLETED BY &Secondary-encoding-object-set]
                    [OVERRIDE &over-ride-encoded-by]]
}
```

23.2.2 Модель для кодирования классов в категории «цепочка битов»

23.2.2.1 Модель кодирований битов содержит следующее:

a) порядок следования битов в цепочке битов может быть реверсирован;

b) биты рассматриваются затем как повторение бита;

c) имеется факультативное преобразование (указанное в «**TRANSFORMS**»), при котором каждый бит преобразуется в цепочку битов (саморазграничивающую);

d) «**REPETITION-ENCODING**» или «**REPETITION-ENCODINGS**» указывает, как следует кодировать повторение последовательностей битов (или исходных битов, если «**TRANSFORMS**» не установлен).

Примечание — Единственной целью разрешения «**REPETITION-ENCODING**» так же, как «**REPETITION-ENCODINGS**», является обеспечение синтаксиса, который не содержит двойных фигурных скобок («{{») в общем случае одиночного условного кодирования. Использование «**REPETITION-ENCODINGS**», когда имеется одиночное условное кодирование, не одобряется, но разрешается.

23.2.2.2 Границы (если присутствуют) для кодируемого класса (класса в категории «цепочка битов») являются границами числа битов в цепочке битов, формирующей каждое абстрактное значение.

23.2.2.3 При рассмотрении повторения битов эти границы считаются границами числа повторений и могут использоваться в спецификации тех объектов кодирования класса **#CONDITIONAL-REPETITION**, которые применены в спецификации этого объекта кодирования.

23.2.3 Цель и ограничения

23.2.3.1 Этот синтаксис используется с целью определения начала пространства кодирования для класса в категории «цепочка битов», для кодирования абстрактных значений этого класса, для факультативного объявления о том, что объект кодирования показывает определенный идентификационный описатель (с заданным набором значений описателя), и для спецификации способа кодирования вложенного типа.

23.2.3.2 **#CONDITIONAL-REPETITION**, который применяется этим объектом кодирования, не должен указывать «**REPLACE**», если он не равен «**REPLACE STRUCTURE**».

23.2.3.3 Если какой-либо объект кодирования **#CONDITIONAL-REPETITION** содержит раздел «**REPLACE STRUCTURE**», то все объекты кодирования **#CONDITIONAL-REPETITION** должны содержать раздел «**REPLACE STRUCTURE**».

23.2.3.4 Если в объектах кодирования **#CONDITIONAL-REPETITION** имеется раздел «**REPLACE STRUCTURE**», то остальные параметры не устанавливаются. Если объект кодирования структуры замены показывает описатель (с заданным набором значений описателя), определяемый объект кодирования показывает тот же идентификационный описатель (с тем же набором значений описателя — см. 22.1.1.11).

23.2.3.5 Первый преобразователь в «**TRANSFORMS**» (если он есть) должен иметь источник, который является одиночным битом, а последний преобразователь должен иметь результат, который является цепочкой битов. Цепочки битов, образованные для одного бита и для нуля битов, должны формировать саморазграничивающий набор (см. 3.2.42).

Примечание — Это означает, что окончательное преобразование должно быть саморазграничивающим.

23.2.3.6 Спецификация ECN или применение будут ошибочными, когда какой-либо преобразователь в «**TRANSFORMS**» не является обратимым для абстрактного значения, к которому он применен.

23.2.3.7 Должен устанавливаться только один из «**REPETITION-ENCODING**» и «**REPETITION-ENCODINGS**».

23.2.3.8 Если объект кодирования в упорядоченном списке «**REPETITION-ENCODINGS**» определен с помощью «**IF**» или «**IF-ALL**», то все предыдущие объекты кодирования в этом списке должны быть определены с помощью «**IF**» или «**IF-ALL**».

23.2.3.9 Если «**DETERMINED BY**» равен «**not-needed**» в одной или в нескольких спецификациях «**REPETITION-ENCODING(S)**», то абстрактные значения исходной цепочки битов, к которой этот объект кодирования применен, должны быть ограничены до конечного саморазграничивающего набора, который может быть определен из спецификации ECN.

Примечание — Это может быть случаем, когда значения цепочки битов, полученные при кодировании Хаффмана (см. приложение Е), указаны путем отображения целочисленных значений в биты (см. 19.7) или когда значения цепочки битов имеют видимую в ECN границу, которая ограничивает их до фиксированного числа битов.

23.2.3.10 Если «**EXHIBITS HANDLE**» установлен, то объект кодирования показывает определенный идентификационный описатель.

Примечание — При этом, как правило, потребуются ограничения на абстрактные значения связанного типа, или добавления избыточных битов при преобразовании в биты, или то и другое.

23.2.3.11 Если «**EXHIBITS HANDLE**» установлен, то «**ALIGNED TO**» не должен устанавливаться во всех спецификациях «**REPETITION-ENCODING(S)**».

23.2.4 Действия кодера

23.2.4.1 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) кодирование значений битов (см. 23.2.4.2);

d) идентификационный описатель;

e) кодирование вложенного типа.

23.2.4.2 Для кодирования значения битов кодер выполняет следующее:

a) реверсирует порядок следования битов в абстрактном значении целой цепочки битов, если «**VALUEREVERSAL**» установлен в **TRUE**;

b) рассматривает значение цепочки битов как повторение битов;

c) применяет указанный «**TRANSFORMS**» (если он есть) к каждому биту для образования повторения битов;

d) кодирует повторение путем применения первого «**REPETITION-ENCODING(S)**», условие которого удовлетворяется.

23.2.4.3 Спецификация ECN будет ошибочной, если отсутствует «**REPETITION-ENCODING(S)**», условие которого удовлетворяется.

23.2.5 Действия декодера

23.2.5.1 Для любой установленной группы признаков кодирования декодер выполняет декодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) декодирование значения битов (см. 23.2.5.2);

d) декодирование вложенного типа.

23.2.5.2 При декодировании значения битов декодер использует «**REPETITION-ENCODING(S)**» для определения пространства повторения и восстановления исходного порядка битов с помощью спецификации «**BIT-REVERSAL**».

23.2.5.3 Если «**TRANSFORMS**» установлен, то декодер использует признак саморазграничения в кодировании каждого бита для определения конца повторения и реверсирует преобразователи для восстановления исходного значения цепочки битов.

23.2.5.4 Если «**VALUE-REVERSAL**» установлен в **TRUE**, то окончательный порядок следования битов в абстрактном значении цепочки битов реверсируется.

23.3 Определение объектов кодирования для классов в категории «булева»

23.3.1 Определенный синтаксис

Синтаксис для определения объектов кодирования для классов в категории «булева» определяется следующим образом:

```
#BOOL ::= ENCODING-CLASS {

    -- Спецификация замены исключительно структуры (см. 22.1)
    &#Replacement-structure
            OPTIONAL,
    &replacement-structure-encoding-object &#Replacement-structure OPTIONAL,

    -- Спецификация предварительного выравнивания и заполнения (см. 22.2)
    &encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding        Padding DEFAULT zero,
    &encoding-space-pre-pattern        Non-Null-Pattern (ALL EXCEPT different:any)
                                       DEFAULT bits:'0'B,

    -- Спецификация начального указателя (см. 22.3)
    &start-pointer                     REFERENCE OPTIONAL,
    &start-pointer-unit                Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms  #TRANSFORM ORDERED OPTIONAL,

    -- Спецификация пространства кодирования (см. 22.4)
    &encoding-space-size               EncodingSpaceSize
                                       DEFAULT self-delimiting-values,
    &encoding-space-unit               Unit (ALL EXCEPT repetitions)
                                       DEFAULT bit,
    &encoding-space-determination      EncodingSpaceDetermination
                                       DEFAULT field-to-be-set,
    &encoding-space-reference          REFERENCE OPTIONAL,
    &Encoder-transforms                #TRANSFORM ORDERED OPTIONAL,
    &Decoder-transforms                #TRANSFORM ORDERED OPTIONAL,

    -- Кодирование
    &value-true-pattern                Pattern DEFAULT bits:'1'B,
    &value-false-pattern               Pattern DEFAULT bits:'0'B,

    -- Заполнение и выравнивание значения (см. 22.8)
    &value-justification               Justification DEFAULT right:0,
    &value-pre-padding                 Padding DEFAULT zero,
    &value-pre-pattern                 Non-Null-Pattern DEFAULT bits:'0'B,
    &value-post-padding                Padding DEFAULT zero,
    &value-post-pattern                Non-Null-Pattern DEFAULT bits:'0'B,
    &unused-bits-determination         UnusedBitsDetermination
                                       DEFAULT field-to-be-set,
    &unused-bits-reference             REFERENCE OPTIONAL,
    &Unused-bits-encoder-transforms    #TRANSFORM ORDERED OPTIONAL,
    &Unused-bits-decoder-transforms    #TRANSFORM ORDERED OPTIONAL,
```

```
        -- Спецификация идентификационного описателя (см. 22.9)
        &exhibited-handle                PrintableString DEFAULT "default-handle",
        &Handle-positions                INTEGER (0..MAX) OPTIONAL,
        &handle-value-set                HandleValueSet DEFAULT tag:any,

        -- Спецификация реверсии битов (см. 22.12)
        &bit-reversal        ReversalSpecification
                             DEFAULT no-reversal

} WITH SYNTAX {
        [REPLACE
                [STRUCTURE]
                WITH &#Replacement-structure
                        [ENCODED BY &replacement-structure-encoding-object]]
        [ALIGNED TO
                [NEXT]
                [ANY]
                        &encoding-space-pre-alignment-unit
                        [PADDING &encoding-space-pre-padding
                                [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER &start-pointer
                [MULTIPLE OF &start-pointer-unit]
                [ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
        ENCODING-SPACE
                [SIZE &encoding-space-size
                        [MULTIPLE OF &encoding-space-unit]]
                [DETERMINED BY &encoding-space-determination]
                [USING &encoding-space-reference
                        [ENCODER-TRANSFORMS &Encoder-transforms]
                        [DECODER-TRANSFORMS &Decoder-transforms]]
        [TRUE-PATTERN &value-true-pattern]
        [FALSE-PATTERN &value-false-pattern]
        [VALUE-PADDING
                [JUSTIFIED &value-justification]
                [PRE-PADDING &value-pre-padding
                        [PATTERN &value-pre-pattern]]
                [POST-PADDING &value-post-padding
                        [PATTERN &value-post-pattern]]
                [UNUSED BITS
                        [DETERMINED BY &unused-bits-determination]
                        [USING &unused-bits-reference
                        [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                        [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]
        [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
                        [AS &handle-value]]
                [BIT-REVERSAL &bit-reversal]
}
```

23.3.2 Цель и ограничения

23.3.2.1 Этот синтаксис используется с целью определения начала пространства кодирования для класса в категории «булева», для кодирования абстрактных значений этого класса, для их размещения внутри пространства кодирования, для факультативного объявления о том, что объект кодирования показывает определенный идентификационный описатель (с заданным набором значений описателя), и для возможной реверсии битов в пространстве кодирования для булевых значений.

23.3.2.2 Если «**REPLACE**» установлен, то никакие другие группы признаков кодирования не устанавливаются.

23.3.2.3 Один самый больший из «**TRUE-PATTERN**» и «**FALSE-PATTERN**» должен быть установлен в «**different:any**».

23.3.2.4 Если альтернатива «**any-of-length**» выбрана в любой из этих комбинаций (или в обеих), то длины этих двух комбинаций в битах должны быть разными.

23.3.2.5 Если «**ENCODING-SPACE SIZE**» равен «**self-delimiting**», то «**TRUE-PATTERN**» и «**FALSE-PATTERN**» должны образовать саморазграничивающий набор (см. 3.2.42).

23.3.2.6 «**UNUSED BITS DETERMINED BY**» не должен быть «**not-needed**», если не соблюдено следующее:

a) обе комбинации кратны единице «**ENCODING-SPACE MULTIPLE OF**» целое число раз, а «**ENCODINGSPACE SIZE**» равен «**variable-with-determinant**», либо

b) обе комбинации имеют одинаковые длины, либо

c) «**JUSTIFIED**» равен «**left**», а комбинации образуют саморазграничивающий набор, либо

d) «**JUSTIFIED**» равен «**right**», а реверсии комбинаций образуют саморазграничивающий набор (см. 3.2.42).

23.3.2.7 Если в пространстве кодирования имеются неиспользованные биты, то должно устанавливаться «**VALUEPADDING**».

23.3.3 Действия кодера

23.3.3.1 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) замена;

b) предварительное выравнивание и заполнение;

c) начальный указатель;

d) пространство кодирования (см. 23.3.3.2);

e) кодирование значения (см. 23.3.3.3);

f) заполнение и выравнивание значения;

g) идентификационный описатель;

h) реверсия битов.

23.3.3.2 Если «**ENCODING-SPACE SIZE**» не установлен в положительное значение, то размером «s» пространства кодирования является наименьшее число единиц «**MULTIPLE OF**» (при условии 23.3.3.3), которое может вместить комбинацию значения, подлежащего кодированию.

23.3.3.3 Кодер может (по своему выбору) увеличить размер «s» пространства кодирования (определенный в 23.3.3.2) на единицы «**MULTIPLE OF**» (при условии любых ограничений, которые предписывают диапазон значений из «**field-tobe-set**» или «**field-to-be-used**»), если «**ENCODING-SPACE SIZE**» установлен в «**encoder-option-withdeterminant**».

23.3.3.4 Число неиспользуемых битов может быть определено из значения «s» и из комбинации значения, подлежащего кодированию.

23.3.3.5 Если число неиспользуемых битов не равно нулю, то применяется «**VALUE-PADDING**».

23.3.4 Действия декодера

23.3.4.1 Для любой установленной группы признаков кодирования декодер выполняет декодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) пространство кодирования;

d) реверсия битов;

e) заполнение и выравнивание значения;

f) декодирование значения (см. 23.3.4.2).

23.3.4.2 Декодирование значения выполняется путем определения «**TRUE-PATTERN**» или «**FALSE-PATTERN**» с помощью:

a) использования определения «**UNUSED BITS**», если оно имеется, либо

b) использования признака саморазграничения для комбинаций или их реверсий.

23.4 Определение объектов кодирования для классов в категории «цепочка знаков»

23.4.1 Определенный синтаксис

Синтаксис для определения объектов кодирования для классов в категории «цепочка знаков» определяется следующим образом:

```
#CHARS ::= ENCODING-CLASS {

    -- Спецификация предварительного выравнивания и заполнения (см. 22.2)
    &encoding-space-pre-alignment-unit   Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding          Padding DEFAULT zero,
    &encoding-space-pre-pattern          Non-Null-Pattern (ALL EXCEPT different:any)
                                         DEFAULT bits:'0'B,

    -- Спецификация начального указателя (см. 22.3)
    &start-pointer                       REFERENCE OPTIONAL,
    &start-pointer-unit                  Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms    #TRANSFORM ORDERED OPTIONAL,

    -- Кодирование значений знаков
    &value-reversal                      BOOLEAN DEFAULT FALSE,
    &Transforms                          #TRANSFORM ORDERED OPTIONAL,
    &Chars-repetition-encodings          #CONDITIONAL-REPETITION ORDERED OPTIONAL,
    &chars-repetition-encoding           #CONDITIONAL-REPETITION OPTIONAL,

    -- Спецификация идентификационного описателя (см. 22.9)
    &exhibited-handle                    PrintableString DEFAULT "default-handle",
    &Handle-positions                    INTEGER (0..MAX) OPTIONAL,
    &handle-value-set                    HandleValueSet DEFAULT tag:any
} WITH SYNTAX {
      [ALIGNED TO
            [NEXT]
            [ANY]
                  &encoding-space-pre-alignment-unit
                  [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
      [START-POINTER &start-pointer
            [MULTIPLE OF &start-pointer-unit]
            [ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
      [VALUE-REVERSAL &value-reversal]
      [TRANSFORMS &Transforms]
      [REPETITION-ENCODINGS &Chars-repetition-encodings]
      [REPETITION-ENCODING &chars-repetition-encoding]
      [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
}
```

23.4.2 Модель для кодирования классов в категории «цепочка знаков»

23.4.2.1 Модель кодирований цепочки знаков содержит следующее:

a) порядок следования знаков в цепочке знаков может быть реверсирован;

b) знаки рассматриваются как повторение знака;

c) имеется преобразование (указанное в «**TRANSFORMS**»), при котором каждый знак преобразуется в саморазграничивающую цепочку битов;

d) «**REPETITION-ENCODING**» или «**REPETITION-ENCODINGS**» указывает, как следует кодировать повторение цепочки битов.

Примечание — Единственной целью разрешения «**REPETITION-ENCODING**» так же, как «**REPETITION-ENCODINGS**", является обеспечение синтаксиса, который не содержит двойных фигурных скобок («{{«) в общем случае одиночного условного кодирования. Использование «**REPETITION-ENCODINGS**», когда имеется одиночное условное кодирование, не одобряется, но разрешается.

23.4.2.2 Границы (если присутствуют) для кодируемого класса (класса в категории «цепочка знаков») являются границами числа знаков в цепочке знаков, формирующей каждое абстрактное значение.

23.4.2.3 При рассмотрении повторения знаков эти границы считаются границами числа повторений и могут использоваться в спецификации тех объектов кодирования класса **#REPETITION-ENCODING**, которые применены в спецификации этого объекта кодирования.

23.4.3 Цель и ограничения

23.4.3.1 Этот синтаксис используется с целью определения начала пространства кодирования для класса в категории «цепочка знаков», для кодирования абстрактных значений, связанных с этим классом, и для факультативного объявления о том, что объект кодирования показывает заданный идентификационный описатель (с заданным набором значений описателя).

23.4.3.2 **#CONDITIONAL-REPETITION**, который применяется этим объектом кодирования, не должен указывать «**REPLACE**», если он не равен «**REPLACE STRUCTURE**».

23.4.3.3 Если какой-либо объект кодирования **#CONDITIONAL-REPETITION** содержит раздел «**REPLACE STRUCTURE**», то все объекты кодирования **#CONDITIONAL-REPETITION** должны содержать раздел «**REPLACE STRUCTURE**».

23.4.3.4 Если в объектах кодирования **#CONDITIONAL-REPETITION** нет раздела «**REPLACE STRUCTURE**», то должен быть установлен «**TRANSFORMS**». Если в объектах кодирования **#CONDITIONAL-REPETITION** имеется раздел «**REPLACE STRUCTURE**», то остальные параметры не устанавливаются. Если объект кодирования структуры замены показывает описатель (с заданным набором значений описателя), определяемый объект кодирования показывает тот же идентификационный описатель (с тем же набором значений описателя — см. 22.1.1.11).

23.4.3.5 Первый преобразователь в «**TRANSFORMS**» должен иметь источник, который является одиночным знаком, а последний преобразователь должен иметь результат, который является цепочкой битов. Цепочки битов, образованные для набора всех знаков, подлежащих кодированию, должны формировать саморазграничивающий набор (см. 3.2.42).

Примечание — Это означает, что окончательное преобразование должно быть саморазграничивающим.

23.4.3.6 Спецификация ECN или применение будут ошибочными, когда какой-либо преобразователь в «**TRANSFORMS**» не является обратимым для абстрактного значения, к которому он применен.

23.4.3.7 Должен устанавливаться только один из «**REPETITION-ENCODING**» и «**REPETITION-ENCODINGS**».

23.4.3.8 Если объект кодирования в упорядоченном списке «**REPETITION-ENCODINGS**» определен с помощью «**IF**» или «**IF-ALL**», то все предыдущие объекты кодирования в этом списке должны быть определены с помощью «**IF**» или «**IF-ALL**».

23.4.3.9 Если «**EXHIBITS HANDLE**» установлен, то объект кодирования показывает определенный идентификационный описатель.

Примечание — При этом, как правило, потребуются ограничения на абстрактные значения связанного типа, или добавления избыточных битов при преобразовании каждого знака, или то и другое.

23.4.3.10 Если «**EXHIBITS HANDLE**» установлен, то «**ALIGNED TO**» не должен устанавливаться во всех спецификациях «**REPETITION-ENCODING(S)**».

23.4.4 Действия кодера

23.4.4.1 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующем последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) кодирование значений знаков (см. 23.4.4.3);.

d) кодирование повторения, как указано первым «**REPETITION-ENCODING(S)**», условие которого удовлетворяется;

e) спецификация идентификационного описателя.

23.4.4.2 Спецификация ECN будет ошибочной, если отсутствует «**REPETITION-ENCODING(S)**», условие которого удовлетворяется.

23.4.4.3 Для кодирования значения цепочки знаков кодер выполняет следующее:

a) реверсирует порядок следования знаков в абстрактном значении целой цепочки знаков, если «**VALUEREVERSAL**» установлен в **TRUE:**

b) рассматривает значение цепочки знаков как повторение знака;

c) применяет указанный «**TRANSFORMS**» (если он есть) к каждому знаку для образования повторения битов;

d) кодирует повторение путем применения «**REPETITION-ENCODING(S)**».

23.4.5 Действия декодера

23.4.5.1 Для любой установленной группы признаков кодирования декодер выполняет декодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) декодирование повторения, как указано первым **REPETITION-ENCODING(S)**», условие которого удовлетворяется;

d) декодирование значения цепочки знаков (см. 23.4.5.2).

23.4.5.2 При декодировании значения цепочки знаков декодер использует «**REPETITION-ENCODING(S)**» для определения пространства повторения и восстановления исходных знаков. Если «**TRANSFORMS**» установлен, то декодер использует признак саморазграничения (который охватывает возможную фиксированную длину) в кодировании каждого знака для определения конца каждого повторения и реверсирует преобразователи для восстановления значения цепочки знаков.

23.4.5.3 Если «**VALUE-REVERSAL**» установлен в **TRUE**, то окончательный порядок следования знаков в абстрактном значении цепочки знаков реверсируется.

23.5 Определение объектов кодирования для классов в категории «конкатенация»

23.5.1 Определенный синтаксис

Синтаксис для определения объектов кодирования для классов в категории «конкатенация» определяется следующим образом:

```
#CONCATENATION ::= ENCODING-CLASS {

    -- Спецификация полной замены (см. 22.1)
    &#Replacement-structure
            OPTIONAL,
    &#Replacement-structure2
            OPTIONAL,
    &replacement-structure-encoding-object &#Replacement-structure      OPTIONAL,
    &replacement-structure-encoding-object2 &#Replacement-structure2    OPTIONAL,
    &#Head-end-structure                    OPTIONAL,
    &#Head-end-structure2                   OPTIONAL,

    -- Спецификация предварительного выравнивания и заполнения (см. 22.2)
    &encoding-space-pre-alignment-unit  Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding         Padding DEFAULT zero,
    &encoding-space-pre-pattern         Non-Null-Pattern (ALL EXCEPT different:any)
                                        DEFAULT bits:'0'B,

    -- Спецификация начального указателя (см. 22.3)
    &start-pointer                      REFERENCE OPTIONAL,
    &start-pointer-unit                 Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,

    -- Спецификация пространства кодирования (см. 22.4)
    &encoding-space-size                EncodingSpaceSize
                                        DEFAULT self-delimiting-values,
    &encoding-space-unit                Unit (ALL EXCEPT repetitions)
                                        DEFAULT bit,
    &encoding-space-determination       EncodingSpaceDetermination
                                        DEFAULT field-to-be-set,
```

```
        &encoding-space-reference       REFERENCE OPTIONAL,
        &Encoder-transforms             #TRANSFORM ORDERED OPTIONAL,
        &Decoder-transforms             #TRANSFORM ORDERED OPTIONAL,

        -- Спецификация конкатенации (см. 22.10)
        &concatenation-order            ENUMERATED {textual, tag, random}
                                        DEFAULT textual,
        &concatenation-alignment        ENUMERATED {none, aligned}
                                        DEFAULT aligned,
        &concatenation-handle           PrintableString
                                        DEFAULT "default-handle",

        -- Заполнение и выравнивание значения (см. 22.8)
        &value-justification                    Justification DEFAULT right:0,
        --&value-pre-padding                   Padding DEFAULT zero,
        &value-pre-pattern                      Non-Null-Pattern DEFAULT bits:'0'B,
        &value-post-padding                     Padding DEFAULT zero,
        &value-post-pattern                     Non-Null-Pattern DEFAULT bits:'0'B,
        &unused-bits-determination              UnusedBitsDetermination
                                                DEFAULT field-to-be-set,
        &unused-bits-reference                  REFERENCE OPTIONAL,
        &Unused-bits-encoder-transforms         #TRANSFORM ORDERED OPTIONAL,
        &Unused-bits-decoder-transforms         #TRANSFORM ORDERED OPTIONAL,

        -- Спецификация идентификационного описателя (см. 22.9)
        &exhibited-handle       PrintableString DEFAULT "default-handle",
        &Handle-positions       INTEGER (0..MAX) OPTIONAL,
        &handle-value-set       HandleValueSet DEFAULT tag:any,

        -- Спецификация реверсии битов (см. 22.12)
        &bit-reversal           ReversalSpecification
                                DEFAULT no-reversal

} WITH SYNTAX {
        [REPLACE
                [STRUCTURE]
                [COMPONENT]
                [ALL COMPONENTS]
                [OPTIONALS]
                [NON-OPTIONALS]
                WITH &#Replacement-structure
                        [ENCODED BY &replacement-structure-encoding-object
                                [INSERT AT HEAD &#Head-end-structure]]
                        [AND OPTIONALS WITH &#Replacement-structure2
                                [ENCODED BY &replacement-structure-encoding-object2
                                        [INSERT AT HEAD &#Head-end-structure2]]] ]
        [ALIGNED TO
                [NEXT]
                [ANY]
                        &encoding-space-pre-alignment-unit
                        [PADDING &encoding-space-pre-padding
                                [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER &start-pointer
                [MULTIPLE OF &start-pointer-unit]
                [ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
        ENCODING-SPACE
```

```
                [SIZE &encoding-space-size
                        [MULTIPLE OF &encoding-space-unit]]
                [DETERMINED BY &encoding-space-determination]
                [USING &encoding-space-reference
                        [ENCODER-TRANSFORMS &Encoder-transforms]
                        [DECODER-TRANSFORMS &Decoder-transforms]]
[CONCATENATION
                [ORDER &concatenation-order]
                [ALIGNMENT &concatenation-alignment]
                [HANDLE &concatenation-handle]]
    [VALUE-PADDING
                [JUSTIFIED &value-justification]
                [PRE-PADDING &value-pre-padding
                        [PATTERN &value-pre-pattern]]
                [POST-PADDING &value-post-padding
                        [PATTERN &value-post-pattern]]
                [UNUSED BITS
                        [DETERMINED BY &unused-bits-determination]
                        [USING &unused-bits-reference
                                [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                                [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
    [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
    [BIT-REVERSAL &bit-reversal]
}
```

23.5.2 Цель и ограничения

23.5.2.1 Этот синтаксис используется для определения начала пространства кодирования для класса в категории «конкатенация», способа комбинирования кодирований компонентов и их расположения внутри пространства кодирования, для факультативного объявления о том, что объект кодирования показывает заданный идентификационный описатель (с заданным набором значений описателя), и для возможной реверсии битов в пространстве кодирования.

23.5.2.2 Если «**REPLACE STRUCTURE**» установлен, то никакие другие группы параметров кодирования не устанавливаются. Если объект кодирования структуры замены показывает описатель (с заданным набором значений описателя), определяемый объект кодирования показывает тот же идентификационный описатель (с тем же набором значений описателя — см. 22.1.1.11).

23.5.2.3 «**ENCODING-SPACE SIZE**» должен быть «**variable-with-determinant**» или «**self-delimitingvalues**».

23.5.2.4 Если «**EXHIBITS HANDLE**» установлен, то объект кодирования показывает определенный идентификационный описатель.

23.5.3 Действия кодера

23.5.3.1 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) замена;

b) предварительное выравнивание и заполнение;

c) начальный указатель;

d) пространство кодирования (см. 23.5.3.2);

e) конкатенация;

f) заполнение и выравнивание значения;

g) спецификация идентификационного описателя;

h) реверсия битов.

23.5.3.2 Если «**ENCODING SPACE**» равен «**variable-with-determinant**», то размер пространства кодирования должен иметь минимальное число единиц «**MULTIPLE OF**», необходимое для размещения конкатенации.

23.5.4 Действия декодера

23.5.4.1 Для любой установленной группы признаков кодирования декодер выполняет декодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;
b) начальный указатель;
c) пространство кодирования;
d) реверсия битов;
e) заполнение и выравнивание значения;
f) конкатенация.

23.6 Определение объектов кодирования для классов в категории «целочисленная»

23.6.1 Определенный синтаксис

Синтаксис с целью определения объектов кодирования для классов в категории «целочисленная» определяется следующим образом:

```
#INT ::= ENCODING-CLASS {
        -- Codage d'entier
        &Integer-encodings      #CONDITIONAL-INT ORDERED OPTIONAL,
        &integer-encoding       #CONDITIONAL-INT OPTIONAL
} WITH SYNTAX {
          [ENCODINGS &Integer-encodings]
          [ENCODING &integer-encoding]
}
```

23.6.2 Цель и ограничения

23.6.2.1 Этот синтаксис используется для определения кодирования класса в категории «целочисленная» путем описания одной или нескольких кодовых последовательностей класса **#CONDITIONAL-INT**.

23.6.2.2 Должен быть установлен только один из «**ENCODING**» и «**ENCODINGS**».

Примечание — Единственной целью разрешения «**ENCODING**», так же как «**ENCODINGS**», является обеспечение синтаксиса, который не содержит двойных фигурных скобок («{{») в общем случае одиночного объекта кодирования. Использование «**ENCODINGS**», когда имеется одиночный объект кодирования, не одобряется, но разрешается.

23.6.2.3 Если какой-либо объект кодирования в упорядоченном списке «**ENCODINGS**» определен с помощью «**IF**» или «**IF-ALL**», то все предыдущие объекты кодирования в этом списке должны быть определены с помощью «**IF**» или «**IF-ALL**».

23.6.3 Действия кодера

23.6.3.1 Кодер выбирает и применяет первый объект кодирования **#CONDITIONAL-INT** в «**ENCODING(S)**», условия которого удовлетворяются. Спецификация ECN будет ошибочной, если ни одно условное кодирование не имеет условий, которые удовлетворяются.

Примечание — Может быть редкий, но незаконный случай, когда присутствуют объекты кодирования **#CONDITIONAL-INT**, которые никогда не используются, так как условия при использовании более ранних объектов кодирования всегда могут удовлетворяться.

23.6.4 Действия декодера

23.6.4.1 Декодер выбирает и использует первый объект кодирования **#CONDITIONAL-INT** в «**ENCODING(S)**», условия которого удовлетворяются.

23.7 Определение объектов кодирования для класса #CONDITIONAL-INT

23.7.1 Определенный синтаксис

Синтаксис с целью определения объектов кодирования для класса **#CONDITIONAL-INT** определяется следующим образом:

```
#CONDITIONAL-INT ::= ENCODING-CLASS {
```

-- Условие (см. 21.11)
&range-condition RangeCondition OPTIONAL,
&comparison Comparison OPTIONAL,
&comparator INTEGER OPTIONAL,
&Range-conditions RangeCondition ORDERED OPTIONAL,
&Comparisons Comparison ORDERED OPTIONAL,
&Comparators INTEGER ORDERED OPTIONAL,

-- Спецификация замены исключительно структуры (см. 22.1)
&#Replacement-structure
OPTIONAL,
&replacement-structure-encoding-object &#Replacement-structure OPTIONAL,

-- Спецификация предварительного выравнивания и заполнения (см. 22.2)
&encoding-space-pre-alignment-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
&encoding-space-pre-padding Padding DEFAULT zero,
&encoding-space-pre-pattern Non-Null-Pattern (ALL EXCEPT different:any) DEFAULT bits:'0'B,

-- Спецификация начального указателя (см. 22.3)
&start-pointer REFERENCE OPTIONAL,
&start-pointer-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
&Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

-- Спецификация пространства кодирования (см. 22.4)
&encoding-space-size EncodingSpaceSize DEFAULT self-delimiting-values,
&encoding-space-unit Unit (ALL EXCEPT repetitions) DEFAULT bit,
&encoding-space-determination EncodingSpaceDetermination DEFAULT field-to-be-set,
&encoding-space-reference REFERENCE OPTIONAL,
&Encoder-transforms #TRANSFORM ORDERED OPTIONAL,
&Decoder-transforms #TRANSFORM ORDERED OPTIONAL,

-- Кодирование значения
&Transform #TRANSFORM ORDERED OPTIONAL,
&encoding ENUMERATED {positive-int, twos-complement, reverse-positive-int, reverse-twos-complement} DEFAULT twos-complement,

-- Заполнение и выравнивание значения (см. 22.8)
&value-justification Justification DEFAULT right:0,
&value-pre-padding Padding DEFAULT zero,
&value-pre-pattern Non-Null-Pattern DEFAULT bits:'0'B,
&value-post-padding Padding DEFAULT zero,
&value-post-pattern Non-Null-Pattern DEFAULT bits:'0'B,
&unused-bits-determination UnusedBitsDetermination DEFAULT field-to-be-set,
&unused-bits-reference REFERENCE OPTIONAL,
&Unused-bits-encoder-transforms #TRANSFORM ORDERED OPTIONAL,
&Unused-bits-decoder-transforms #TRANSFORM ORDERED OPTIONAL,

-- Спецификация идентификационного описателя (см. 22.9)
&exhibited-handle PrintableString DEFAULT "default-handle",
&Handle-positions INTEGER (0..MAX) OPTIONAL,
&handle-value-set HandleValueSet DEFAULT tag:any,

```
        -- Спецификация реверсии битов (см. 22.12)
        &bit-reversal       ReversalSpecification
                            DEFAULT no-reversal
    } WITH SYNTAX {
        [IF &range-condition [&comparison &comparator]]
        [IF-ALL &Range-conditions [&Comparisons &Comparators]]
        [ELSE]
        [REPLACE
                [STRUCTURE]
                WITH &#Replacement-structure
                        [ENCODED BY &replacement-structure-encoding-object]]
        [ALIGNED TO
                [NEXT]
                [ANY]
                        &encoding-space-pre-alignment-unit
                        [PADDING &encoding-space-pre-padding
                                [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER &start-pointer
                [MULTIPLE OF &start-pointer-unit]
                [ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
          ENCODING-SPACE
                [SIZE &encoding-space-size
                      [MULTIPLE OF &encoding-space-unit]]
                [DETERMINED BY &encoding-space-determination]
                [USING &encoding-space-reference
                        [ENCODER-TRANSFORMS &Encoder-transforms]
                        [DECODER-TRANSFORMS &Decoder-transforms]]
          [TRANSFORMS &Transforms]
          [ENCODING &encoding]
          [VALUE-PADDING
                [JUSTIFIED &value-justification]
                [PRE-PADDING &value-pre-padding
                        [PATTERN &value-pre-pattern]]
                [POST-PADDING &value-post-padding
                        [PATTERN &value-post-pattern]]
                [UNUSED BITS
                        [DETERMINED BY &unused-bits-determination]
                        [USING &unused-bits-reference
                                [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                                [DECODER-TRANSFORMS&Unused-bits-decoder-transforms]]]]
          [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
                [AS &handle-value]]
          [BIT-REVERSAL &bit-reversal]
    }
```

23.7.2 Цель и ограничения

23.7.2.1 Этот синтаксис используется для определения объекта кодирования **#CONDITIONAL-INT**. В спецификации объекта кодирования класса в категории «целочисленная» будет использоваться только такой объект кодирования.

23.7.2.2 Синтаксис допускает спецификацию одиночного условия на границах целого числа для этого применяемого кодирования (с помощью «**IF**»). Он допускает также спецификацию того, что весь набор условий подлежит удовлетворению (применение «**IF-ALL**»). Он допускает также спецификацию того, что нет условия. Использование «**ELSE**» или пропуск «**IF**», «**IF-ALL**» и «**ELSE**» указывают, что нет условия. «**IF-ALL**» должен использоваться с тремя списками, если один или более size-range-conditions требуют сравнения, или же с одним списком в противном случае. При использовании трех списков size-range-conditions, которые не требуют сравнения или компаратора (если таковые имеются),

должны следовать всем тем, что требуют сравнения, и не должны иметь соответствующей записи во втором и третьем списках. При использовании «**IF-ALL**» с тремя списками списки должны быть интерпретированы как списки предикатов, использующие значения в соответствующих позициях в трех списках.

Примечание — Рекомендуется, чтобы три списка были отформатированы для предоставления условия в каждом столбце.

Пример
IF-ALL {test-lower-bound, test-range , bounded-with-negatives }
{greater-than , less-than-or-equal-to }
{-10 , 20 }

23.7.2.3 Используя этот синтаксис, спецификатор ECN может определять начало пространства кодирования для кодирования класса в категории «целочисленная», кодирование абстрактных значений, связанных с этим классом, их размещение внутри пространства кодирования и возможную реверсию битов в пространстве кодирования.

23.7.2.4 Должен присутствовать один самый больший из «**IF**», «**IF-ALL**» и «**ELSE**».

23.7.2.5 Если «**REPLACE**» установлен, то никакие другие группы признаков кодирования не устанавливаются.

23.7.2.6 Спецификация ECN или применение будут ошибочными, когда какой-либо преобразователь в «**TRANSFORMS**» не является обратимым для абстрактного значения, к которому он применен. Первый преобразователь из «**TRANSFORMS**», если присутствует, должен иметь источник, который является целым числом, а последний преобразователь должен иметь результат, который является целым числом.

Примечание — Тест на условие «**IF**» или «**IF-ALL**» производится на границах исходного значения и не зависит от этих преобразователей.

23.7.2.7 Преобразователь «**INT-TO-INT**» со значением «**subtract:lower-bound**» вводится только в случаях, когда условие «**IF**» или «**IF-ALL**» ограничивает применение этого кодирования до класса в категории «целочисленная» с нижней границей, и должен быть (если имеется) первым преобразователем в списке.

23.7.2.8 «**ENCODING-SPACE SIZE**» не должен равняться «**fixed-to-max**», если условие «**IF**» или «**IF-ALL**» не ограничивает кодирования до класса с верхней и нижней границами.

23.7.2.9 «**ENCODING-SPACE SIZE**» не должен устанавливаться в «**self-delimiting-values**».

Примечание — Это означает, что значение по умолчанию (устанавливается для обеспечения совместимости с другими использованиями этого типа) всегда должно быть переопределено.

23.7.2.10 Если «**EXHIBITS HANDLE**» установлен, то спецификатор считает, что кодирование любого значения показывает идентификационный описатель.

Примечание — При этом, как правило, потребуется использование «**VALUE-PADDING**» с выравниванием слева, чтобы разрешать заполнению показывать идентификационный описатель.

23.7.3 Действия кодера

23.7.3.1 Кодер обнаруживает ошибку спецификации ECN или применения, когда какое-либо ограничение из 23.7.2 нарушается.

23.7.3.2 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) замена;
b) предварительное выравнивание и заполнение;
c) начальный указатель;
d) пространство кодирования;
e) кодирование значения (см. ниже);
f) заполнение и выравнивание значения;
g) идентификационный описатель;
h) реверсия битов.

23.7.3.3 Кодер применяет «**TRANSFORMS**», если он есть, к значению, подлежащему кодированию.

23.7.3.4 Кодер использует следующий диапазон целочисленных значений, которые могут быть закодированы в «n» битах:

«**ENCODING**»	Минимальное значение	Максимальное значение
«**positive-int**»	0	$2^n - 1$
«**reverse-positive-int**»	0	$2^n - 1$
«**twos-complement**»	-2^{n-1}	$-2^{n-1} - 1$
«**reverse-twos-complement**»	-2^{n-1}	$-2^{n-1} - 1$

23.7.3.5 Параметр «**ENCODING**» выбирает кодирование в виде кодирования с дополнением до 2, или кодирования положительных целых чисел, или реверсии одного из них. Описание кодирования с дополнением до 2 и кодирования положительных целых чисел приведено в ИСО/МЭК 8825-1, пункты 8.3.2 и 8.3.3. Реверсией этих кодирований является кодирование, в котором после создания «n» битов порядок следования этих «n» битов реверсируется.

23.7.3.6 Кодер обнаруживает ошибку спецификации ECN или применения, когда какое-либо значение должно быть закодировано в число битов, которое недостаточно, как указано в 23.7.3.4.

23.7.3.7 Если «**ENCODING-SPACE SIZE**» является положительным целым числом, то его размер в битах вычисляется как «**SIZE**», умноженный на единицы «**MULTIPLE OF**». Если «**VALUE-PADDING**» не установлен, то это будет числом битов «n», в которые будет кодироваться целое число, а неиспользуемых битов не будет. Если «**VALUE-PADDING**» установлен, то число битов, в которые будет кодироваться целое число, уменьшается на значение целого числа «m», указанное для «**JUSTIFIED**», и будет «m» неиспользуемых битов.

23.7.3.8 Если «**ENCODING-SPACE SIZE**» равен «**fixed-to-max**», то кодер определяет минимальное число единиц «**MULTIPLE OF**», которое имеет достаточно битов для кодирования любого из значений класса, и действует далее (как описано выше), считая «**SIZE**» положительным числом, установленным в это значение.

23.7.3.9 Если «**ENCODING-SPACE SIZE**» равен «**variable-with-determinant**», то кодер определяет минимальное число единиц «**MULTIPLE OF**» (например, «s»), которое имеет достаточно битов для кодирования реального абстрактного значения, подлежащего кодированию, и действует далее (как описано выше), считая «**SIZE**» положительным целым числом, установленным в это значение.

23.7.3.10 Кодер может (по своему выбору) увеличить «s» (определенное в 23.7.3.9) на единицы «**MULTIPLE OF**» (при условии любых ограничений, которые предписывают диапазон значений из «**field-to-be-set**» или «**field-tobe-used**»), если «**ENCODING-SPACE SIZE**» установлен в «**encoder-option-with-determinant**».

23.7.3.11 Кодер действует далее (как описано выше), считая «**SIZE**» положительным целым числом, установленным в «s».

23.7.4 Действия декодера

23.7.4.1 Для любой установленной группы признаков кодирования декодер выполняет декодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;
b) начальный указатель;
c) пространство кодирования;
d) реверсия битов;
e) заполнение и выравнивание значения;
f) декодирование значения (см. 23.7.4.2).

23.7.4.2 Декодер восстанавливает значение целого числа из битов, использованных для кодирования, декодируя согласно указанному кодированию, и затем реверсирует «**TRANSFORMS**» (если он указан) для восстановления исходного абстрактного значения.

23.8 Определение объектов кодирования для классов в «вырожденной» категории

23.8.1 Определенный синтаксис

Синтаксис с целью определения объектов кодирования для классов в «вырожденной» категории определяется следующим образом:

```
#NUL ::= ENCODING-CLASS {

    -- Спецификация замены исключительно структуры (см. 22.1)
    &#Replacement-structure
        OPTIONAL,
    &replacement-structure-encoding-object &#Replacement-structure OPTIONAL,

    -- Спецификация предварительного выравнивания и заполнения (см. 22.2)
    &encoding-space-pre-alignment-unit   Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding          Padding DEFAULT zero,
    &encoding-space-pre-pattern          Non-Null-Pattern (ALL EXCEPT different:any)
                                         DEFAULT bits:'0'B,

  -- Спецификация начального указателя (см. 22.3)
  &start-pointer                         REFERENCE OPTIONAL,
  &start-pointer-unit                    Unit (ALL EXCEPT repetitions) DEFAULT bit,
  &Start-pointer-encoder-transforms      #TRANSFORM ORDERED OPTIONAL,

  -- Спецификация пространства кодирования (см. 22.4)
  &encoding-space-size                   EncodingSpaceSize
                                         DEFAULT self-delimiting-values,
  &encoding-space-unit                   Unit (ALL EXCEPT repetitions)
                                         DEFAULT bit,
  &encoding-space-determination          EncodingSpaceDetermination
                                         DEFAULT field-to-be-set,
  &encoding-space-reference              REFERENCE OPTIONAL,
  &Encoder-transforms                    #TRANSFORM ORDERED OPTIONAL,
  &Decoder-transforms                    #TRANSFORM ORDERED OPTIONAL,

  -- Структура значения
  &value-pattern        Pattern (ALL EXCEPT different:any)
                        DEFAULT bits:"B,

  -- Заполнение и выравнивание значения (см. 22.8)
  &value-justification                   Justification DEFAULT right:0,
  &value-pre-padding                     Padding DEFAULT zero,
  &value-pre-pattern                     Non-Null-Pattern DEFAULT bits:'0'B,
  &value-post-padding                    Padding DEFAULT zero,
  &value-post-pattern                    Non-Null-Pattern DEFAULT bits:'0'B,
  &unused-bits-determination             UnusedBitsDetermination
                                         DEFAULT field-to-be-set,
  &unused-bits-reference                 REFERENCE OPTIONAL,
  &Unused-bits-encoder-transforms        #TRANSFORM ORDERED OPTIONAL,
  &Unused-bits-decoder-transforms        #TRANSFORM ORDERED OPTIONAL,

  -- Спецификация идентификационного описателя (см. 22.9)
  &exhibited-handle                      PrintableString DEFAULT "default-handle",
  &Handle-positions                      INTEGER (0..MAX) OPTIONAL,
  &handle-value-set                      HandleValueSet DEFAULT tag:any,

    -- Спецификация реверсии битов (см. 22.12)
    &bit-reversal       ReversalSpecification
                        DEFAULT no-reversal
} WITH SYNTAX {
    [REPLACE
        [STRUCTURE]
```

```
                WITH &#Replacement-structure
                    [ENCODED BY &replacement-structure-encoding-object]]
        [ALIGNED TO
                [NEXT]
                [ANY]
                    &encoding-space-pre-alignment-unit
                    [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER &start-pointer
                [MULTIPLE OF &start-pointer-unit]
                [ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
        ENCODING-SPACE
                [SIZE &encoding-space-size
                    [MULTIPLE OF &encoding-space-unit]]
                [DETERMINED BY &encoding-space-determination]
                [USING &encoding-space-reference
                    [ENCODER-TRANSFORMS &Encoder-transforms]
                    [DECODER-TRANSFORMS &Decoder-transforms]]
        [NULL-PATTERN &value-pattern]
        [VALUE-PADDING
                [JUSTIFIED &value-justification]
                [PRE-PADDING &value-pre-padding
                    [PATTERN &value-pre-pattern]]
                [POST-PADDING &value-post-padding
                    [PATTERN &value-post-pattern]]
                [UNUSED BITS
                [DETERMINED BY &unused-bits-determination]
                [USING &unused-bits-reference
                    [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                    [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
    [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
                [AS &handle-value]]
          [BIT-REVERSAL &bit-reversal]
}
```

23.8.2 Цель и ограничения

23.8.2.1 Этот синтаксис используется для определения кодирования класса в вырожденной категории (null category).

23.8.2.2 Если «**REPLACE STRUCTURE**» установлен, то никакие другие группы признаков кодирования не устанавливаются. Если объект кодирования структуры замены показывает описатель (с заданным набором значений описателя), определяемый объект кодирования показывает тот же идентификационный описатель (с тем же набором значений описателя — см. 22.1.1.11).

23.8.2.3 Если «**ENCODING-SPACE SIZE**» положителен, то он должен быть достаточным для помещения размера «**NULL-PATTERN**» вместе с любыми битами, добавленными в результате спецификации «**VALUE-PADDING**».

23.8.2.4 Если в пространстве кодирования имеются неиспользуемые биты, то должно быть установлено «**VALUEPADDING**».

23.8.3 Действия кодера

23.8.3.1 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) замена;

b) предварительное выравнивание и заполнение;

c) начальный указатель;

d) пространство кодирования;

e) кодирование значения (см. 23.8.3.2);

f) заполнение и выравнивание значения;

g) идентификационный описатель;

h) реверсия битов.

23.8.3.2 Кодированием значения будут биты из «**NULL-PATTERN**».

23.8.3.3 Если «**ENCODING-SPACE SIZE**» равен «**variable-with-determinant**» или «**encode-option-withdeterminant**», то он должен быть минимальным числом единиц «**MULTIPLE OF**», необходимым для размещения комбинации (например, «s»), которая определяется в 23.8.3.4.

23.8.3.4 Кодер может (по своему выбору) увеличить «s» (определенный в 23.8.3.3) на единицы «**MULTIPLE OF**» (при условии любых ограничений, которые предписывают диапазон значений из «**field-to-be-set**» или «**field-tobe-used**»), если «**ENCODING-SPACE SIZE**» установлен в «**encoder-option-with-determinant**».

23.8.3.5 Если в пространстве кодирования имеются неиспользуемые биты, то должно быть установлено «**VALUEPADDING**».

23.8.4 Действия декодера

23.8.4.1 Для любой установленной группы признаков кодирования декодер выполняет декодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) пространство кодирования;

d) реверсия битов;

e) заполнение и выравнивание значения.

23.8.4.2 Декодер определяет размер нулевой комбинации и опознает такие биты в кодировании, но пассивно принимает любое значение таких битов.

23.9 Определение объектов кодирования для классов в категории «цепочка октетов»

23.9.1 Определенный синтаксис

Синтаксис с целью определения объектов кодирования для классов в категории «цепочка октетов» определяется следующим образом:

```
#OCTETS ::= ENCODING-CLASS {

    -- Спецификация предварительного выравнивания и заполнения (см. 22.2)
    &encoding-space-pre-alignment-unit   Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &encoding-space-pre-padding          Padding DEFAULT zero,
    &encoding-space-pre-pattern          Non-Null-Pattern (ALL EXCEPT different:any)
                                         DEFAULT bits:'0'B,

    -- Спецификация начального указателя (см. 22.3)
    &start-pointer                       REFERENCE OPTIONAL,
    &start-pointer-unit                  Unit (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms    #TRANSFORM ORDERED OPTIONAL,

    -- Кодирование значений октетов
    &value-reversal                      BOOLEAN DEFAULT FALSE,
    &Transforms                          #TRANSFORM ORDERED OPTIONAL,
    &Octets-repetition-encodings         #CONDITIONAL-REPETITION ORDERED OPTIONAL,
    &octets-repetition-encoding          #CONDITIONAL-REPETITION OPTIONAL,

    -- Спецификация идентификационного описателя (см. 22.9)
    &exhibited-handle                    PrintableString DEFAULT "default-handle",
    &Handle-positions                    INTEGER (0..MAX) OPTIONAL,
    &handle-value-set                    HandleValueSet DEFAULT tag:any,

    -- Спецификация кодирования вложенного типа (см. 22.11)
    &Primary-encoding-object-set         #ENCODINGS OPTIONAL,
```

```
        &Secondary-encoding-object-set      #ENCODINGS OPTIONAL,
        &over-ride-encoded-by               BOOLEAN DEFAULT FALSE

    } WITH SYNTAX {
        [ALIGNED TO
            [NEXT]
                [ANY]
                    &encoding-space-pre-alignment-unit
                    [PADDING &encoding-space-pre-padding
                        [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER &start-pointer
            [MULTIPLE OF &start-pointer-unit]
            [ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
        [VALUE-REVERSAL &value-reversal]
        [TRANSFORMS &Transforms]
        [REPETITION-ENCODINGS &Octets-repetition-encodings]
        [REPETITION-ENCODING &octets-repetition-encoding]
        [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
            [AS &handle-value]]
        [CONTENTS-ENCODING &Primary-encoding-object-set
                        [COMPLETED BY &Secondary-encoding-object-set]
                        [OVERRIDE &over-ride-encoded-by]]
    }
```

23.9.2 Модель для кодирования классов в категории «цепочка октетов»

23.9.2.1 Модель кодирования цепочки октетов содержит следующее:

a) порядок следования октетов в цепочке октетов может быть реверсирован;

b) октеты рассматриваются затем как повторение октета;

c) имеется факультативное преобразование (указанное в «**TRANSFORMS**»), при котором каждый октет преобразуется в саморазграничивающую цепочку битов;

d) «**REPETITION-ENCODING**» или «**REPETITION-ENCODINGS**» указывает, как следует кодировать повторение октета.

Примечание — Единственной целью разрешения «**REPETITION-ENCODING**» так же, как «**REPETITION-ENCODINGS**», является обеспечение синтаксиса, который не содержит двойных фигурных скобок («{{») в общем случае одиночного условного кодирования. Использование «**REPETITION-ENCODINGS**», когда имеется одиночное условное кодирование, не одобряется, но разрешается.

23.9.2.2 Границы (если присутствуют) для кодируемого класса (класса в категории «цепочка октетов») являются границами числа октетов в цепочке октетов, формирующей каждое абстрактное значение.

23.9.2.3 При рассмотрении повторения октета эти границы считаются границами числа повторений и могут использоваться в спецификации тех объектов кодирования класса **#CONDITIONAL-REPETITION**, которые применены в спецификации этого объекта кодирования.

23.9.3 Цель и ограничения

23.9.3.1 Этот синтаксис используется с целью определения начала пространства кодирования для класса в категории «цепочка октетов», для кодирования абстрактных значений, связанных с этим классом, для факультативного объявления о том, что объект кодирования показывает заданный идентификационный описатель (с заданным набором значений описателя), для спецификации способа кодирования вложенного типа.

23.9.3.2 **#CONDITIONAL-REPETITION**, применяемый этим объектом кодирования, не должен указывать «**REPLACE**», если он не равен «**REPLACE STRUCTURE**».

23.9.3.3 Если какой-либо объект кодирования **#CONDITIONAL-REPETITION** содержит раздел «**REPLACE STRUCTURE**», то все объекты кодирования **#CONDITIONAL-REPETITION** должны содержать раздел «**REPLACE STRUCTURE**».

23.9.3.4 Если в объектах кодирования **#CONDITIONAL-REPETITION** имеется раздел «**REPLACE STRUCTURE**», то остальные параметры не устанавливаются. Если объект кодирования структуры замены показывает описатель (с заданным набором значений описателя), определяемый объект кодирования показывает тот же идентификационный описатель (с тем же набором значений описателя — см. 22.1.1.11).

23.9.3.5 Первый преобразователь в «**TRANSFORMS**» (если он есть) должен иметь источник, который является цепочкой битов, а последний преобразователь должен иметь результат, который является саморазграничивающей цепочкой битов (см. 3.2.42).

23.9.3.6 Спецификация ECN или применение будут ошибочными, когда какой-либо преобразователь в «**TRANSFORMS**» не является обратимым для абстрактного значения, к которому он применен.

23.9.3.7 Должен устанавливаться только один из «**REPETITION-ENCODING**» и «**REPETITION-ENCODINGS**».

23.9.3.8 Если объект кодирования в упорядоченном списке «**REPETITION-ENCODINGS**» определен с помощью «**IF**» или «**IF-ALL**», то все предыдущие объекты кодирования в этом списке должны быть определены с помощью «**IF**» или «**IF-ALL**».

23.9.3.9 Если «**EXHIBITS HANDLE**» установлен, то объект кодирования показывает определенный идентификационный описатель.

Примечание — При этом, как правило, потребуются ограничения на абстрактные значения связанного типа.

23.9.3.10 Если «**EXHIBITS HANDLE**» установлен, то «**ALIGNED TO**» не должен устанавливаться во всех спецификациях «**REPETITION-ENCODING(S)**».

23.9.4 Действия кодера

23.9.4.1 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) кодирование значения, как указывается ниже;

d) кодирование повторения, как указано первым «**REPETITION-ENCODING(S)**», условие которого удовлетворяется;

e) идентификационный описатель;

f) кодирование вложенного типа.

23.9.4.2 Для кодирования значения кодер выполняет следующее:

a) реверсирует порядок следования октетов в абстрактном значении целой цепочки октетов, если «**VALUEREVERSAL**» установлен в **TRUE**;

b) рассматривает значение цепочки октетов как повторение октета;

c) применяет «**TRANSFORMS**» (если он есть) к каждому октету для образования повторения цепочки битов;

d) кодирует повторение путем применения первого «**REPETITION-ENCODING(S)**», условие которого удовлетворяется.

23.9.4.3 Спецификация ECN будет ошибочной, если отсутствует «**REPETITION-ENCODING(S)**», условие которого удовлетворяется.

23.9.5 Действия декодера

23.9.5.1 Для любой установленной группы признаков кодирования декодер выполняет декодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) декодирование значения (см. 23.9.5.2);

d) декодирование вложенного типа.

23.9.5.2 Декодер реверсирует «**TRANSFORMS**» (если он есть) для восстановления исходных октетов.

23.9.5.3 Если «**VALUE-REVERSAL**» установлен в **TRUE**, то окончательный порядок следования октетов в абстрактном значении цепочки октетов реверсируется.

23.10 Определение объектов кодирования для классов в категории «открытый тип»

23.10.1 Определенный синтаксис

Синтаксис с целью определения объектов кодирования для классов в категории «открытый тип» определяется следующим образом:

```
#OPEN-TYPE ::= ENCODING-CLASS {
```

```
        -- Спецификация замены исключительно структуры (см. 22.1)
        &#Replacement-structure                                    OPTIONAL,
        &replacement-structure-encoding-object &#Replacement-structure  OPTIONAL,

        -- Спецификация предварительного выравнивания и заполнения (см. 22.2)
        &encoding-space-pre-alignment-unit  Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &encoding-space-pre-padding         Padding DEFAULT zero,
        &encoding-space-pre-pattern         Non-Null-Pattern (ALL EXCEPT different:any)
                                            DEFAULT bits:'0'B,

    -- Спецификация начального указателя (см. 22.3)
    &start-pointer                      REFERENCE OPTIONAL,
    &start-pointer-unit Unit            (ALL EXCEPT repetitions) DEFAULT bit,
    &Start-pointer-encoder-transforms   #TRANSFORM ORDERED OPTIONAL,

    -- Спецификация пространства кодирования (см. 22.4)
    &encoding-space-size            EncodingSpaceSize
                                    DEFAULT self-delimiting-values,
    &encoding-space-unit            Unit (ALL EXCEPT repetitions)
                                    DEFAULT bit,
    &encoding-space-determination   EncodingSpaceDetermination
                                    DEFAULT field-to-be-set,
    &encoding-space-reference       REFERENCE OPTIONAL,
    &Encoder-transforms             #TRANSFORM ORDERED OPTIONAL,
    &Decoder-transforms             #TRANSFORM ORDERED OPTIONAL,

    -- Кодирование открытого типа
    &Known-structure-encodings          #ENCODINGS OPTIONAL,
    &Unknown-structure                  OPTIONAL,
    &Unknown-structure-encodings        #ENCODINGS OPTIONAL,

    -- Заполнение и выравнивание значения (см. 22.8)
    &value-justification            Justification DEFAULT right:0,
    &value-pre-padding              Padding DEFAULT zero,
    &value-pre-pattern              Non-Null-Pattern DEFAULT bits:'0'B,
    &value-post-padding             Padding DEFAULT zero,
    &value-post-pattern             Non-Null-Pattern DEFAULT bits:'0'B,
    &unused-bits-determination      UnusedBitsDetermination
                                    DEFAULT field-to-be-set,
    &unused-bits-reference          REFERENCE OPTIONAL,
    &Unused-bits-encoder-transforms #TRANSFORM ORDERED OPTIONAL,
    &Unused-bits-decoder-transforms #TRANSFORM ORDERED OPTIONAL,

        -- Спецификация реверсии битов (см. 22.12)
        &bit-reversal  ReversalSpecification
                       DEFAULT no-reversal
} WITH SYNTAX {
    [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                    [ENCODED BY &replacement-structure-encoding-object]]
    [ALIGNED TO
            [NEXT]
            [ANY]
                    &encoding-space-pre-alignment-unit
                    [PADDING &encoding-space-pre-padding
```

[PATTERN &encoding-space-pre-pattern]]]
[START-POINTER &start-pointer
[MULTIPLE OF &start-pointer-unit]
[ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
ENCODING-SPACE
[SIZE &encoding-space-size
[MULTIPLE OF &encoding-space-unit]]
[DETERMINED BY &encoding-space-determination]
[USING &encoding-space-reference
[ENCODER-TRANSFORMS &Encoder-transforms]
[DECODER-TRANSFORMS &Decoder-transforms]]
[ENCODED WITH &Known-structure-encodings]
[UNKNOWN IS &Unknown-structure
[ENCODED WITH &Unknown-structure-encodings]]
[VALUE-PADDING
[JUSTIFIED &value-justification]
[PRE-PADDING &value-pre-padding
[PATTERN &value-pre-pattern]]
[POST-PADDING &value-post-padding
[PATTERN &value-post-pattern]]
[UNUSED BITS
[DETERMINED BY &unused-bits-determination]
[USING &unused-bits-reference
[ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
[DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
[EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
[AS &handle-value]]
[BIT-REVERSAL &bit-reversal]
}

23.10.2 Модель для кодирования классов в категории «открытый тип»

23.10.2.1 Модель кодирования открытого типа:

a) класс в категории «открытый тип» может быть заменен на другую структуру, если требуется обеспечить разграничение длины;

b) объект кодирования, определенный для данной категории, применяет «**ENCODED WITH**» набор объектов кодирования к типу, значение которого должно быть закодировано для открытого типа. Если «**ENCODED WITH**» нет, то используется текущий комбинированный набор объектов кодирования;

c) декодер будет запрашивать приложение для идентификации типа, закодированного в открытом типе. Приложение будет либо отвечать идентификацией типа, которая затем декодируется, или будет утверждать, что тип, закодированный в открытый тип, не может быть определен (ответ «unknown»);

d) если ответ «unknown» и присутствует «**UNKNOWN IS**», то декодер будет использовать «**UNKNOWN IS**» структуру и «**ENCODED WITH**» внутри «**UNKNOWN IS**» (если он присутствует), чтобы определить конец пространства кодирования;

e) если ответ «unknown» и «**UNKNOWN IS**» отсутствует, то размер пространства кодирования может быть определен за счет «**ENCODING-SPACE**» (см. 23.10.3.3), и декодер вернет приложению все биты, содержащиеся в определенных пространствах кодирования, за исключением значений предварительного и последующего заполнения.

23.10.2.2 В случае неизвестного декодирования декодер передаст биты, образующие неизвестное кодирование, приложению как значение открытого типа.

23.10.3 Цель и ограничения

23.10.3.1 Этот синтаксис используется для определения способа кодирования открытого типа и тех средств, что декодер использует для определения конца кодирования неизвестного типа в открытом типе.

23.10.3.2 Если «**REPLACE STRUCTURE**» установлен, то никакие другие параметры не устанавливаются.

23.10.3.3 Если «**ENCODING-SPACE SIZE**» равен «**self-delimiting**», то устанавливается «**UNKNOWN IS**».

23.10.4 Действия кодера

23.10.4.1 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) замена;

b) предварительное выравнивание и заполнение;

c) начальный указатель;

d) пкодирования (см. 23.10.4.3);

e) кодирование открытого типа (см. 23.10.4.2);

f) заполнение и выравнивание значения (см. 23.10.4.5);

g) реверсия битов.

23.10.4.2 Кодер должен закодировать значение типа, предоставленного приложением, используя «**ENCODED WITH**» набор объектов кодирования, если он присутствует, или же должен быть использован текущий комбинированный набор объектов кодирования.

23.10.4.3 Если «**ENCODING-SPACE SIZE**» равен «**variable-with-determinant**» или «**encode-option-withdeterminant**», то он должен быть минимальным числом единиц «**MULTIPLE OF**», необходимым для размещения комбинации (например, «s»), которая определяется в 23.10.4.5.

23.10.4.4 Кодер может (по своему выбору) увеличить «s» (определенный в 23.10.4.3) на единицы «**MULTIPLE OF**» (при условии любых ограничений, которые предписывают диапазон значений из «**added-field**» или «**asn1-field**»), если «**ENCODING-SPACE SIZE**» установлен в «**encoder-option-with-determinant**».

23.10.4.5 Если в пространстве кодирования имеются неиспользуемые биты, то «**VALUE-JUSTIFICATION**» должно быть установлено с использованием либо значений набора, либо значений по умолчанию.

23.10.5 Действия декодера

23.10.5.1 Для любой установленной группы признаков кодирования декодер выполняет декодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) пространство кодирования;

d) реверсия битов;

e) заполнение и выравнивание значения;

f) декодирование открытого типа (см. 23.10.5.2).

23.10.5.2 Для декодирования открытого типа декодер должен запросить у приложения тип, который был закодирован, и расшифровывать значение этого типа или «**UNKNOWN IS**» структуры в соответствии с «**ENCODED WITH**» спецификациями в «**UNKNOWN IS**».

23.10.5.3 Если декодирование было неизвестного типа, то биты, образующие неизвестное кодирование (без битов предварительного заполнения и без значения битов предварительного и последующего заполнений, если таковые имеются), должны быть переданы приложению как значение открытого типа.

23.11 Определение объектов кодирования для классов в категории «факультативные возможности»

23.11.1 Определенный синтаксис

Синтаксис с целью определения объектов кодирования для классов в категории «факультативные возможности» определяется следующим образом:

```
#OPTIONAL ::= ENCODING-CLASS {

    -- Спецификация замены исключительно структуры (см. 22.1)
    &#Replacement-structure
            OPTIONAL,
    &replacement-structure-encoding-object &#Replacement-structure OPTIONAL,
```

```
        -- Спецификация предварительного выравнивания и заполнения (см. 22.2)
        &encoding-space-pre-alignment-unit   Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &encoding-space-pre-padding          Padding DEFAULT zero,
        &encoding-space-pre-pattern          Non-Null-Pattern (ALL EXCEPT different:any)
                                             DEFAULT bits:'0'B,

        -- Спецификация начального указателя (см. 22.3)
        &start-pointer                       REFERENCE OPTIONAL,
        &start-pointer-unit                  Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &Start-pointer-encoder-transforms    #TRANSFORM ORDERED OPTIONAL,

        -- Определение факультативных возможностей (см. 22.5)
        &optionality-determination  OptionalityDetermination
                                    DEFAULT field-to-be-set,
        &optionality-reference      REFERENCE OPTIONAL,
        &Encoder-transforms         #TRANSFORM ORDERED OPTIONAL,
        &Decoder-transforms         #TRANSFORM ORDERED OPTIONAL,
        &handle-id                  PrintableString
                                    DEFAULT "default-handle"

    } WITH SYNTAX {
        [REPLACE
            [STRUCTURE]
            WITH &#Replacement-structure
                [ENCODED BY &replacement-structure-encoding-object]]
        [ALIGNED TO
            [NEXT]
            [ANY]
                &encoding-space-pre-alignment-unit
                [PADDING &encoding-space-pre-padding
                    [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER &start-pointer
            [MULTIPLE OF &start-pointer-unit]
            [ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
        PRESENCE
            [DETERMINED BY &optionality-determination
            [HANDLE &handle-id]]
          [USING &optionality-reference
            [ENCODER-TRANSFORMS &Encoder-transforms]
            [DECODER-TRANSFORMS &Decoder-transforms]]
}
```

23.11.2 Цель и ограничения

23.11.2.1 Этот синтаксис используется для определения кодирования класса в категории «факультативные возможности».

23.11.2.2 Если «**REPLACE STRUCTURE**» установлен, то никакие другие группы признаков кодирования не устанавливаются. Если объект кодирования структуры замены показывает описатель (с заданным набором значений описателя), определяемый объект кодирования показывает тот же идентификационный описатель (с тем же набором значений описателя — см. 22.1.1.11).

23.11.3 Действия кодера

23.11.3.1 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) замена (см. 23.11.3.2);

b) предварительное выравнивание и заполнение;

c) начальный указатель;

d) определение факультативных возможностей.

23.11.3.2 Если «**REPLACE STRUCTURE**» установлен, то полный компонент (включая любые классы в категории «тег», но исключая классы категории «факультативные возможности») предоставляется в качестве реального параметра для структуры замены, которая становится обязательным компонентом.

23.11.4 Действия декодера

23.11.4.1 Для любой установленной группы признаков кодирования декодер выполняет декодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) определение функциональных возможностей.

23.12 Определение объектов кодирования для классов в категории «pad»

23.12.1 Определенный синтаксис

Синтаксис с целью определения объектов кодирования для классов в категории «pad» определяется следующим образом:

```
#PAD ::= ENCODING-CLASS {

        -- Спецификация замены исключительно структуры (см. 22.1)
        &#Replacement-structure
                OPTIONAL,
        &replacement-structure-encoding-object &#Replacement-structure OPTIONAL,

        -- Спецификация предварительного выравнивания и заполнения (см. 22.2)
    &encoding-space-pre-alignment-unit  Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &encoding-space-pre-padding     Padding DEFAULT zero,
        &encoding-space-pre-pattern     Non-Null-Pattern (ALL EXCEPT different:any)
                                        DEFAULT bits:'0'B,

        -- Спецификация начального указателя (см. 22.3)
        &start-pointer                  REFERENCE OPTIONAL,
        &start-pointer-unit             Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

        -- Спецификация пространства кодирования (см. 22.4)
        &encoding-space-size            EncodingSpaceSize
                                        DEFAULT self-delimiting-values,
        &encoding-space-unit            Unit (ALL EXCEPT repetitions)
                                        DEFAULT bit,
        &encoding-space-determination   EncodingSpaceDetermination
                                        DEFAULT field-to-be-set,
        &encoding-space-reference       REFERENCE OPTIONAL,
        &Encoder-transforms             #TRANSFORM ORDERED OPTIONAL,
        &Decoder-transforms             #TRANSFORM ORDERED OPTIONAL,

        -- Кодирование значения
        &pad-pattern Pattern  (ALL EXCEPT different:any)
                              DEFAULT bits:"B,

        -- Спецификация идентификационного описателя (см. 22.9)
        &exhibited-handle  PrintableString DEFAULT "default-handle",
        &Handle-positions  INTEGER (0..MAX) OPTIONAL,
        &handle-value-set  HandleValueSet DEFAULT tag:any,
```

```
        -- Спецификация реверсии битов (см. 22.12)
        &bit-reversal  ReversalSpecification
                       DEFAULT no-reversal

} WITH SYNTAX {
        [REPLACE
                [STRUCTURE]
                WITH &#Replacement-structure
                        [ENCODED BY &replacement-structure-encoding-object]]
        [ALIGNED TO
                [NEXT]
                [ANY]
                        &encoding-space-pre-alignment-unit
                        [PADDING &encoding-space-pre-padding
                                [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER &start-pointer
                [MULTIPLE OF &start-pointer-unit]
                [ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
        ENCODING-SPACE
                 [SIZE &encoding-space-size
                        [MULTIPLE OF &encoding-space-unit]]
        [DETERMINED BY &encoding-space-determination]
        [USING &encoding-space-reference
                                [ENCODER-TRANSFORMS &Encoder-transforms]
                                [DECODER-TRANSFORMS &Decoder-transforms]]
          [PAD-PATTERN &pad-pattern]
          [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
                [AS &handle-value]]
          [BIT-REVERSAL &bit-reversal]
}
```

23.12.2 Цель и ограничения

23.12.2.1 Этот синтаксис используется для определения кодирования класса в категории «pad».

23.12.2.2 Если «**ENCODING-SPACE SIZE**» положителен, то «**PAD-PATTERN**» должен иметь ненулевую длину; он копируется и обрезается, чтобы заполнить пространство кодирования.

23.12.2.3 Если «**REPLACE STRUCTURE**» установлен, то никакая другая группа признаков кодирования не устанавливается. Если объект кодирования структуры замены показывает описатель (с заданным набором значений описателя), определяемый объект кодирования показывает тот же идентификационный описатель (с тем же набором значений описателя — см. 22.1.1.11).

23.12.3 Действия кодера

23.12.3.1 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) замена;

b) предварительное выравнивание и заполнение;

c) начальный указатель;

d) пространство кодирования;

e) кодирование значения (см. ниже);

f) идентификационный описатель;

g) реверсия битов.

23.12.3.2 Если «**ENCODING-SPACE SIZE**» положителен, то значением будет «**PAD-PATTERN**», который копируется и обрезается, чтобы заполнить пространство кодирования.

23.12.3.3 Если «**ENCODING-SPACE SIZE**» равен «**fixed-to-max**» или «**variable-with-determinant**», или «**encoder-option-with-determinant**», то размер пространства кодирования должен быть минимальным

числом единиц «**MULTIPLE OF**», которое превышает размер «**PAD-PATTERN**» (например, «s»), а «**PAD-PATTERN**» затем копируется и обрезается, чтобы заполнить это пространство (но см. 23.12.3.4).

Примечание — Это будет пустым пространством кодирования, если «**PAD-PATTERN**» является нулем.

23.12.3.4 Кодер может (по своему выбору) увеличить «s» (определенный в 23.12.3.3) на единицы «**MULTIPLE OF**» (при условии любых ограничений, которые предписывают диапазон значений из «**field-to-be-set**» или «**field-to-beused**»), если «**ENCODING-SPACE SIZE**» установлен в «**encoder-option-with-determinant**».

23.12.4 Действия декодера

23.12.4.1 Для любой установленной группы признаков кодирования декодер выполняет декодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) реверсия битов;

d) пространство кодирования.

23.12.4.2 Декодер определяет размер кодирования значения pad и опознает такие биты в кодировании, но пассивно принимает любое значение таких битов.

23.13 Определение объектов кодирования для классов в категории «повторение»

23.13.1 Определенный синтаксис

Синтаксис с целью определения объектов кодирования для классов в категории «повторение» определяется следующим образом:

```
#REPETITION ::= ENCODING-CLASS {

        -- Кодирование повторения
        &Repetition-encodings  #CONDITIONAL-REPETITION ORDERED OPTIONAL,
        &repetition-encoding   #CONDITIONAL-REPETITION OPTIONAL
} WITH SYNTAX {
        [REPETITION-ENCODINGS &Repetition-encodings]
        [REPETITION-ENCODING &repetition-encoding]

}
```

23.13.2 Цель и ограничения

23.13.2.1 Этот синтаксис используется для определения кодирования класса в категории «повторение» путем указания одного или нескольких кодирований класса **#CONDITIONAL-REPETITION**.

23.13.2.2 Должен устанавливаться только один из «**REPETITION-ENCODING**» и «**REPETITION-ENCODINGS**».

Примечание — Единственной целью разрешения «**REPETITION-ENCODING**» так же, как «**REPETITION-ENCODINGS**», является обеспечение синтаксиса, который не содержит двойных фигурных скобок («{{«) в общем случае одиночного условного кодирования. Использование «**REPETITION-ENCODINGS**», когда имеется одиночное условное кодирование, не одобряется, но разрешается.

23.13.2.3 Если объект кодирования в упорядоченном списке «**REPETITION-ENCODINGS**» определен с помощью «**IF**» или «**IF-ALL**», то все предыдущие объекты кодирования в этом списке должны быть определены с помощью «**IF**» или «**IF-ALL**».

23.13.3 Действия кодера

23.13.3.1 Кодер выбирает и применяет первый объект кодирования **#CONDITIONAL-REPETITION** в «**ENCODING(S)**», условия которого удовлетворяются. Спецификация ECN будет ошибочной, если ни одно условное кодирование не имеет условий, которые удовлетворяются.

Примечание — Может быть редкий, но неразрешенный случай, когда присутствуют объекты кодирования **#CONDITIONAL-REPETITION**, которые никогда не используются, так как условия при использовании более ранних объектов кодирования всегда могут удовлетворяться.

23.13.4 Действия декодера

23.13.4.1 Декодер выбирает и использует первый объект кодирования **#CONDITIONAL-REPETITION** в «**ENCODING(S)**», чьи условия удовлетворяются.

23.14 Определение объектов кодирования для класса #CONDITIONAL-REPETITION

23.14.1 Определенный синтаксис

Синтаксис с целью определения объектов кодирования для класса **#CONDITIONAL-REPETITION** определяется следующим образом:

```
#CONDITIONAL-REPETITION ::= ENCODING-CLASS {

        -- Условие (см. 21.12)
        &size-range-condition                   SizeRangeCondition OPTIONAL,
        &comparison                             Comparison OPTIONAL,
        &comparator                             INTEGER OPTIONAL,
        &Size-range-conditions                  SizeRangeCondition ORDERED OPTIONAL,
        &Comparisons                            Comparison ORDERED OPTIONAL,
        &Comparators                            INTEGER ORDERED OPTIONAL,

        -- Спецификация замены структуры или компонента (см. 22.1)
        &#Replacement-structure
                OPTIONAL,
        &replacement-structure-encoding-object &#Replacement-structure OPTIONAL,
        &#Head-end-structure                    OPTIONAL,

        -- Спецификация предварительного выравнивания и заполнения (см. 22.2)
    &encoding-space-pre-alignment-unit          Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &encoding-space-pre-padding             Padding DEFAULT zero,
        &encoding-space-pre-pattern             Non-Null-Pattern (ALL EXCEPT different:any)
                                                DEFAULT bits:'0'B,

        -- Спецификация начального указателя (см. 22.3)
        &start-pointer                          REFERENCE OPTIONAL,
        &start-pointer-unit                     Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &Start-pointer-encoder-transforms       #TRANSFORM ORDERED OPTIONAL,

        -- Спецификация пространства повторения (см. 22.7)
        &repetition-space-size                  EncodingSpaceSize
                                                DEFAULT self-delimiting-values,
        &repetition-space-unit                  Unit
                                                DEFAULT bit,
        &repetition-space-determination         RepetitionSpaceDetermination
                                                DEFAULT field-to-be-set,
        &main-reference                         REFERENCE OPTIONAL,
        &Encoder-transforms                     #TRANSFORM ORDERED OPTIONAL,
        &Decoder-transforms                     #TRANSFORM ORDERED OPTIONAL,
        &handle-id                              PrintableString
                                                DEFAULT "default-handle",
        &termination-pattern                    Non-Null-Pattern (ALL EXCEPT
                                                different:any) DEFAULT bits '0'B,

        -- Выравнивание повторения
        &repetition-alignment           ENUMERATED {none, aligned}
                                        DEFAULT none,
```

```
        -- Заполнение и выравнивание значения (см. 22.8)
        &value-justification            Justification DEFAULT right:0,
        &value-pre-padding              Padding DEFAULT zero,
        &value-pre-pattern              Non-Null-Pattern DEFAULT bits:'0'B,
        &value-post-padding             Padding DEFAULT zero,
        &value-post-pattern             Non-Null-Pattern DEFAULT bits:'0'B,
        &unused-bits-determination      UnusedBitsDetermination
                                        DEFAULT field-to-be-set,
    &unused-bits-reference              REFERENCE OPTIONAL,
    &Unused-bits-encoder-transforms     #TRANSFORM ORDERED OPTIONAL,
    &Unused-bits-decoder-transforms     #TRANSFORM ORDERED OPTIONAL,

    -- Спецификация идентификационного описателя (см. 22.9)
    &exhibited-handle PrintableString   DEFAULT "default-handle",
    &Handle-positions                   INTEGER (0..MAX) OPTIONAL,
    &handle-value-set                   HandleValueSet DEFAULT tag:any,

    -- Спецификация реверсии битов (см. 22.12)
    &bit-reversal  ReversalSpecification
                   DEFAULT no-reversal

} WITH SYNTAX {
    [IF &size-range-condition [&comparison &comparator]]
    [IF-ALL &Size-range-conditions [&Comparisons &Comparators]]
    [ELSE]
    [REPLACE
            [STRUCTURE]
            [COMPONENT]
            [ALL COMPONENTS]
            WITH &Replacement-structure
            [ENCODED BY &replacement-structure-encoding-object
                    [INSERT AT HEAD &#Head-end-structure]]]
    [ALIGNED TO
            [NEXT]
            [ANY]
                    &encoding-space-pre-alignment-unit
                    [PADDING &encoding-space-pre-padding
                            [PATTERN &encoding-space-pre-pattern]]]
    [START-POINTER &start-pointer
            [MULTIPLE OF &start-pointer-unit]
            [ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
    REPETITION-SPACE
            [SIZE &repetition-space-size
                    [MULTIPLE OF &repetition-space-unit]]
                [DETERMINED BY &repetition-space-determination
                        [HANDLE &handle-id]]
                [USING &main-reference
                        [ENCODER-TRANSFORMS &Encoder-transforms]
                        [DECODER-TRANSFORMS &Decoder-transforms]]
                [PATTERN &termination-pattern]
        [ALIGNMENT &repetition-alignment]
        [VALUE-PADDING
                [JUSTIFIED &value-justification]
                [PRE-PADDING &value-pre-padding
                        [PATTERN &value-pre-pattern]]
                [POST-PADDING &value-post-padding
```

[PATTERN &value-post-pattern]]
[UNUSED BITS
[DETERMINED BY &unused-bits-determination]
[USING &unused-bits-reference
[ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
[DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
[EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
[AS &handle-value]]
[BIT-REVERSAL &bit-reversal]
}

23.14.2 Цель и ограничения

23.14.2.1 Этот синтаксис используется для кодирования класса в категории «повторение», подлежащей удовлетворению условия, основанного на границах повторения (с помощью «**IF**»). Он допускает также спецификацию того, что весь набор условий подлежит удовлетворению (применение «**IF-ALL**»). Он допускает также спецификацию того, что нет условия. Использование «**ELSE**» или пропуск «**IF**», «**IF-ALL**» и «**ELSE**» указывают, что нет условия. «**IF-ALL**» должен использоваться с тремя списками, если один или более size-range-conditions требуют сравнения, или же с одним списком в противном случае. При использовании трех списков size-range-conditions, которые не требуют сравнения или компаратора (если таковые имеются), должны следовать всем тем, что требуют сравнения, и не должны иметь соответствующей записи во втором и третьем списке. При использовании «**IF-ALL**» с тремя списками, списки должны быть интерпретированы как списки предикатов, использующие значения в соответствующих позициях в трех списках.

Примечание — Рекомендуется, чтобы три списка были отформатированы для предоставления условия в каждом столбце (см. пример в 23.7.2.2).

23.14.2.2 Должен присутствовать один самый больший из «**IF**», «**IF-ALL**» и «**ELSE**».

23.14.2.3 Если «**REPLACE STRUCTURE**» установлен, то никакие другие группы признаков кодирования не устанавливаются. Если объект кодирования структуры замены показывает описатель (с заданным набором значений описателя), определяемый объект кодирования показывает тот же идентификационный описатель (с тем же набором значений описателя — см. 22.1.1.11).

23.14.2.4 Если «**EXHIBITS HANDLE**» установлен, то это предполагает, что объект кодирования показывает определенный идентификационный описатель.

23.14.2.5 «**REPETITION-SPACE SIZE**» не должен быть «**fixed-to-max**».

23.14.2.6 Если «**REPETITION-SPACE SIZE**» равен «**self-delimiting-values**», а «**MULTIPLE OF**» равен «**repetitions**», то число повторений должно ограничиваться границами до одного значения.

23.14.2.7 Если в пространстве кодирования имеются неиспользуемые биты, то должно быть установлено «**VALUEPADDING**».

23.14.3 Действия кодера

23.14.3.1 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) замена;
b) предварительное выравнивание и заполнение;
c) начальный указатель;
d) пространство повторения;
e) кодирование повторения (см. 23.14.3.4);
f) заполнение и выравнивание значения;
g) идентификационный описатель;
h) реверсия битов.

23.14.3.2 Если «**ALIGNMENT**» установлен в «**aligned**», то используются установки предварительного выравнивания и заполнения, чтобы предварительно выровнять каждое кодирование компонента.

Примечание — Это выполняется перед любым предварительным выравниванием, которое указано компонентом.

23.14.3.3 К полным кодированиям компонентов (с любым предварительным выравниванием, указанным любым способом) применяется конкатенация для формирования битов значения повторения.

23.14.3.4 Если «**REPETITION-SPACE SIZE**» равен «**variable-with-determinant**» или «**encoder-optionwith-determinant**», то размером будет наименьшее кратное единицам «**MULTIPLE OF**» (например, «s»), которое вместит значение повторения (но см. 23.14.3.5).

23.14.3.5 Кодер может (по своему выбору) увеличить «s» (определенный в 23.14.3.4) на единицы «**MULTIPLE OF**» (при условии любых ограничений, которые предписывает диапазон значений из «**field-to-be-set**» или «**field-tobe-used**»), если «**ENCODING-SPACE SIZE**» установлен в «**encoder-option-with-determinant**».

23.14.3.6 Значение повторения затем помещается в пространстве кодирования, используя «**VALUE-PADDING**», если имеются неиспользуемые биты.

23.14.4 Действия декодера

23.14.4.1 Для любой установленной группы признаков кодирования декодер выполняет действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;
b) начальный указатель;
c) пространство повторения;
d) реверсия битов;
e) заполнение и выравнивание значения;
f) декодирование повторения (см. 23.14.4.2).

23.14.4.2 Каждое повторение выделяется и декодируется согласно спецификации кодирования компонента класса «повторение».

23.15 Определение объектов кодирования для классов в категории «тег»

23.15.1 Определенный синтаксис

Синтаксис с целью определения объектов кодирования для классов в категории «тег» определяется следующим образом:

```
#TAG ::= ENCODING-CLASS {

        -- Спецификация замены исключительно структуры (см. 22.1)
        &#Replacement-structure
                OPTIONAL,
        &replacement-structure-encoding-object &#Replacement-structure OPTIONAL,

        -- Спецификация предварительного выравнивания и заполнения (см. 22.2)
&encoding-space-pre-alignment-unit   Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &encoding-space-pre-padding  Padding DEFAULT zero,
        &encoding-space-pre-pattern  Non-Null-Pattern (ALL EXCEPT different:any)
                                     DEFAULT bits:'0'B,

        -- Спецификация начального указателя (см. 22.3)
        &start-pointer                    REFERENCE OPTIONAL,
        &start-pointer-unit               Unit (ALL EXCEPT repetitions) DEFAULT bit,
        &Start-pointer-encoder-transforms #TRANSFORM ORDERED OPTIONAL,

        -- Спецификация пространства кодирования (см. 22.4)
        &encoding-space-size              EncodingSpaceSize
                                          DEFAULT self-delimiting-values,
        &encoding-space-unit              Unit (ALL EXCEPT repetitions)
                                          DEFAULT bit,
        &encoding-space-determination     EncodingSpaceDetermination
                                          DEFAULT field-to-be-set,
        &encoding-space-reference         REFERENCE OPTIONAL,
        &Encoder-transforms               #TRANSFORM ORDERED OPTIONAL,
        &Decoder-transforms               #TRANSFORM ORDERED OPTIONAL,
```

```
        -- Заполнение и выравнивание значения (см. 22.8)
        &value-justification        Justification DEFAULT right:0,
        &value-pre-padding          Padding DEFAULT zero,
        &value-pre-pattern          Non-Null-Pattern DEFAULT bits:'0'B,
        &value-post-padding         Padding DEFAULT zero,
        &value-post-pattern         Non-Null-Pattern DEFAULT bits:'0'B,
          &unused-bits-determination          UnusedBitsDetermination
                                              DEFAULT field-to-be-set,
          &unused-bits-reference              REFERENCE OPTIONAL,
          &Unused-bits-encoder-transforms     #TRANSFORM ORDERED OPTIONAL,
          &Unused-bits-decoder-transforms     #TRANSFORM ORDERED OPTIONAL,

        -- Спецификация идентификационного описателя (см. 22.9)
          &exhibited-handle     PrintableString DEFAULT "default-handle",
          &Handle-positions     INTEGER (0..MAX) OPTIONAL,
          &handle-value-set     HandleValueSet DEFAULT tag:any,

        -- Спецификация реверсии битов (см. 22.12)
          &bit-reversal  ReversalSpecification
                         DEFAULT no-reversal

} WITH SYNTAX {
        [REPLACE
                [STRUCTURE]
                WITH &#Replacement-structure
                        [ENCODED BY &replacement-structure-encoding-object]]
        [ALIGNED TO
                [NEXT]
                [ANY]
                        &encoding-space-pre-alignment-unit
                        [PADDING &encoding-space-pre-padding
                                [PATTERN &encoding-space-pre-pattern]]]
        [START-POINTER &start-pointer
                [MULTIPLE OF &start-pointer-unit]
                [ENCODER-TRANSFORMS &Start-pointer-encoder-transforms]]
        ENCODING-SPACE
                [SIZE &encoding-space-size
                        [MULTIPLE OF &encoding-space-unit]]
                [DETERMINED BY &encoding-space-determination]
                [USING &encoding-space-reference
                        [ENCODER-TRANSFORMS &Encoder-transforms]
                [DECODER-TRANSFORMS &Decoder-transforms]]
[VALUE-PADDING
          [JUSTIFIED &value-justification]
          [PRE-PADDING &value-pre-padding
                [PATTERN &value-pre-pattern]]
          [POST-PADDING &value-post-padding
                [PATTERN &value-post-pattern]]
          [UNUSED BITS
                [DETERMINED BY &unused-bits-determination]
                [USING &unused-bits-reference
                        [ENCODER-TRANSFORMS &Unused-bits-encoder-transforms]
                        [DECODER-TRANSFORMS &Unused-bits-decoder-transforms]]]]
  [EXHIBITS HANDLE &exhibited-handle AT &Handle-positions
        [AS &handle-value]]
  [BIT-REVERSAL &bit-reversal]
}
```

23.15.2 Цель и ограничения

23.15.2.1 Этот синтаксис используется для определения кодирования класса в категории «тег».

23.15.2.2 Если «**REPLACE STRUCTURE**» установлен, то никакие другие спецификации не устанавливаются. Если объект кодирования структуры замены показывает описатель (с заданным набором значений описателя), определяемый объект кодирования показывает тот же идентификационный описатель (с тем же набором значений описателя — см. 22.1.1.11).

23.15.2.3 «**ENCODING-SPACE SIZE**» не должен быть «**fixed-to-max**» или «**self-delimiting-values**».

Примечание — Это означает, что значение по умолчанию (устанавливается для обеспечения совместимости с другими использованиями этого типа) всегда должно быть переопределено.

23.15.3 Действия кодера

23.15.3.1 Для любой установленной группы признаков кодирования кодер выполняет кодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) замена;

b) предварительное выравнивание и заполнение;

c) начальный указатель;

d) пространство кодирования;

e) кодирование значения (см. 23.15.3.3);

f) заполнение и выравнивание значения;

g) идентификационный описатель;

h) реверсия битов.

23.15.3.2 Кодер определяет минимальное число битов «n», нужное для кодирования номера тега, как такое наименьшее значение «n», при котором $2^n - 1$ будет не менее номера тега. Если «n» равно нулю, то оно увеличивается до 1.

23.15.3.3 Это кодирование должно быть кодированием положительного целого числа. Спецификация кодирования положительного целого числа приведена в ИСО/МЭК 8825-1, пункты 8.3.2 и 8.3.3.

23.15.3.4 Кодер обнаруживает ошибку спецификации ECN, когда номер тега должен быть закодирован в число битов, которое недостаточно в сравнении с указанным выше.

23.15.3.5 Если «**ENCODING-SPACE SIZE**» является положительным целым числом, то его размер в битах вычисляется как «**SIZE**, умноженный на единицы «**MULTIPLE OF**». Если «**VALUE-PADDING**» не установлен, то это будет числом битов «n», в которое кодируется номер тега, а неиспользуемых битов не будет. Если «**VALUE-PADDING**» установлено, то число битов, в которое кодируется номер тега, уменьшается на целочисленное значение «m», указанное для «**JUSTIFIED**», и будет «m» неиспользуемых битов.

23.15.3.6 Если «**ENCODING-SPACE SIZE**» равен «**variable-with-determinant**» или «**encoder-option-withdeterminant**», то кодер определяет минимальное число единиц «**MULTIPLE OF**», которое имеет достаточно битов для кодирования номера тега (например, «s»), и действует далее (как описано выше), считая «**SIZE**» положительным целым числом, установленным в это значение (но см. 23.15.3.7).

23.15.3.7 Кодер может (по своему выбору) увеличить «s» (определенное в 23.15.3.6) на единицы «**MULTIPLE OF**» (при условии любых ограничений, которые предписывают диапазон значений из «**field-to-be-set**» или «**field-to-beused**»), если «**ENCODING-SPACE SIZE**» установлен в «**encoder-option-with-determinant**».

23.15.4 Действия декодера

23.15.4.1 Для любой установленной группы признаков кодирования декодер выполняет декодирующие действия, указанные в разделе 22, в следующей последовательности и согласно определению объекта кодирования:

a) предварительное выравнивание и заполнение;

b) начальный указатель;

c) пространство кодирования;

d) реверсия битов;

e) заполнение и выравнивание значения;

f) декодирование значения.

23.15.4.2 Декодер восстанавливает номер тега из битов, которые использованы для его кодирования, декодируя кодовую последовательность положительного целого числа.

23.16 Определение объектов кодирования для классов в других категориях

В настоящем стандарте отсутствуют определенные синтаксисы для классов в следующих категориях:
- «идентификатор объекта» (**objectidentifier**);
- «открытый тип» (**opentype**);
- «действительное число» (**real**);
- «время» (**time**).

24 Спецификация определенного синтаксиса для класса кодирования #TRANSFORM

24.1 Сводный перечень признаков кодирования и определенного синтаксиса

24.1.1 Синтаксисом при определении объектов кодирования для класса #TRANSFORM будет следующий:

```
#TRANSFORM ::= ENCODING-CLASS {

        -- int-to-int (см. 24.3)
        &int-to-int     CHOICE
                        {increment          INTEGER (1..MAX),
                        decrement           INTEGER (1..MAX),
                        multiply            INTEGER (2..MAX),
                        divide              INTEGER (2..MAX),
                        negate              ENUMERATED{value},
                        modulo              INTEGER (2..MAX),
                        subtract            ENUMERATED{lower-bound}
                        } OPTIONAL,

        -- bool-to-bool (см. 24.4)
        &bool-to-bool   CHOICE
                        {logical            ENUMERATED{not}}
                        DEFAULT logical:not,

        -- bool-to-int (см. 24.5)
        &bool-to-int                ENUMERATED {true-zero, true-one}
                                    DEFAULT true-one,

        -- int-to-bool (см. 24.6)
        &int-to-bool                ENUMERATED {zero-true, zero-false}
                                    DEFAULT zero-false,
        &Int-to-bool-true-is        INTEGER OPTIONAL,
        &Int-to-bool-false-is       INTEGER OPTIONAL,

        -- int-to-chars (см. 24.7)
        &int-to-chars-size      ResultSize DEFAULT variable,
        &int-to-chars-plus      BOOLEAN DEFAULT FALSE,
        &int-to-chars-pad       ENUMERATED
        {spaces, zeros}         DEFAULT zeros,

        -- int-to-bits (см. 24.8)
&int-to-bits-encoded-as         ENUMERATED
                                {positive-int, twos-complement}
                                DEFAULT twos-complement,
&int-to-bits-unit               Unit (1..MAX) DEFAULT bit,
&int-to-bits-size               ResultSize DEFAULT variable,
```

```
        -- bits-to-int (см. 24.9)
        &bits-to-int-decoded-assuming    ENUMERATED
                                         positive-int, twos-complement}
                                         DEFAULT twos-complement,

        -- char-to-bits (см. 24.10)
        &char-to-bits-encoded-as    ENUMERATED
                                    {iso10646, compact, mapped}
        DEFAULT compact,
        &Char-to-bits-chars         UniversalString (SIZE(1))
                                    ORDERED OPTIONAL,
        &Char-to-bits-values        BIT STRING ORDERED OPTIONAL,
        &char-to-bits-unit          Unit (1..MAX) DEFAULT bit,
        &char-to-bits-size          ResultSize DEFAULT variable,

        -- bits-to-char (см. 24.11)
        &bits-to-char-decoded-assuming  ENUMERATED
                                        {iso10646, mapped}
                                        DEFAULT iso10646,
        &Bits-to-char-values            BIT STRING ORDERED OPTIONAL,
        &Bits-to-char-chars             UniversalString (SIZE(1))
                                        ORDERED OPTIONAL,

        -- bit-to-bits (см. 24.12)
        &bit-to-bits-one            Non-Null-Pattern DEFAULT bits:'1'B,
        &bit-to-bits-zero           Non-Null-Pattern DEFAULT bits:'0'B,

        -- bits-to-bits (см. 24.13)
        &Source-values              BIT STRING ORDERED,
            &Result-values          BIT STRING ORDERED,

            -- chars-to-composite-char (см. 24.14)
            -- Признаки кодирования для этого преобразования отсутствуют.

            -- bits-to-composite-bits (см. 24.15)
            &bits-to-composite-bits-unit    Unit (1..MAX) DEFAULT bit

            -- octets-to-composite-bits (см. 24.16)
            -- Признаки кодирования для этого преобразования отсутствуют.

            -- composite-char-to-chars (см. 24.17)
            -- Признаки кодирования для этого преобразования отсутствуют.

            -- composite-bits-to-bits (см. 24.18)
            -- Признаки кодирования для этого преобразования отсутствуют.

            -- composite-bits-to-octets (см. 24.19)
            -- Признаки кодирования для этого преобразования отсутствуют.

    } WITH SYNTAX {

            -- Может использоваться только один из следующих классов.
            [INT-TO-INT &int-to-int]
            [BOOL-TO-BOOL [AS &bool-to-bool]]
            [BOOL-TO-INT AS &bool-to-int]
            [INT-TO-BOOL
                [AS &int-to-bool]
```

```
                [TRUE-IS &Int-to-bool-true-is]
                [FALSE-IS &Int-to-bool-false-is]]
            [INT-TO-CHARS
                [SIZE &int-to-chars-size]
                [PLUS-SIGN &int-to-chars-plus]
                [PADDING &int-to-chars-pad]]
            [INT-TO-BITS
                [AS &int-to-bits-encoded-as]
                [SIZE &int-to-bits-size]
                [MULTIPLE OF &int-to-bits-unit]]
        [BITS-TO-INT
                [AS &bits-to-int-decoded-assuming]]
        [CHAR-TO-BITS
                [AS &char-to-bits-encoded-as]
                [CHAR-LIST &Char-to-bits-chars]
                [BITS-LIST &Char-to-bits-values]
                [SIZE &char-to-bits-size]
                [MULTIPLE OF &char-to-bits-unit]]
        [BITS-TO-CHAR
                [AS &bits-to-char-decoded-assuming]
                [BITS-LIST &Bits-to-char-values]
                [CHAR-LIST &Bits-to-char-chars]]
        [BIT-TO-BITS
                [ZERO-PATTERN &bit-to-bits-zero]
                [ONE-PATTERN &bit-to-bits-one]]
        [BITS-TO-BITS
                SOURCE-LIST &Source-values
                RESULT-LIST &Result-values]
        [CHARS-TO-COMPOSITE-CHAR]
        [BITS-TO-COMPOSITE-BITS
                [UNIT &bits-to-composite-bits-unit]]
        [OCTETS-TO-COMPOSITE-BITS]
        [COMPOSITE-CHAR-TO-CHARS]
        [COMPOSITE-BITS-TO-BITS]
        [COMPOSITE-BITS-TO-OCTETS]
}
```

24.2 Источник и цель преобразователей

24.2.1 Класс кодирования **#TRANSFORM** позволяет специфицировать процедуры, которые преобразуют входные абстрактные значения (источник) в выходные абстрактные значения того же или другого типа (результат). Он позволяет также специфицировать процедуры, которые отображают источник «цепочка знаков», «цепочка октетов» или «цепочка битов» в преобразовательную смесь, а преобразовательную смесь (значениями которой являются одиночный знак, одиночный октет или цепочки битов с фиксированным размером единицы) — в абстрактное значение (цепочку знаков, цепочку октетов или цепочку битов). Источник является либо результатом предыдущего преобразователя, либо полученным из класса источника (см. 19.4). Результат является либо источником для последующего преобразователя, либо становится связанным с классом цели (см. 19.4).

Примечание — В разделе 23 использованы также преобразователи, источниками которых являются одиночный бит и одиночный знак.

24.2.2 Эти преобразователи используются в определении отображений значения и в определении объектов кодирования для классов кодирования в группе категорий «битовое поле» (см. разделы 20—23).

24.2.3 Источник и результат указываются словами («**INT-TO-INT**», «**BOOL-TO-BOOL**» и т. п.) в спецификации объекта кодирования **#TRANSFORM** и определяются в соответствующем тексте.

24.2.4 В пунктах 24.2.4.1—24.2.4.3 описываются правила для последовательного использования преобразователей и для классов источника и цели из списка преобразователей.

24.2.4.1 Когда объекты кодирования класса **#TRANSFORM** указаны в упорядоченном списке, источник последующего объекта кодирования **#TRANSFORM** должен быть результатом предыдущего объекта кодирования **#TRANSFORM**.

24.2.4.2 Для первого и последнего элементов упорядоченного списка преобразователей, использованных при определении объектов кодирования в разделах 22 и 23, тексты в этих разделах определяют источник для первого преобразователя и требуемый результат для последнего преобразователя.

24.2.4.3 Для первого и последнего элементов упорядоченного списка преобразователей, использованных при спецификации отображения значения преобразователями в 19.4, текст в этом подразделе определяет класс источника и класс цели, которые оба будут в категории «цепочка битов», «булева», «цепочка знаков», «целочисленная» или «цепочка октетов» (см. 19.4.2). Требуемый источник для первого преобразователя и требуемый результат для последнего преобразователя (для каждой из этих категорий) определяются в 24.2.7.

24.2.5 Текст в настоящем пункте определяет источник преобразователя и результат преобразователя в виде целого числа, булева значения, цепочки знаков, цепочки битов, одиночного знака или одиночного бита (только источник). Источник и результат преобразователя могут быть также смесью этих значений. Преобразовательные смеси могут создаваться только преобразователями и должны создаваться другим (следующим) преобразователем из списка преобразователей. Имеются две группы преобразователей: рассчитанные на создание смесей из абстрактных значений или на создание абстрактного значения из смеси, а также рассчитанные на преобразование одиночных значений. Последние могут также преобразовывать смеси таких значений, создавая смесь в виде результата, который является преобразованием каждого элемента в смеси источника.

24.2.6 Источник или цель, являющиеся одиночным битом или одиночным знаком, появляются только тогда, когда успешные преобразователи имеют их в качестве выходного или входного элемента, либо как указано в разделах 22 и 23. Первый преобразователь упорядоченного списка, указанного в 19.4, не должен иметь источник в виде одиночного бита или одиночного знака. Последний преобразователь упорядоченного списка, указанного в 19.4, не должен иметь цель в виде одиночного бита или одиночного знака.

24.2.7 При использовании в 19.4 источник первого преобразователя и цель последнего преобразователя должны быть в той же категории, что и класс кодирования источника и класс кодирования цели (соответственно), за исключением следующих случаев. Когда категорией класса кодирования источника является «цепочка октетов», источником для первого преобразователя должна быть цепочка битов (обрабатывая каждое значение цепочки октетов как значение цепочки битов). Когда последним преобразователем является «**BITS-TO-BITS**» с «**MULTIPLE OF**», установленным в разделе 8, классом цели может быть «цепочка октетов».

24.2.8 В последующих подразделах описываются такие условия для абстрактных значений источника, которые дают возможность определять преобразователи как обратимые. ECN или применение будут ошибочными, когда такие значения подаются к преобразователю, который должен быть обратимым, а кодеры не генерируют кодовых последовательностей для таких значений.

24.3 Преобразователь int-to-int

Примечание — Примеры этого преобразователя приведены в D.1.2.2.

24.3.1 Преобразователь int-to-int использует следующий признак кодирования:

```
&int-to-int     CHOICE
                increment       INTEGER (1..MAX),
                decrement       INTEGER (1..MAX),
                multiply        INTEGER (2..MAX),
                divide          INTEGER (2..MAX),
                negate          ENUMERATED{value},
                modulo          INTEGER (2..MAX),
                subtract        ENUMERATED{lower-bound}
                mapping         IntegerMapping
                } OPTIONAL
```

24.3.2 Синтаксисом для преобразователя int-to-int будет следующий:

```
[INT-TO-INT &int-to-int]
```

24.3.3 Определение типа, используемого в int-to-int преобразовании:

```
IntegerMapping ::= SET OF SEQUENCE {
        source SET OF INTEGER,
        result INTEGER} (CONSTRAINED BY {/* пересечение исходных
                                          компонентов должно
                                          быть пустым (см. 21.17) */})
```

24.3.4 Как источник, так и результат этого преобразователя является целым числом или смесью целых чисел. Отсутствуют границы, связанные с результатом, если он не является последним преобразователем в отображении с помощью преобразователей (см. 19.4) (в котором имеется в виду, что ни источник, ни цель не могут быть смесью), а класс цели в отображении с помощью преобразователей не имеет границ. В этом случае спецификация ECN или приложение будут ошибочными, когда к преобразователю подаются целочисленные значения источника, которые не преобразуются в границы класса цели.

24.3.5 Преобразование int-to-int определяется путем установки значения в «**INT-TO-INT**», что позволяет в любом заданном объекте кодирования указывать точно одну арифметическую операцию. Общая арифметика может, однако, определяться с помощью упорядоченного списка преобразователей (это разрешается везде, где допускаются преобразователи, охватывающие целые числа).

24.3.6 Значения «**increment:n**», «**decrement:n**», «**multiply:n**», «**negate:n**» имеют свой обычный математический смысл.

24.3.7 Значение «**divide:n**» определено для образования целочисленного результата, который является целочисленным значением, наиболее близким к математическому результату, но не более удаленным от нуля, чем этот результат. В терминах программирования «**divide:n**» округляет в направлении нуля так, что значение 1 с «**divide:2**» даст нуль.

24.3.8 Преобразователь для значения «**modulo:n**» определяется следующим образом. Пусть «i» будет исходным целочисленным значением, а преобразователем будет «**modulo:n**». Пусть «j» будет результатом применения «**divide:n**», за которым следует «**multiply:n**», к «i». Тогда «**modulo:n**», примененный к «i», определяется как то же, что и применение «**decrement:j**» к «i».

24.3.9 Преобразователь для значения «**subtract:lower-bound**» используется только как первый в упорядоченном списке преобразователей (и поэтому не может использоваться, если источник является смесью). Источник должен иметь нижнюю границу.

24.3.10 Преобразователь для значения «**mapping:integerMapping**» определяется следующим образом. Исходное целочисленное значение заменяется на значение, связанное с набором значений, к которому оно принадлежит. Если пересечение наборов значений не пусто, то это является ошибкой спецификации ECN; если же исходное целое число не относится ни к одному из наборов значений, то это ошибка приложения.

24.3.11 Каждый из этих преобразователей определяется, чтобы стать обратимым, если источник является одиночным значением, а не смесью, и если удовлетворяется условие для абстрактного значения (к которому он применяется), приведенное в таблице 6. Он определяется также, чтобы стать обратимым, если источник является смесью, а в таблице 6 условием является *Всегда обратим*.

Примечание — Хотя преобразователь int-to-int с композитным входом формально обратим, если в таблице 6 указано Always reversible, на практике он не может являться частью цепи обратимых преобразователей, поскольку нет такой цепи, которая бы начиналась с некомпозитного входа и производила бы композитное целое число (с текущими описанными преобразователями).

Таблица 6 — Обратимость преобразователей «**INT-TO-INT**»

Преобразователь	Условие
increment:n	*Всегда обратим*
decrement:n	*Всегда обратим*
multiply:n	*Всегда обратим*
divide:n	*Значение кратно n*
negate:value	*Всегда обратим*
modulo:n	*Никогда не обратим*
subtract:lower-bound	*Всегда обратим*
mapping:integerMapping	*Исходные наборы значений, каждый из которых содержит только одно значение, и результирующие значения различны*

24.4 Преобразователь bool-to-bool

24.4.1 Преобразователь **BOOL-TO-BOOL** использует следующий признак кодирования:

```
&bool-to-bool     CHOICE
                  {logical      ENUMERATED{not}}
                  DEFAULT logical:not
```

24.4.2 Синтаксисом для преобразователя bool-to-bool будет следующий:

```
[BOOL-TO-BOOL [AS &bool-to-bool]]
```

24.4.3 Как источник, так и результат этого преобразователя является булевым значением или смесью булевых значений.

24.4.4 Если источник является булевым значением, то и результат является булевым значением. Если источник является смесью булевых значений, то результат является смесью булевых значений, в которой каждый элемент источника был преобразован согласно 24.4.5.

24.4.5 Имеется только одно значение «**BOOL-TO-BOOL**», а именно «**AS logical:not**», которое может быть опущено. Этот преобразователь превращает булево **TRUE** в **FALSE** и наоборот.

24.4.6 Этот преобразователь определен так, чтобы быть обратимым для всех абстрактных значений.

24.5 Преобразователь bool-to-int

24.5.1 Преобразователь bool-to-int использует следующий признак кодирования:

```
&bool-to-int    ENUMERATED {true-zero, true-one}
                DEFAULT true-one
```

24.5.2 Синтаксисом для преобразователя bool-to-int будет следующий.

```
[BOOL-TO-INT AS &bool-to-int]
```

24.5.3 Источником для этого преобразователя является булево значение или смесь булевых значений, а результат является целым числом или смесью целых чисел. Целочисленный результат (и каждый элемент в смеси целых чисел) имеет значение нуль или единица. Результат не имеет связанных с ним границ.

24.5.4 Если источник является булевым значением, то результат является целым числом. Если источник является смесью булевых значений, то результат является смесью целых чисел, в которой каждый элемент источника был преобразован согласно 24.5.5.

24.5.5 Значение «**true-zero**» преобразователя «**BOOL-TO-INT**» создает целое число 0 для **TRUE** и целое число 1 для **FALSE**. Значение «**true-one**» создает целое число 1 для **TRUE** и целое число 0 для **FALSE**.

24.5.6 Этот преобразователь определен так, чтобы быть обратимым для всех абстрактных значений.

24.6 Преобразователь int-to-bool

24.6.1 Преобразователь int-to-bool использует следующие признаки кодирования:

```
&int-to-bool            ENUMERATED {zero-true, zero-false}
                        DEFAULT zero-false,
&Int-to-bool-true-is    INTEGER OPTIONAL,
&Int-to-bool-false-is   INTEGER OPTIONAL
```

24.6.2 Синтаксисом для преобразователя int-to-bool будет следующий:

```
[INT-TO-BOOL
        [AS &int-to-bool]
        [TRUE-IS &Int-to-bool-true-is]
        [FALSE-IS &Int-to-bool-false-is]]
```

24.6.3 Источником для этого преобразователя является целое число или смесь целых чисел, а результатом является булево значение или смесь булевых значений.

24.6.4 Либо устанавливается один из «**AS**», «**TRUE-IS**» и «**FALSE IS**», либо устанавливаются оба «**TRUE-IS**» и «**FALSE-IS**» (а «**AS**» не устанавливается), либо не устанавливается ничего. Если не устанавливается ничего, то предполагается безусловное значение (по умолчанию) «**AS**».

24.6.5 Если устанавливается (возможно, по умолчанию) «**AS**», то значение «**zero-true**» создает **TRUE** для значения нуль и **FALSE** для всех ненулевых значений, а значение «**zero-false**» создает **FALSE** для значения нуль и **TRUE** для всех ненулевых значений.

24.6.6 Если устанавливается только «**TRUE-IS**», то все целочисленные значения для «**TRUE-IS**» создают **TRUE**, а все другие целочисленные значения создают **FALSE**.

24.6.7 Если устанавливается только «**FALSE-IS**», то все целочисленные значения для «**FALSE-IS**» создают **FALSE**, а все другие целочисленные значения создают **TRUE**.

24.6.8 Если устанавливаются оба «**TRUE-IS**» и «**FALSE-IS**», то целочисленные значения в «**TRUE-IS**» и «**FALSEIS**» должны быть непересекающимися. В этом случае спецификация ECN или применение будут ошибочными, когда абстрактные значения, которые не включены ни в «**TRUE-IS**», ни в «**FALSE-IS**», включены в источник, а кодеры не генерируют кодовых последовательностей для таких значений.

24.6.9 Этот преобразователь определен так, чтобы быть обратимым, если, и только если, установлены оба «**TRUE-IS**» и «**FALSE-IS**», причем каждый из них указывает одиночное целочисленное значение.

24.7 Преобразователь int-to-chars

24.7.1 Преобразователь int-to-chars использует следующие признаки кодирования:

```
&int-to-chars-size      ResultSize DEFAULT variable,
&int-to-chars-plus      BOOLEAN DEFAULT FALSE,
&int-to-chars-pad       ENUMERATED
                        {spaces, zeros} DEFAULT zeros
```

24.7.2 Синтаксисом для преобразователя int-to-chars будет следующий:

```
[INT-TO-CHARS
    [SIZE &int-to-chars-size]
    [PLUS-SIGN &int-to-chars-plus]
    [PADDING &int-to-chars-pad]]
```

24.7.3 Определением типа, используемого в преобразователе int-to-chars, является:

```
ResultSize ::= INTEGER {variable(-1), fixed-to-max(0)} (-1..MAX) -- (см. 21.15)
```

24.7.4 Источником для этого преобразователя является целое число или смесь целых чисел, а результатом является цепочка знаков или смесь цепочек знаков.

24.7.5 Если источник является целым числом, то результат является цепочкой знаков. Если источник является смесью целых чисел, то результат является смесью цепочек знаков, в которой каждый элемент источника был преобразован согласно 24.7.6—24.7.13.

24.7.6 Все компоненты «**SIZE**», «**PLUS-SIGN**» и «**PADDING**» имеют безусловные значения (по умолчанию) и могут быть опущены.

24.7.7 «**SIZE**» указывает:

a) фиксированный размер знаков для результирующего размера (положительное значение «**SIZE**») либо

b) что должна создаваться цепочка знаков с переменной длиной (значение «**variable**» для «**SIZE**»);

c) фиксированный размер, а именно настолько большой, чтобы вместить преобразователь всех абстрактных значений в классе источника (значение «**fixed-to-max**» для «**SIZE**»).

24.7.8 «**SIZE**» не устанавливается в «**fixed-to-max**», если это не первый преобразователь в упорядоченном наборе, а класс источника не имеет ни нижней, ни верхней границ. Это является синонимом спецификации положительного значения, равного наименьшему значению, которое необходимо для вмещения преобразователя каждого абстрактного значения внутри границ.

24.7.9 Значение целого числа сначала превращается в десятичное представление без начальных нулей и с префиксом «-» (ДЕФИС-МИНУС), если оно отрицательное. Положительные значения имеют «+» (ЗНАК ПЛЮСА) в виде префикса перед цифрами, если и только если «**PLUS-SIGN**» установлен в **TRUE**.

24.7.10 Цифра старшего порядка должна быть в начале цепочки знаков.

24.7.11 Если «**SIZE**» равен «**variable**», то это будет результирующей цепочкой знаков. В этом случае указание значения для «**PADDING**» не будет ошибкой, но это значение игнорируется.

24.7.12 Если «**SIZE**» является положительным значением или «**fixed-to-max**», а результирующая цепочка (в экземпляре применения этого преобразователя во время кодирования) слишком велика для фиксированного размера, то это будет ошибкой спецификации ECN или применения, а кодеры не должны генерировать кодовых последовательностей для таких абстрактных значений.

24.7.13 Если «**SIZE**» является положительным значением или «**fixed-to-max**», а цепочка меньше, чем фиксированный размер, то она заполняется префиксами, а именно знаками « » (ПРОБЕЛ) или «0» (ЦИФРА НУЛЬ) согласно значению «**PADDING**», чтобы образовать указанный размер.

24.7.14 Этот преобразователь определен так, чтобы быть обратимым для всех абстрактных значений.

24.8 Преобразователь int-to-bits

Примечание — Пример этого преобразователя приведен в D.1.5.5.

24.8.1 Преобразователь int-to-bits использует следующие признаки кодирования:

```
&int-to-bits-encoded-as     ENUMERATED
                            {positive-int, twos-complement}
                            DEFAULT twos-complement,
&int-to-bits-unit           Unit (1..MAX) DEFAULT bit,
&int-to-bits-size           ResultSize DEFAULT variable
```

24.8.2 Синтаксисом для преобразователя int-to-bits будет следующий:

```
[INT-TO-BITS
        [AS &int-to-bits-encoded-as]
        [SIZE &int-to-bits-size]
        [MULTIPLE OF &int-to-bits-unit]]
```

24.8.3 Определением типов, используемых в преобразователе int-to-bits, является:

```
Unit ::= INTEGER
    {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
    dword32(32)} (0..256) -- (см. 21.1)
ResultSize ::= INTEGER {variable(-1), fixed-to-max(0)} (-1..MAX) -- (см. 21.15)
```

24.8.4 Источником для этого преобразователя является целое число или смесь целых чисел, а результатом является цепочка битов или смесь цепочек битов. Отсутствуют границы, связанные с результатом. В последующих разделах используется термин «результирующая цепочка битов».

24.8.5 Если источник является целым числом, то результат является результирующей цепочкой битов. Если источник является смесью целых чисел, то результат является смесью цепочек битов, в которой каждый элемент источника был преобразован в результирующую цепочку битов, как указано в 24.5.5.

24.8.6 «**AS**» и «**MULTIPLE OF**» имеют безусловные значения (по умолчанию) и не требуют установки.

24.8.7 «**SIZE**» имеет безусловное значение (по умолчанию) и не требует установки, если источник не является смесью. Он должен устанавливаться в положительное значение, если источник является смесью.

24.8.8 «**SIZE**» не устанавливается в «**fixed-to-max**», если это не первый преобразователь в упорядоченном наборе в синтаксисе, определенном в 19.4, а класс источника не имеет ни нижней, ни верх-

ней границ. Это является синонимом спецификации положительного значения, равного наименьшему значению, которое необходимо для вмещения преобразователя каждого абстрактного значения внутри границ.

Примечание — «**SIZE**» не может устанавливаться в «**fixed-to-max**», если источник является смесью преобразователей.

24.8.9 «**AS**» выбирает кодирование целого числа в виде кодирования с дополнением до 2 или кодирования положительного целого числа. Описание этих кодирований дано в ИСО/МЭК 8825-1, пункты 8.3.2 и 8.3.3.

24.8.10 Бит старшего порядка должен быть в начале цепочки битов.

24.8.11 Целое число сначала кодируется в минимальное число битов, необходимое для создания начальной цепочки битов. Это значит, что кодирование положительного целого числа не должно иметь нуль в качестве начального бита (если в кодировании нет одиночного бита «нуль»), а кодирование с дополнением до 2 не должно иметь двух следующих друг за другом начальных битов «нуль» или двух следующих друг за другом начальных битов «единица».

24.8.12 Если «**AS**» установлен в «**positive-int**», а преобразуемое значение отрицательно, то спецификация ECN или применение будут ошибочными, а кодеры не будут кодировать такие значения.

24.8.13 Если «**SIZE**» равно «**variable**», то начальная цепочка битов становится результирующей цепочкой битов. В этом случае указание значения для **«MULTIPLE OF»** не будет ошибкой, но это значение игнорируется.

Примечание — Этот раздел не может применяться, если источник является смесью.

24.8.14 Если «**SIZE**» является положительным значением, то размер результирующей цепочки битов должен равняться произведению «**MULTIPLE OF**» на «**SIZE**».

24.8.15 Если «**SIZE**» равен «**fixed-to-max**», то размер результирующей цепочки битов должен быть наименьшим числом «**MULTIPLE OF**», которое достаточно велико для приема кодирования любого абстрактного значения класса, к которому преобразователь применен.

Примечание — Этот раздел не может применяться, если источник является смесью.

24.8.16 Если начальная цепочка битов (в экземпляре применения этого преобразователя во время кодирования) слишком велика для фильтрованного размера, то это будет ошибкой спецификации ECN или применения, а кодеры не должны кодировать такие значения.

24.8.17 Если начальная цепочка битов меньше, чем указанный размер, то при кодировании положительного целого числа оно должно иметь нуль битов в виде префикса при создании результирующей цепочки битов. Если применено кодирование с дополнением до 2, то оно должно иметь префикс в виде битов, равных по значению исходному начальному биту, при создании результирующей цепочки битов.

24.8.18 Этот преобразователь определен так, чтобы быть обратимым для всех абстрактных значений. Этот преобразователь создает саморазграничивающую цепочку битов, если, и только если, «**SIZE**» не равен «**variable**», а источник не является смесью. Результат в виде смеси никогда не будет саморазграничивающим.

24.9 Преобразователь bits-to-int

24.9.1 Преобразователь bits-to-int использует следующий признак кодирования:

```
&bits-to-int-decoded-assuming      ENUMERATED
                                   {positive-int, twos-complement}
                                   DEFAULT twos-complement
```

24.9.2 Синтаксисом для преобразователя bits-to-int будет следующий:

```
[BITS-TO-INT
        [AS &bits-to-int-decoded-assuming]]
```

24.9.3 Источником для этого преобразователя является цепочка битов или смесь цепочек битов, а результатом является целое число или смесь целых чисел. Отсутствуют границы, связанные с результатом.

24.9.4 Если источник является цепочкой битов, то результат является целым числом. Если источник является смесью цепочек битов, то результат является смесью целых чисел, в которой каждое целое число является результатом спецификации по 24.9.5.

24.9.5 Значение целого числа образуется путем распознавания битов как кодирование с дополнением до 2 или как кодирование положительного целого числа, которые определены в ИСО/МЭК 8825-1, пункты 8.3.2 и 8.3.3. Значение «**AS**» (если оно не установлено, то берется его значение по умолчанию) определяет кодирование, которое предполагается.

24.9.6 Этот преобразователь не используется, когда необходимы обратимые преобразователи.

24.10 Преобразователь char-to-bits

24.10.1 Преобразователь char-to-bits использует следующие признаки кодирования:

```
&char-to-bits-encoded-as    ENUMERATED
                            {iso10646, compact, mapped}
                            DEFAULT compact,
&Char-to-bits-chars         UniversalString (SIZE(1))
                            ORDERED OPTIONAL,
&Char-to-bits-values        BIT STRING ORDERED OPTIONAL,
&char-to-bits-unit          Unit (1..MAX) DEFAULT bit,
&char-to-bits-size          ResultSize DEFAULT variable
```

24.10.2 Синтаксисом для преобразователя char-to-bits будет следующий:

```
[CHAR-TO-BITS
        [AS &char-to-bits-encoded-as]
        [CHAR-LIST &Char-to-bits-chars]
  [BITS-LIST &Char-to-bits-values]
  [SIZE &char-to-bits-size]
  [MULTIPLE OF &char-to-bits-unit]]
```

24.10.3 Определением типов, используемых в преобразователе char-to-bits, являются:

```
Unit ::= INTEGER
      {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
      dword32(32)} (0..256) -- (см. 21.1)
ResultSize ::= INTEGER {variable(-1), fixed-to-max(0)} (-1..MAX) -- (см. 21.15)
```

24.10.4 Источником для этого преобразователя является одиночный знак из:

a) спецификации кодирования для категории «цепочка знаков» (см. 23.4.2.1) или

b) смеси, состоящей из одиночных знаков, а результатом является цепочка битов в случае a) и смесь цепочек битов в данном случае b).

24.10.5 Источником для этого преобразователя является один символ или символьный композит. Если источником является один символ, то результатом — строка битов. Если источником является символьный композит, то результатом — композит битовых строк.

24.10.6 Когда источник является смесью, результирующая смесь определяется путем применения следующей спецификации ко всем элементам смеси источника для формирования результирующей смеси. Спецификация ECN будет ошибочной, если этот преобразователь применяется к смеси с «**AS**», установленным в «**mapped**», а размеры цепочек битов в «**BITS-LIST**» не все одинаковы.

24.10.7 Когда в последующем тексте упоминается возможное «ограничение реального разрешенного алфавита», такое ограничение существует, если, и только если, преобразователь является первым в упорядоченном списке, использованном в 23.4, а класс, к которому применен объект кодирования, имеет ограничение реального разрешенного алфавита.

Примечание — Это может быть лишь в случае, когда класс, к которому применен преобразователь, является частью неявно или явно генерируемой структуры. Этот раздел не может применяться к смеси, элементы которой не имеют ограничений реального разрешенного алфавита.

24.10.8 Все компоненты «**AS**», «**SIZE**» и «**MULTIPLE OF**» имеют безусловные значения (по умолчанию) и не требуют установки. «**CHAR-LIST**» и «**BITS-LIST**» используются только в случаях, когда «**AS**» установлен в «**mapped**», причем их наличие в этом случае обязательно, и тогда они должны содержать как минимум один элемент из упорядоченного списка.

24.10.9 ECN поддерживает только знаки из набора знаков ИСО/МЭК 10646. Когда используются типы АСН.1, такие как «GeneralString», теоретически могут появляться знаки, не входящие в этот набор знаков. Такие знаки не поддерживаются этим преобразователем.

24.10.10 Если «**AS**» равен «**mapped**», то преобразователь определяется значениями «**CHAR-LIST**» и «**BITS-LIST**», которые оба должны быть указаны, а значения «**MULTIPLE OF**» и «**SIZE**» игнорируются. Этот преобразователь описывается в 24.10.10.1—24.10.10.5.

24.10.10.1 «**CHAR-LIST**» и «**BITS-LIST**» являются упорядоченными списками значений одиночных знаков и цепочек битов соответственно (эти параметры игнорируются, если «**AS**» не установлен в «**mapped**»).

24.10.10.2 В этих списках должно быть одинаковое число значений, а все значения знаков в «**CHAR-LIST**» должны быть разными.

24.10.10.3 Преобразованием знака из «**CHAR-LIST**» является цепочка битов, указанная в соответствующей позиции в «**BITS-LIST**».

24.10.10.4 Если в экземпляре применения этого преобразователя должен быть преобразован знак, отсутствующий в «**CHAR-LIST**», то возникает ошибка спецификации ECN или применения.

Примечание — Обычно для инструмента будет возможно обнаружить эту ошибку только во время кодирования, так как ограничения на возможные абстрактные значения могут формально отсутствовать в спецификации АСН.1.

24.10.10.5 В этом случае («**AS**» установлен в «**mapped**») преобразователь определяется так, чтобы он был обратимым (для всех абстрактных значений), если, и только если, все значения цепочек битов в «**BITS-LIST**» различны, в других случаях он не используется, когда требуется обратимый преобразователь. Результат является саморазграничивающим, если значения цепочек битов в «**BITS-LIST**» являются саморазграничивающими (см. 3.2.41). Результат в виде смеси никогда не будет саморазграничивающим.

24.10.11 Если «**AS**» равен «**iso10646**», то используется преобразователь, который описывается в 24.10.11.1—24.10.11.5.

24.10.11.1 Знак сначала превращается в целое число с численным значением, указанным в ИСО/МЭК 10646.

Примечание — ИСО/МЭК 10646 содержит так называемые «управляющие знаки ASCII», имеющие позиции в строке 1.

24.10.11.2 Если знак взят из цепочки знаков, которая имеет связанное ограничение реального разрешенного алфавита (см. 24.10.7), то целое число имеет ограничения реального размера, вполне достаточные для вмещения цифровых значений всех знаков из реального разрешенного алфавита.

24.10.11.3 Если нет ограничения реального разрешенного алфавита, то целое число имеет связанное ограничение реального размера 0—32767.

24.10.11.4 Это значение целого числа затем превращается в биты с помощью преобразователя:

INT-TO-BITS -- (см. **24.8**)
AS positive-int
SIZE <size>
MULTIPLE OF <multiple-of>,

где «<size>» является значением «**SIZE**», а «<multiple-of>» является значением «**MULTIPLE OF**» для преобразователя char-to-bits («**SIZE**» и «**MULTIPLE OF**» принимают значения по умолчанию, если они не установлены).

24.10.11.5 В этом случае («**AS**» установлен в «iso10646») преобразователь определяется так, чтобы он был обратимым для всех абстрактных значений. Он вырабатывает саморазграничивающую цепочку битов, если, и только если, «**SIZE**» не равен «**variable**». Результат в виде смеси никогда не будет саморазграничивающим.

24.10.12 Если «**AS**» равен «**compact**», то возникает ошибка спецификации ECN, когда нет ограничения реального разрешенного алфавита, а для других случаев преобразователь описывается в 24.10.12.1—24.10.12.4.

24.10.12.1 Все знаки в реальном разрешенном алфавите располагаются в каноническом порядке согласно их значениям из ИСО/МЭК 10646: первым будет наименьшее значение. Затем первому в списке присваивается целочисленное значение нуль, следующему — единица и т. д.

24.10.12.2 Если реальный разрешенный алфавит содержит «n» знаков, то целое число имеет ограничение реального размера от 0 до «n» – 1.

24.10.12.3 Это целое число затем превращается в биты с помощью преобразователя:

INT-TO-BITS -- (см. **24.8**)
AS positive-int
SIZE <size>
MULTIPLE OF <multiple-of>,

где «<size>» является значением «**SIZE**», а «<multiple-of>» является значением «**MULTIPLE OF**» для преобразователя char-to-bits («**SIZE**» и «**MULTIPLE OF**» принимают значения по умолчанию, если они не установлены).

Примечание — Кодирование PER типа «цепочка знаков» использует эквивалент «**compact**», если только применение этого алгоритма сокращает число битов, необходимых для кодирования знаков (с помощью «**fixed-to-max**»). Такая степень управления невозможна в настоящем стандарте.

24.10.12.4 В этом случае («**AS**» установлен в «**compact**») преобразователь определяется так, чтобы он был обратимым для всех абстрактных значений. Он вырабатывает саморазграничивающую цепочку битов, если, и только если, «**SIZE**» не равен «**variable**». Результат в виде смеси никогда не будет саморазграничивающим.

24.11 Преобразователь bits-to-char

24.11.1 Преобразователь bits-to-char использует следующие признаки кодирования:

&bits-to-char-decoded-assuming	**ENUMERATED** **{iso10646, mapped}** **DEFAULT iso10646,**
&Bits-to-char-values	**BIT STRING ORDERED OPTIONAL,**
&Bits-to-char-chars	**UniversalString (SIZE(1))** **ORDERED OPTIONAL**

24.11.2 Синтаксисом для преобразователя bits-to-char будет следующий:

[BITS-TO-CHAR
[AS &bits-to-char-decoded-assuming]
[BITS-LIST &Bits-to-char-values]
[CHAR-LIST &Bits-to-char-chars]]

24.11.3 Источником для этого преобразователя является цепочка битов или смесь цепочек битов. Если источником является цепочка битов, то результатом является одиночный знак. Если источником является смесь цепочек битов, то результатом является смесь одиночных знаков.

24.11.4 Если источник является смесью цепочек битов, то результирующая смесь одиночных знаков является упорядоченным списком одиночных знаков, образованных преобразованием каждого элемента смеси цепочек битов.

24.11.5 Если «**AS**» равен «**iso10646**», то цепочка битов рассматривается как кодирование положительного целого числа, которое содержит численное значение знака согласно ИСО/МЭК 10646. Спецификация ECN будет ошибочной, когда целочисленное значение превышает 32767.

24.11.6 Если «**AS**» равен «**mapped**», то преобразователь определяется значениями «**CHAR-LIST**» и «**BITS-LIST**». Этот преобразователь описывается в 24.11.6.1—24.11.6.5.

24.11.6.1 «**CHAR-LIST**» и «**BITS-LIST**» являются упорядоченными списками значений одиночных знаков и цепочек битов соответственно (эти параметры игнорируются, если «**AS**» не установлен в «**mapped**»).

24.11.6.2 В этих списках должно быть одинаковое число значений, а все значения знаков и все значения цепочек битов в списке должны быть разными.

24.11.6.3 Преобразованием цепочки битов из «**BITS-LIST**» является знак, указанный в соответствующей позиции в «**CHAR-LIST**».

24.11.6.4 Если в экземпляре применения этого преобразователя должна быть преобразована цепочка битов, отсутствующая в «**BITS-LIST**», то возникает ошибка спецификации ECN или применения.

Примечание — Обычно для инструмента будет возможно обнаружить эту ошибку только во время кодирования, так как ограничения на возможные абстрактные значения могут формально отсутствовать в спецификации АСН.1.

24.11.6.5 Этот преобразователь определен так, чтобы быть обратимым для всех абстрактных значений.

24.12 Преобразователь bit-to-bits

24.12.1 Преобразователь bit-to-bits использует следующие признаки кодирования:

```
&bit-to-bits-one Non-Null-Pattern     DEFAULT bits:'1'B,
&bit-to-bits-zero Non-Null-Pattern    DEFAULT bits:'0'B
```

24.12.2 Синтаксисом для преобразователя bit-to-bits будет следующий:

```
[BIT-TO-BITS
        [ZERO-PATTERN &bit-to-bits-zero]
        [ONE-PATTERN &bit-to-bits-one]]
```

24.12.3 Определением типа, используемого в преобразователе bit-to-bits, является:

```
Non-Null-Pattern ::= Pattern
        (ALL EXCEPT (bits:"B | octets:"H | char8:"" | char16:"" |
                char32:"")) -- (см. 21.10.2)
```

24.12.4 Источником для этого преобразователя является одиночный бит из:

a) спецификации кодирования для категории «цепочка битов» (см. 23.2) или

b) смеси цепочек битов с единичным блоком 1 бит. Результатом является цепочка битов для перечисления a) и смесь цепочек битов для данного перечисления.

24.12.5 Смесь цепочек битов перечисления b) должна быть упорядоченной последовательностью цепочек битов, созданной с помощью следующих преобразований, примененных к каждому элементу смеси цепочек битов источника. Спецификация ECN будет ошибочной, когда «**ZERO-PATTERN**» и «**ONE-PATTERN**» имеют разные размеры.

24.12.6 Только один из «**ZERO-PATTERN**» и «**ONE-PATTERN**» будет равен «**different:any**».

Примечание — Значение «**different:any**» здесь означает комбинацию, которая не совпадает с другой комбинацией, но имеет ту же длину.

24.12.7 Альтернатива «**any-of-length**» не используется ни для «**ZERO-PATTERN**», ни для «**ONE-PATTERN**».

24.12.8 Если бит установлен в нуль, то результатом будет «**ZERO-PATTERN**». Если бит установлен в единицу, то результатом будет «**ONE-PATTERN**».

24.12.9 Спецификация ECN будет ошибочной, когда «**ZERO-PATTERN**» и «**ONE-PATTERN**» одинаковы или когда одна из них является начальной субцепочкой другой.

24.12.10 Этот преобразователь определен так, чтобы он был обратимым для всех абстрактных значений, а результат был саморазграничивающим, если этот преобразователь не применяется к смеси. Результат в виде смеси никогда не будет саморазграничивающим.

24.13 Преобразователь bits-to-bits

24.13.1 Преобразователь bits-to-bits использует следующие признаки кодирования:

```
&Source-values      BIT STRING ORDERED,
&Result-values      BIT STRING ORDERED
```

24.13.2 Синтаксисом для преобразователя bits-to-bits будет:

[BITS-TO-BITS
SOURCE-LIST &Source-values
RESULT-LIST &Result-values]

24.13.3 Источником для этого преобразователя является либо цепочка битов, либо смесь цепочек битов. Если источником является цепочка битов, то результатом является цепочка битов. Если источником является смесь цепочек битов, то результатом является смесь цепочек битов.

24.13.4 Если источник является смесью цепочек битов, то результирующая смесь цепочек битов является упорядоченным списком цепочек битов, полученным путем применения следующей спецификации к каждой цепочке битов в источнике.

24.13.5 Оба компонента «**SIZE**» и «**MULTIPLE OF**» имеют безусловные значения (по умолчанию) и не требуют установки. Требуются «**SOURCE-LIST**» и «**RESULT-LIST**», и должен содержаться по меньшей мере один элемент в упорядоченном списке.

24.13.6 Преобразователь указывается значениями «**SOURCE-LIST**» и «**RESULT-LIST**».

24.13.7 В этих списках должно быть одинаковое число значений цепочек битов, а все значения цепочек битов в «**SOURCE-LIST**» должны быть разными.

24.13.8 Преобразованием цепочки битов из «**SOURCE-LIST**» является цепочка битов, указанная в соответствующей позиции в «**RESULT-LIST**».

24.13.9 Если этот преобразователь применен к смеси, то все цепочки битов в «**RESULT-LIST**» должны иметь один и тот же размер.

24.13.10 Если в экземпляре применения этого преобразователя цепочка битов источника отсутствует в «**SOURCELIST**», то возникает ошибка спецификации ECN или применения.

Примечание — Как правило, инструмент может обнаружить эту ошибку только во время кодирования, так как ограничения на возможные абстрактные значения могут формально отсутствовать в спецификации ACH.1.

24.13.11 Этот преобразователь определен так, чтобы быть обратимым (для всех абстрактных значений), если, и только если, все значения цепочек битов в «**RESULT-LIST**» различны, в других случаях он не используется, когда требуется обратимый преобразователь. Результат является саморазграничивающим, если значения цепочек битов в «**RESULTLIST**» являются различными и саморазграничивающими (см. 3.2.42), а преобразователь применен к какой-либо цепочке битов. Результат в виде смеси никогда не будет саморазграничивающим.

24.14 Преобразователь chars-to-composite-char

24.14.1 Преобразователь chars-to-composite-char превращает цепочку знаков в смесь одиночных знаков.

24.14.2 Синтаксисом для преобразователя chars-to-composite-char будет:

[CHARS-TO-COMPOSITE-CHAR]

24.14.3 Источником для этого преобразователя является цепочка знаков, а результатом является смесь одиночных знаков.

24.14.4 Смесь одиночных знаков является упорядоченным списком знаков из цепочки знаков источника.

24.14.5 Этот преобразователь определен так, чтобы быть обратимым для всех абстрактных значений.

24.15 Преобразователь bits-to-composite-bits

24.15.1 Преобразователь bits-to-composite-bits превращает цепочку битов в смесь цепочек битов, где каждый элемент в виде цепочки битов имеет один и тот же размер (известный).

24.15.2 Преобразователь bits-to-composite-bits использует следующие признаки кодирования:

&bits-to-composite-bits-unit Unit (1..MAX) DEFAULT bit

24.15.3 Синтаксисом для преобразователя bits-to-composite-bits будет:

```
[BITS-TO-COMPOSITE-BITS
    [UNIT &bits-to-composite-bits-unit]]
```

24.15.4 Определением типа, используемого в преобразователе bits-to-composite-bits, является:

```
Unit ::= INTEGER
    {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
    dword32(32)} (0..256) -- (см. 21.1)
```

24.15.5 Источником для этого преобразователя является цепочка битов, а результатом является смесь цепочек битов с размером «**UNIT**».

24.15.6 Смесь цепочек битов с размером «**UNIT**» является упорядоченным списком цепочек битов, каждая из которых имеет размер «**UNIT**». Первой цепочкой битов в смеси будут первые биты «**UNIT**» из цепочки битов источника. Второй будут следующие биты «**UNIT**» и т. д. Если цепочка битов источника не кратна битам «**UNIT**», то возникнет ошибка спецификации ECN или применения.

24.15.7 Этот преобразователь определен так, чтобы быть обратимым для всех абстрактных значений.

24.16 Преобразователь octets-to-composite-bits

24.16.1 Преобразователь octets-to-composite-bits превращает цепочку октетов в смесь цепочек битов размером 8 битов.

24.16.2 Синтаксисом для преобразователя octets-to-composite-bits будет:

```
[OCTETS-TO-COMPOSITE-BITS]
```

24.16.3 Источником для этого преобразователя является цепочка октетов, а результатом является смесь цепочек битов размером 8 битов.

24.16.4 Смесь цепочек битов размером 8 является упорядоченным списком цепочек битов, соответствующих октетам в цепочке октетов источника.

24.16.5 Этот преобразователь определен так, чтобы быть обратимым для всех абстрактных значений.

24.17 Преобразователь composite-char-to-chars

24.17.1 Преобразователь composite-char-to-chars превращает смесь одиночных знаков в цепочку знаков.

24.17.2 Синтаксисом для преобразователя composite-char-to-chars будет:

```
[COMPOSITE-CHAR-TO-CHARS]
```

24.17.3 Источником для этого преобразователя является смесь одиночных знаков, а результатом является цепочка знаков.

24.17.4 Эта цепочка знаков формируется из упорядоченного списка знаков, имеющихся в смеси одиночных знаков (источника).

24.17.5 Этот преобразователь определен так, чтобы быть обратимым для всех абстрактных значений.

24.18 Преобразователь composite-bits-to-bits

24.18.1 Преобразователь composite-bits-to-bits превращает смесь цепочек битов с известным размером единицы в цепочку битов.

24.18.2 Синтаксисом для преобразователя composite-bits-to-bits будет:

```
[COMPOSITE-BITS-TO-BITS]
```

24.18.3 Источником для этого преобразователя является смесь цепочек битов, а результатом является цепочка битов.

24.18.4 Эта цепочка битов формируется из упорядоченного списка цепочек битов, имеющихся в смеси цепочек битов (источника).

24.18.5 Этот преобразователь определен так, чтобы быть обратимым для всех абстрактных значений. Цепочка битов результата не является саморазграничивающей.

Примечание — Этот преобразователь является обратимым, так как единичные блоки, используемые при его генерировании, указываются в преобразователе, который вырабатывает смесь цепочек битов, и связаны с этой смесью.

24.19 Преобразователь composite-bits-to-octets

24.19.1 Преобразователь composite-bits-to-octets превращает смесь цепочек битов размером единичного блока 8 в цепочку октетов. Возникает ошибка спецификации ECN, когда это применяется к смеси цепочек битов, которая имеет размер единичного блока, не равный 8.

24.19.2 Синтаксисом для преобразователя composite-bits-to-octets будет:

[COMPOSITE-BITS-TO-OCTETS]

24.19.3 Источником для этого преобразователя является смесь цепочек битов, а результатом является цепочка октетов.

24.19.4 Эта цепочка октетов формируется из упорядоченного списка цепочек битов, имеющихся в смеси цепочек битов (источника).

24.19.5 Этот преобразователь определен так, чтобы быть обратимым для всех абстрактных значений.

25 Полные кодирования и класс #OUTER

Если нет объекта кодирования класса **#OUTER** в комбинированном наборе объектов кодирования, примененном к какому-либо типу в ELM, то кодер и декодер должны предполагать наличие объекта кодирования этого класса, в котором все признаки кодирования имеют свои безусловные значения (по умолчанию).

25.1 Признаки кодирования, синтаксис и цель для класса #OUTER

25.1.1 Синтаксис для определения объектов кодирования класса **#OUTER** определяется следующим образом.

```
#OUTER ::= ENCODING-CLASS {

        -- Точка выравнивания
        &alignment-point            ENUMERATED
                                    {unchanged, reset } DEFAULT reset,

        -- Заполнение
        &post-padding-unit          Unit (1..MAX) DEFAULT octet,
        &post-padding               Padding DEFAULT zero,
        &post-padding-pattern       Non-Null-Pattern (ALL EXCEPT different:any)
                                    DEFAULT bits:'0'B,

        -- Спецификация реверсии битов (см. 22.12)
        &bit-reversal               ReversalSpecification
                                    DEFAULT no-reversal,

            -- Действие в отношении добавленных битов
            &added-bits             ENUMERATED
                                    {hard-error, signal-application,
```

```
                                    silently-ignore, next-value}
                                    DEFAULT hard-error

} WITH SYNTAX {
        [ALIGNMENT &alignment-point]
        [PADDING
                [MULTIPLE OF &post-padding-unit]
                [POST-PADDING &post-padding
                        [PATTERN &post-padding-pattern]]]
        [BIT-REVERSAL &bit-reversal]
        [ADDED BITS DECODING &added-bits]
}
```

25.1.2 Определениями типов, используемых в спецификации **#OUTER**, являются.

```
Unit ::= INTEGER
        {repetitions(0), bit(1), nibble(4), octet(8), word16(16),
        dword32(32)} (0..256) -- (см. 21.1)
Padding ::= ENUMERATED {zero, one, pattern, encoder-option} -- (см. 21.9)
Non-Null-Pattern ::= Pattern
        (ALL EXCEPT (bits:"B | octets:"H | char8:"" | char16:"" |
                char32:"")) -- (см. 21.10.2)
```

25.1.3 Объекты кодирования класса **#OUTER** определяют действия кодера и декодера по отношению к полному кодированию типа, который кодируется путем:

a) применения кодирования в ELM или

b) применения кодирования к вложенному типу.

25.1.4 Могут быть сделаны три независимые спецификации (см. 21.1.5—21.1.7).

25.1.5 Спецификация «**ALIGNMENT**» (Выравнивание) применима только к вложенному типу и определяет, следует ли точку выравнивания сбросить к заголовку контейнера или оставить такой же, какая используется для кодирования контейнера.

25.1.6 Спецификация «**PADDING**» (Заполнение) определяет, что полное кодирование должно иметь заполнение заканчивающими битами, чтобы сделать число битов от точки выравнивания целым, кратным некоторой единице.

25.1.7 Спецификация «**ADDED BITS DECODING**» (Декодирование добавленных битов) применима только к декодерам и определяет действие, которое следует выполнить, если после завершения декодирования согласно спецификациям кодирования в PDU имеются добавленные биты.

Примечание — Это положение действует, в первую очередь, для обеспечения простого механизма расширяемости без использования маркера расширяемости ACH.1. Последующая версия настоящего стандарта, как ожидается, даст улучшенную поддержку расширяемости.

25.1.8 Все спецификации «**ALIGNMENT**», «**PADDING**» и «**ADDED BITS DECODING**» принимают свои безусловные значения (по умолчанию), если они не установлены или если нет объекта кодирования класса **#OUTER** в комбинированном наборе объектов кодирования.

Примечание — Безусловными значениями являются те, которые используются объектом кодирования класса **#OUTER** для базового невыравненного кодирования PER.

25.2 Действия кодера для #OUTER

25.2.1 Если «**ALIGNMENT**» равен «**unchanged**», то точкой выравнивания, используемой при кодировании вложенного типа, будет точка выравнивания, используемая при кодировании контейнера.

25.2.2 Если «**ALIGNMENT**» равен «**reset**», то точкой выравнивания, используемой при кодировании вложенного типа, будет начало кодирования этого типа.

25.2.3 Если «**PADDING**» установлен, то кодер добавляет биты согласно значениям «**PADDING**» и «**PATTERN**», чтобы сделать число битов от точки выравнивания кратным единице «**MULTIPLE OF**». Если необходимо, «**PADDING**» копируется и обрезается.

25.2.4 Кодер обнаружит ошибку спецификации ECN или применения, когда производится кодирование типа с ограничением содержимого только цепочками октетов, а конкретная кодовая последовательность этого типа (после всех указанных «**PADDING**» действий) не является целым, кратным 8 битам.

25.2.5 Если установлена реверсия битов, то применяются кодирующие действия, указанные в 22.12, с использованием значения «**MULTIPLE OF**», определенного (возможно, по умолчанию) для «**PADDING**».

25.2.6 Кодер будет игнорировать «**ADDED BITS DECODING**».

25.3 Действия декодера для #OUTER

25.3.1 Если установлена реверсия битов, то применяются декодирующие действия, указанные в 22.12, с использованием значения «**MULTIPLE OF**», определенного (возможно, по умолчанию) для «**PADDING**».

25.3.2 Если «**ALIGNMENT**» равен «**unchanged**», то точкой выравнивания, используемой при кодировании вложенного типа, будет точка выравнивания, используемая при кодировании контейнера.

25.3.3 Если «**ALIGNMENT**» равен «**reset**», то точкой выравнивания, используемой при кодировании вложенного типа, будет начало кодирования этого типа.

25.3.4 Декодер определяет биты, добавленные с помощью «**PADDING**» (если он есть) и пассивно игнорирует добавленные биты независимо от их значения.

25.3.5 Если PDU (или контейнер вложенного типа) содержит добавленные биты после конца кодовой последовательности, то декодер выполняет следующие действия:

a) если «**ADDED BITS DECODING**» равен «**hard-error**», определяет ошибку кодера;

b) если «**ADDED BITS DECODING**» равен «**signal-application**», игнорирует все добавленные биты и сообщает приложению, что могут появиться критичные расширения для протокола;

c) если «**ADDED BITS DECODING**» равен «**silently-ignore**», игнорирует все добавленные биты;

d) если «**ADDED BITS DECODING**» равен «**next-value**», прекращает декодирование и ожидает от приложения возобновления декодирования нового значения из оставшихся битов.

Приложение А
(обязательное)

Добавление к ИСО/МЭК 8824-1

Настоящее приложение определяет изменения, которые должны применяться при ссылках в настоящем стандарте на продукции и/или разделы ИСО/МЭК 8824-1.

А.1 Разделы об экспортах и импортах

Продукции «AssignedIdentifier», «Symbol» и «Reference» из 13.1, а также ИСО/МЭК 8824-1, пункты 13.13 и 13.16 изменяются следующим образом:

13.1

```
AssignedIdentifier ::= DefinitiveIdentifier |
empty
Symbol ::=
                Reference
              | BuiltinEncodingClassReference
              | ParameterizedReference
Reference ::=
                encodingclassreference
              | ExternalEncodingClassReference
              | encodingobjectreference
              | encodingobjectsetreference,
```

где "DefinitiveIdentifier" описывается как:

```
DefinitiveIdentifier ::=
        "{" DefinitiveObjIdComponentList "}"
    |   empty
```

Примечания

1 Продукция «AssignedIdentifier» изменена потому, что «valuereference» не может быть определена или импортирована в модули ELM или EDM.

2 «BuiltinEncodingClassReference» может использоваться в качестве «Symbol» только в разделе импортов. Использование продукции «ExternalEncodingClassReference» в «Reference» поясняется в 14.11.

13.13 Когда выбрана альтернатива «SymbolsExported» для «Exports», каждый «Symbol» в «SymbolsExported» должен удовлетворять одному и только одному из следующих условий:

a) он определен в модуле, из которого экспортирован или

b) он появляется только один раз в альтернативе «SymbolsImported» для «Imports» в модуле, из которого он экспортирован.

13.16 Когда выбрана альтернатива «SymbolsImported» для «Imports»:

a) каждый «Symbol» в «SymbolsFromModule» должен быть:

1) определен в теле модуля, обозначенном с помощью «GlobalModuleReference» в «SymbolsFromModule», или

2) представлен только один раз в разделе импортов в модуле, обозначенном с помощью «GlobalModuleReference» в «SymbolsFromModule»;

b) во всех «SymbolsFromModule» в «SymbolsFromModuleList» должен присутствовать «GlobalModuleReference» так, чтобы:

i) все «modulereference» в них отличались друг от друга (как для модулей ACH.1, так и для модулей EDM) и от «modulereference», связанной с модулем, делающим ссылку, а также

ii) «AssignedIdentifier», когда он не пустой, обозначал значения идентификатора объекта, которые все отличаются друг от друга и от значения идентификатора объекта (если он есть), связанного с модулем, делающим ссылку.

А.2 Добавление REFERENCE

Примечание — Это изменение вводится исключительно для применения в разделе 23.

Продукция «Type» в ИСО/МЭК 8824-1, пункт 17.1 изменяется следующим образом:

```
Type ::=
          BuiltinType
        | ReferencedType
        | ConstrainedType
        | REFERENCE
```

А.3 Нотация для значений цепочки знаков

Продукция «CharsDefn» в ИСО/МЭК, пункт 41.8 изменяется следующим образом:

```
CharsDefn ::=
            cstring
          | Quadruple
          | Tuple
          | AbsoluteCharReference
AbsoluteCharReference ::=
          ModuleIdentifier
          "."
          Valuereference
```

«AbsoluteCharReference» является полностью определенным именем, которое ссылается на значение цепочки знаков (типа «**IA5String**» или «**BMPString**»), определенное в «**ASN1-CHARACTER-MODULE**» (см. ИСО/МЭК 8824-1, пункт 42.1).

Приложение В
(обязательное)

Добавление к ИСО/МЭК 8824-2

Это приложение определяет изменения, которые должны применяться при ссылках в настоящем стандарте на продукции и/или разделы ИСО/МЭК 8824-2.

В.1 Определения терминов

ИСО/МЭК 8824-2, подраздел 3.4 добавляется следующими терминами с соответствующими определениями:

поле класса кодирования (encoding class field): Поле, которое содержит произвольный класс кодирования;

тип поля класса кодирования (encoding class field type): Тип, указанный в ссылке на некоторое поле типа для класса объекта кодирования;

поле объекта кодирования (encoding object field): Поле, которое содержит объект кодирования некоторого класса кодирования. Такое поле будет фиксированного класса или переменного класса. В первом случае класс объекта кодирования является фиксированным с помощью спецификации поля. В последнем случае класс объекта кодирования содержится в некотором (конкретном) поле класса кодирования в том же объекте кодирования;

поле набора объектов кодирования (encoding object set field): Поле, которое содержит набор объектов кодирования некоторого указанного класса кодирования;

поле упорядоченного списка значений фиксированного типа (fixed-type ordered value list field): Поле, которое содержит упорядоченный (возможно, пустой) список значений некоторого указанного типа;

поле упорядоченного списка объектов кодирования (ordered encoding object list field): Поле, которое содержит упорядоченный непустой список объектов кодирования некоторого указанного класса кодирования.

поле ссылки (reference field): Поле, которое содержит ссылку на поле структуры кодирования (см. также 17.5.15).

В.2 Дополнительные лексические элементы

Примечание — Это изменение вводится исключительно для применения в разделе 23.

Следующие термины добавляются в ИСО/МЭК 8824-2, раздел 7:

В.2.1 Ссылки на поле упорядоченного списка значений

Имя элемента — orderedvaluelistfieldreference

Элемент «orderedvaluelistfieldreference» должен содержать знак «**&**», непосредственно за которым следует последовательность знаков, указанная для «typereference» в ИКО/МЭК 8824-1, подраздел 12.2.

В.2.2 Ссылки на поле упорядоченного списка объектов кодирования

Имя элемента — orderedencodingobjectlistfieldreference

Элемент «orderedencodingobjectlistfieldreference» должен содержать знак «**&**», непосредственно за которым следует последовательность знаков, указанная для «objectsetreference» в ИСО/МЭК 8824-2, подраздел 7.3.

В.2.3 Ссылки на поле класса кодирования

Имя элемента — encodingclassfieldreference

Элемент «encodingclassfieldreference» должен содержать знак «**&**», непосредственно за которым следует последовательность знаков, указанная для «encodingclassreference» в 8.3.

В.3 Добавление «ENCODING-CLASS»

Примечание — Это изменение вводится исключительно для применения в разделе 23.

Заменить зарезервированное слово «CLASS» на «ENCODING-CLASS» в пункте 9.3 ИСО/МЭК 8824-2.

В.4 Добавления «FieldSpec»

Примечание — Это изменение вводится исключительно для применения в разделе 23.

Пункт 9.4 ИСО/МЭК 8824-2 изменяется следующим образом:

```
FieldSpec ::=
          FixedTypeValueFieldSpec
        | FixedTypeValueSetFieldSpec
        | FixedTypeOrderedValueListFieldSpec
        | FixedClassEncodingObjectFieldSpec
```

| VariableClassEncodingObjectFieldSpec
| FixedClassEncodingObjectSetFieldSpec
| FixedClassOrderedEncodingObjectListFieldSpec
| EncodingClassFieldSpec

B.5 Спецификация поля упорядоченного списка значений фиксированного типа

Примечание — Это изменение вводится исключительно для применения в разделе 23.

Элемент «FixedTypeOrderedValueListFieldSpec» указывает, что поле является полем упорядоченного списка значений фиксированного типа [см. B.1 (приложение B)]:

FixedTypeOrderedValueListFieldSpec ::=
orderedvaluelistfieldreference
DefinedType
ORDERED
FixedTypeOrderedValueListOptionalitySpec ?
FixedTypeOrderedValueListOptionalitySpec ::= OPTIONAL | DEFAULT OrderedValueList

Именем поля является «orderedvaluelistfieldreference». «DefinedType» ссылается на тип значений, содержащихся в этом поле. «FixedTypeOrderedValueListOptionality-Spec», если присутствует, указывает, что поле может не указываться в определении объектов кодирования, либо в случае «DEFAULT», что пропуск создает последующий «OrderedValueList» (см. ИКО/МЭК 8824-1, пункт 26.3), все значения которого будут «DefinedType».

B.6 Спецификация поля объекта кодирования фиксированного класса

Примечание — Это изменение вводится исключительно для применения в разделе 23.

«FixedClassEncodingObjectFieldSpec» указывает, что поле является полем объекта кодирования фиксированного класса [см. B.1 (приложение B)]:

FixedClassOrderedEncodingObjectListFieldSpec ::=
orderedencodingobjectlistfieldreference
DefinedOrBuiltinEncodingClass
ORDERED
OrderedEncodingObjectListOptionalitySpec?
OrderedEncodingObjectListOptionalitySpec ::= OPTIONAL | DEFAULT OrderedEncodingOb-jectList

Именем поля является «objectfieldreference». «DefinedOrBuiltinEncodingClass» указывает класс кодирования объекта кодирования, содержащегося в этом поле (он может быть текущим определяемым «EncodingClass»). «EncodingObjectOptionality-Spec», если присутствует, указывает, что поле может не указываться в определении объекта кодирования, либо в случае **DEFAULT**, что пропуск создает последующий «EncodingObject» (см. 17.1.5), который принадлежит «DefinedOrBuiltinEncodingClass».

B.7 Спецификация поля объекта кодирования переменного класса

«VariableClassEncodingObjectFieldSpec» указывает, что поле является полем объекта кодирования переменного класса [см. B.1 (приложение B)]:

VariableClassEncodingObjectFieldSpec ::=
objectfieldreference
encodingclassfieldreference
EncodingObjectOptionalitySpec?

Именем поля является «objectfieldreference». Компонент «encodingclassfieldreference» указывает поле класса кодирования для определяемого класса кодирования. «EncodingObjectOptionalitySpec», если присутствует, указывает, что объект кодирования может быть пропущен в определении объекта кодирования, либо в случае **DEFAULT**, что пропуск создает последующий «EncodingObject». «EncodingObjectOptionalitySpec» будет таким, чтобы:

a) если поле типа, указанное в «encodingclassfieldreference», имеет «EncodingClassOptionalitySpec» для **OPTIONAL**, то «EncodingObjectOptionalitySpec» тоже будет **OPTIONAL** и

b) если «EncodingObjectOptionalitySpec» равен «**DEFAULT** EncodingObject», то поле класса кодирования, указанное в «encodingclassfieldreference», должно иметь «EncodingClassOptionalitySpec» для «**DEFAULT** DefinedOrBuiltinEncodingClass», а «EncodingObject» должен быть объектом кодирования этого класса.

B.8 Спецификация поля набора объектов кодирования фиксированного класса

Примечание — Это изменение вводится исключительно для применения в разделе 23.

«FixedClassEncodingObjectSetFieldSpec» указывает, что поле является полем набора объектов кодирования фиксированного класса [см. B.1 (приложение B)]:

```
FixedClassEncodingObjectSetFieldSpec ::=
                    objectsetfieldreference
                    DefinedOrBuiltinEncodingClass
                    EncodingObjectSetOptionalitySpec?
EncodingObjectSetOptionalitySpec ::= OPTIONAL | DEFAULT EncodingObjectSet
```

Именем этого поля является «objectsetfieldreference». Компонент «DefinedOrBuiltinEncodingClass» указывает класс объектов кодирования, содержащихся в поле. «EncodingObjectSetOptionalitySpec», если присутствует, указывает, что поле может не указываться в определении объекта кодирования, либо в случае **DEFAULT**, что пропуск создает последующий «EncodingObjectSet» (см. раздел 18), все объекты которого будут из «DefinedOrBuiltinEncodingClass».

B.9 Спецификация поля упорядоченного списка объектов кодирования фиксированного класса

Примечание — Это изменение вводится исключительно для применения в разделе 23.

«FixedClassOrderedEncodingObjectListFieldSpec» указывает, что поле является полем упорядоченного списка объектов кодирования фиксированного класса [см. B.1 (приложение B)]:

```
FixedClassOrderedEncodingObjectListFieldSpec ::=
                    orderedencodingobjectlistieldreference
                    DefinedOrBuiltinEncodingClass
                    ORDERED
                    OrderedEncodingObjectListOptionalitySpec?
OrderedEncodingObjectListOptionalitySpec ::= OPTIONAL | DEFAULTOrderedEncodingObjectList
```

Именем этого поля является «orderedencodingobjectlistfieldreference». Компонент «DefinedOrBuiltinEncodingClass» указывает класс объектов кодирования, содержащихся в поле. «OrderedEncodingObjectListOptionalitySpec», если присутствует, указывает, что поле может не указываться в определении объекта кодирования, либо в случае **DEFAULT**, что пропуск создает последующий «OrderedEncodingObjectList» [см. B.11 (приложение B)], все объекты которого будут из «DefinedOrBuiltinEncodingClass».

B.10 Спецификация поля класса кодирования

Примечание — Это изменение вводится исключительно для применения в разделе 23.

«EncodingClassFieldSpec» указывает, что поле является полем класса кодирования [см. B.1 (приложение B)]:

```
EncodingClassFieldSpec ::=
        encodingclassfieldreference
        EncodingClassOptionalitySpec?
EncodingClassOptionalitySpec ::= OPTIONAL | DEFAULT DefinedOrBuiltinEncodingClass
```

Именем этого поля является «encodingclassfieldreference». Если «EncodingClassOptionalitySpec» отсутствует, то все определения объектов кодирования для этого класса должны содержать спецификацию класса кодирования для этого поля. Если присутствует **OPTIONAL**, то поле может быть оставлено неопределенным. Если присутствует **DEFAULT**, то последующий «DefinedOrBuiltinEncodingClass» обеспечит установку по умолчанию для поля, если оно пропущено в определении.

B.11 Нотация упорядоченного списка значений

```
OrderedValueList ::= "{" Value "," + "}"
```

«OrderedValueList» является упорядоченным списком с одним или несколькими значениями руководящего типа. Он используется, когда приложение применяет семантику к порядку расположения значений в списке.

Примечание — Список значений может определяться только инлайновой нотацией (которой руководит поле типа, или поле набора значений фиксированного типа, или поле упорядоченного списка значений фиксированного типа).

B.12 Нотация упорядоченного списка объектов кодирования

OrderedEncodingObjectList ::= "{" EncodingObject "," + "}"

«OrderedEncodingObjectList» является упорядоченным списком с одним или несколькими объектами кодирования руководящего класса. Он используется, когда приложение применяет семантику к порядку расположения объектов кодирования в списке.

Пример — Список объектов кодирования #TRANSFORM применятся в назначенном порядке.

Примечание — Следующие ограничения возникают из нормативного текста и продукций BNF (Backus-Naur Form, форма Бэкуса — Наура): упорядоченный список объектов кодирования может быть определен только инлайновой нотацией (которой управляет поле упорядоченного списка объектов кодирования); объекты кодирования в этом списке могут определяться с помощью либо справочного имени, либо инлайновой нотации; руководителем не может быть **#ENCODINGS**.

B.13 Имена простейших полей

Изменить 9.13 ИСО/МЭК 8824-2 следующим образом:

9.13 Конструкция «PrimitiveFieldName» используется для указания поля по отношению к классу кодирования содержащего спецификацию:

PrimitiveFieldName ::=
valuefieldreference
| valuesetfieldreference
| orderedvaluelistfieldreference

B.14 Дополнительные зарезервированные слова

Изменить 10.6 и 10.7 ИСО/МЭК 8824-2 следующим образом:

10.6 Лексический элемент «word», используемый в «Literal», не может быть одним из следующих:

BEGIN	**MINUS-INFINITY**	**PER-CANONICAL-UNALIGNED**
BER	**NON-ECN-BEGIN**	**PLUS-INFINITY**
CER	**NULL**	**TRUE**
DER	**OPTIONS**	**UNION**
ENCODE	**OUTER**	**USE**
ENCODE-DECODE	**PER-BASIC-ALIGNED**	**USE-SET**
END	**PER-BASIC-UNALIGNED**	
FALSE	**PER-CANONICAL-UNALIGNED**	

Примечание — Этот список охватывает только те зарезервированные слова АСН.1, которые могут появляться в качестве первого элемента в «Value», «EncodingObject» или «EncodingObjectSet», а также зарезервированное слово **END**. Использование остальных зарезервированных слов ECN не будет вызывать неоднозначность и разрешается. Там, где определенный синтаксис используется в среде, в которой «word» является также «encodingobjectsetreference», использование «word» предпочтительнее.

10.7 «Literal» указывает реальное включение того «Literal», который должен быть «word» в такой позиции в определенном синтаксисе.

B.15 Определение объектов кодирования

Ограничение, наложенное в перечислении d) 10.12 ИСО/МЭК 8824-2, отменяется.

Примечание — Это влияет на определенный синтаксис для определяющих объектов кодирования в некоторых классах (см. разделы 23 и 24). Это означает, например, что для такого определенного синтаксиса:

[BOOL-TO-INT [AS &bool-to-int]]

пользователю разрешается писать:

BOOL-TO-INT

при определении объекта кодирования этого класса. В таком случае значение **DEFAULT**, связанное с параметром «**&bool-to-int**» (то есть «**false-zero**»), используется в определении преобразователя «**BOOL-TO-INT**».

B.16 Добавление к «Setting»

Изменить 11.7 ИСО/МЭК 8824-2 следующим образом:

11.7 «Setting» указывает установку некоторого поля внутри объекта кодирования, подлежащего определению:

```
Setting ::=
          Value
        | ValueSet
        | OrderedValueList
        | EncodingObject
        | EncodingObjectSet
        | OrderedEncodingObjectList
        | DefinedOrBuiltinEncodingClass
        | OUTER
```

Если поле является:

a) полем значения, то должна быть выбрана альтернатива «Value»;

b) полем набора значений фиксированного типа, то альтернатива «ValueSet»;

c) полем упорядоченного списка значений фиксированного типа, то альтернатива «OrderedValueList»;

d) полем объекта кодирования, то альтернатива «EncodingObjectSet»;

e) полем набора объектов кодирования, то альтернатива «EncodingObjectSet»;

f) полем упорядоченного списка объектов кодирования, то альтернатива «OrderedEncodingObjectList»;

g) полем класса кодирования, то альтернатива «DefinedOrBuiltinEncodingClass»;

h) полем ссылки, то должна быть выбрана альтернатива «Value» или **OUTER**. Для поля ссылки, указанного с помощью синтаксиса в разделах 20—25, «Value» должно быть фиктивным параметром. **OUTER** может использоваться всякий раз, когда требуется ссылка, указывающая контейнер, который содержит полное кодирование.

Примечание — Эта установка дополнительно ограничивается, как описано в ИСО/МЭК 8824-2, пункты 9.5—9.12, 11.8 и 11.9.

B.17 Тип поля класса кодирования

Тип, на который указывает эта нотация, зависит от категории имени поля. Для разных категорий имени поля в B.17.2—B.17.4 (приложение B) определяется указываемый тип.

B.17.1 Нотацией для типа поля класса кодирования будет «EncodingClassFieldType»:

```
EncodingClassFieldType ::=
          DefinedOrBuiltinEncodingClass
          "."
          FieldName,
```

где «FieldName» является именем, описанным в ИСО/МЭК 8824-2, пункт 9.14 применительно к классу кодирования, указанному с помощью «DefinedOrBuiltinEncodingClass».

B.17.2 Для значения фиксированного типа, поля набора значений фиксированного типа или поля упорядоченного списка значений фиксированного типа эта нотация указывает «Type», который появляется в спецификации этого поля в определении класса объекта кодирования.

B.17.3 Эта нотация не разрешается, если поле является объектом кодирования, или набором объектов кодирования, или полем упорядоченного списка объектов кодирования.

B.17.4 Нотацией для определения значения этого типа должна быть «FixedTypeFieldVal», определенная в ИСО/МЭК 8824-2, пункт 14.6.

Приложение С
(обязательное)

Добавление к ИСО/МЭК 8824-4

Настоящее приложение определяет изменения, которые должны применяться при ссылках в настоящем стандарте на продукции и/или разделы ИСО/МЭК 8824-4.

С.1 Параметризованные присвоения

Изменить 8.1 и 8.3 ИСО/МЭК 8824-4 следующим образом:

8.1 Имеются операторы параметризованного присвоения, соответствующие каждому из операторов присвоения, указанных в настоящем стандарте. Конструкцией «ParameterizedAssignment» является:

```
ParameterizedAssignment ::=
ParameterizedEncodingObjectAssignment
| ParameterizedEncodingClassAssignment
| ParameterizedEncodingObjectSetAssignment
```

```
8.3 ParameterList ::= "{<" Parameter "," + ">}"
```

```
Governor ::=
          EncodingClassFieldType
        | REFERENCE
        | DefinedOrBuiltinEncodingClass
        | #ENCODINGS
        | Type
```

«DummyReference» в «Parameter» может помещаться для:

a) класса кодирования; в этом случае не должно быть «ParamGovernor»;

b) значения ACH.1, набора значений или упорядоченного списка значений фиксированного типа; в этом случае «ParamGovernor» должен присутствовать как «Governor», который является типом, извлеченным из класса кодирования («EncodingClassFieldType»);

c) «identifier»; в этом случае «ParamGovernor» должен присутствовать как «Governor», который является **REFERENCE**;

d) объекта кодирования или упорядоченного списка объектов кодирования; в этом случае «ParamGovernor» должен присутствовать как «Governor», который является классом кодирования («DefinedOrBuiltinEncodingClass»);

e) набора объектов кодирования; в этом случае «ParamGovernor» должен присутствовать как «Governor», который является **#ENCODINGS**.

Примечание — Параметры «DummyGovernor» не разрешаются в ECN.

С.2 Параметризованные присвоения кодирования

Добавить к 8.2 ИСО/МЭК 8824-4 следующие продукции:

```
ParameterizedEncodingClassAssignment ::=
        encodingclassreference
        ParameterList
        "::="
        EncodingClass
ParameterizedEncodingObjectAssignment ::=
        encodingobjectreference
        ParameterList
        DefinedOrBuiltinEncodingClass
        "::="
        EncodingObject
ParameterizedEncodingObjectSetAssignment ::=
        encodingobjectsetreference
```

```
            ParameterList
            #ENCODINGS
            "::="
        EncodingObjectSet
```

Изменить 8.4 ИСО/МЭК 8824-4 следующим образом:

8.4 Областью действия «DummyReference», которая появляется в «ParameterList», является сам «ParameterList» вместе с той частью «ParameterizedAssignment», которая следует за «::=». В случае «ParameterizedEncodingObjectAssignment» эта область действия расширяется до «DefinedOrBuiltinEncodingClass», который предшествует символу «::=». «DummyReference» делает невидимыми любые другие «Reference» с тем же именем в области действия.

Примечание — Специальный случай «ParameterizedEncodingObjectAssignment» предназначен для использования сообща в разделах о переименованиях (см. D.3.3.3). Он позволяет записать присвоение, такое как указано ниже, в котором фиктивный параметр «**#Any-Class**» объекта кодирования «**new-component-encoding**» используется в качестве реального параметра для класса кодирования «**#New-component**»:

```
new-component-encoding {< #Any-class >} #New-component {< #Any-class >} ::=
        { -- определение объекта кодирования -- }
```

С.3 Ссылки на параметризованные определения

Изменить продукцию «ParameterizedReference» в 9.1 ИСО/МЭК 8824-4 следующим образом:

```
    ParameterizedReference ::=
            Reference
    | Reference "{<" ">}"
```

Добавить в 9.2 ИСО/МЭК 8824-4 следующие продукции:

```
    ParameterizedEncodingObject ::=
            SimpleDefinedEncodingObject
            ActualParameterList
    SimpleDefinedEncodingObject ::=
                ExternalEncodingObjectReference
              | encodingobjectreference
ParameterizedEncodingObjectSet ::=
            SimpleDefinedEncodingObjectSet
            ActualParameterList
  SimpleDefinedEncodingObjectSet ::=
            ExternalEncodingObjectSetReference
          | encodingobjectsetreference
            ParameterizedEncodingClass ::=
            SimpleDefinedEncodingClass
            ActualParameterList
  SimpleDefinedEncodingClass ::=
            ExternalEncodingClassReference
          | encodingclassreference
```

С.4 Список реальных параметров

Изменить 9.5 ИСО/МЭК 8824-4 следующим образом:
9.5 Конструкцией «ActualParameterList» является:

```
    ActualParameterList ::=
            "{<" ActualParameter "," + ">}"
    ActualParameter ::=
             Value
            | ValueSet
            | OrderedValueList
            | DefinedOrBuiltinEncodingClass
            | EncodingObject
            | EncodingObjectSet
            | OrderedEncodingObjectList
```

| identifier
| STRUCTURE
| OUTER

Если соответствующий фиктивный параметр является:

a) значением, то должна быть использована альтернатива «Value»;

b) набором значений, то должна быть использована альтернатива «ValueSet»;

c) упорядоченным списком значений фиксированного типа, то должна быть использована альтернатива «OrderedValueList»;

d) классом кодирования, то должна быть использована альтернатива «DefinedOrBuiltinEncodingClass»;

e) объектом кодирования, то должна быть использована альтернатива «EncodingObject»;

f) набором объектов кодирования, то должна быть использована альтернатива «EncodingObjectSet»;

g) упорядоченным набором объектов кодирования, то должна быть использована альтернатива «OrderedEncodingObjectList»;

h) ссылкой, то должна быть использована альтернатива «identifier», **STRUCTURE** или **OUTER**.

STRUCTURE должна быть использована только тогда, когда используется реальный параметр, указанный в 17.5.15. **OUTER** может использоваться всякий раз, когда требуется ссылка на контейнер, указывающая контейнер полного кодирования.

Приложение D
(справочное)

Примеры

Настоящее приложение содержит примеры использования ECN. Примеры разделены на пять групп:

- общие примеры, которые показывают, как выглядят определения ECN (D.1);
- специализированные примеры, которые показывают, как изменяются некоторые части какого-либо стандартизованного кодирования. Каждый пример содержит описание требований к кодированию и описание выбранных решений и возможных альтернативных решений (D.2);
- примеры специально оперируемой структуры, которые показывают использование явно генерируемых структур, когда одно и то же специализированное кодирование используется несколько раз (D.3);
- пример традиционного протокола, показывающий три способа обращения с проблемой традиционного подхода «бит-еще» («more-bit») к окончанию «последовательности-из» (D.4);
- второй пример традиционного протокола, показывающий, как конструировать определения ECN для протокола, в котором кодовые последовательности сообщения определены с помощью табличной нотации (D.5).

D.1 Общие примеры

Примеры, описанные в D.1.1—D.1.14, являются частью полной спецификации ECN, модули которой ACH.1, EDM и ELM даны в общих чертах в D.1.15, D.1.16 и D.1.17, а полностью даются в экземпляре этого приложения, помещенного на web-сайте, указанном в приложении F.

D.1.1 Объект кодирования булева типа

D.1.1.1 Присвоение ACH.1 равно:

```
Married ::= BOOLEAN
```

D.1.1.2 Присвоениями объекта кодирования (см. 23.3.1) являются:

```
booleanEncoding #BOOLEAN ::= {
        ENCODING-SPACE
                SIZE 1
                MULTIPLE OF bit
        TRUE-PATTERN bits:'1'B
        FALSE-PATTERN bits:'0'B}
marriedEncoding-1 #Married ::= booleanEncoding
```

D.1.1.3 Предварительное выравнивание отсутствует, а пространство кодирования равно одному биту, поэтому «**Married**» кодируется в битовом поле длиной 1. Комбинациями для значений **TRUE** и **FALSE** (в случае одиночного бита) являются '1'B и '0'B соответственно.

D.1.1.4 Указанные выше значения являются такими значениями, которые могут устанавливаться по умолчанию (см. 23.3.1), если соответствующие признаки кодирования были пропущены, поэтому то же самое кодирование может быть достигнуто с помощью менее длинной продукции:

```
marriedEncoding-2 #Married ::= {
        ENCODING-SPACE
                SIZE 1}
```

D.1.1.5 Это кодирование для булева типа является, конечно, как раз тем, которое обеспечивает PER, а другая альтернатива должна определять кодирование, использующее объект кодирования PER для булева типа с помощью синтаксиса, приведенного в 17.3.1.

```
marriedEncoding-3 #Married ::= {
        ENCODE WITH PER-BASIC-UNALIGNED}
```

D.1.1.6 Как показывает этот пример, часто бывают случаи, когда ECN обеспечивает несколько путей для определения кодирования. Дело пользователя — выбирать альтернативу для использования, уравновешивая многословие (формулировки явных значений, которые могут быть даны по умолчанию) с удобочитаемостью и ясностью.

D.1.2 Объект кодирования для целочисленного типа

D.1.2.1 Присвоения ACH.1 равны:

```
EvenPositiveInteger ::= INTEGER (1..MAX) (CONSTRAINED BY {-- Должен быть четным --})
EvenNegativeInteger ::= INTEGER (MIN..-1) (CONSTRAINED BY {-- Должен быть четным --})
```

D.1.2.2 Присвоениями объекта кодирования являются:

```
evenPositiveIntegerEncoding #EvenPositiveInteger ::= {
        USE #NonNegativeInt
MAPPING TRANSFORMS {{INT-TO-INT divide:2}}
        WITH PER-BASIC-UNALIGNED}
#NonNegativeInt ::= #INT(0..MAX)
evenNegativeIntegerEncoding #EvenNegativeInteger ::= {
        USE #NonPositiveInt
MAPPING TRANSFORMS {{INT-TO-INT divide:2
                        -- Примечание: –1 / 2 = 0 — см. раздел 24.3.7 -- }}
        WITH PER-BASIC-UNALIGNED}
#NonPositiveInt ::= #INT(MIN..0)
```

D.1.2.3 Четное значение делится на два, поэтому кодируется с использованием стандартизованных правил кодирования PER для положительных и отрицательных целочисленных типов.

D.1.3 Другой объект кодирования для целочисленного типа

D.1.3.1 Определить объект кодирования, кодирующий целое число в двухоктетном поле, которое начинается на границе октета.

D.1.3.2 Присвоение ACH.1 равно:

```
Altitude ::= INTEGER (0..65535)
```

D.1.3.3 Присвоением объекта кодирования (см. 23.6.1 и 23.7.1) является:

```
integerRightAlignedEncoding #Altitude ::= {
        ENCODING {
            ALIGNED TO NEXT octet
            ENCODING-SPACE
                    SIZE 16}}
```

D.1.4 Объект кодирования для целочисленного типа с пропусками

D.1.4.1 Присвоение ACH.1 равно:

```
IntegerWithHole ::= INTEGER (-256..-1 | 32..1056)
```

D.1.4.2 Присвоениями объекта кодирования (см. 19.5.2) являются:

```
integerWithHoleEncoding #IntegerWithHole ::= {
        USE #IntFrom0To1280
        MAPPING ORDERED VALUES
        WITH PER-BASIC-UNALIGNED}
#IntFrom0To1280 ::= #INT (0..1280)
```

D.1.4.3 «**IntegerWithHole**» кодируется в виде положительного целого числа. Значения в диапазоне от –256 до –1 отображаются в значения в диапазоне 0—255, а значения в диапазоне 32—1056 отображаются в 256—1280.

D.1.5 Более сложный объект кодирования для целочисленного типа

D.1.5.1 Присвоения АСН.1 равны:

```
PositiveInteger ::= INTEGER (1..MAX)
NegativeInteger ::= INTEGER (MIN..-1)
```

D.1.5.2 Присвоениями объекта кодирования являются:

```
positiveIntegerEncoding #PositiveInteger ::=
        integerEncoding
negativeIntegerEncoding #NegativeInteger ::=
        integerEncoding
```

D.1.5.3 Значения типов «**PositiveInteger**» и «**NegativeInteger**» кодируются объектом кодирования «**integerEncoding**» в виде положительного целого числа или целого числа с дополнением до двух соответственно. Это определяется ниже и обеспечивает разные кодовые последовательности в зависимости от границ типа, к которому это применяется.

D.1.5.4 Объект кодирования «**integerEncoding**», определенный здесь, является очень мощным, но довольно сложным. Он содержит пять объектов кодирования класса **#CONDITIONAL-INT**; все они определяют кодирование, выровненное по октетам. Когда кодируемые целочисленные значения имеют границы, число битов будет фиксированным; когда значения не имеют границ, тип должен быть последним в PDU, а значение выравнивается вправо в оставшихся октетах PDU.

D.1.5.5 Определением этого объекта кодирования (см. 23.6.1 и 23.7.1) является:

```
integerEncoding #INT ::= {ENCODINGS {
        { IF unbounded-or-no-lower-bound
                ENCODING-SPACE
                        SIZE variable-with-determinant
                        DETERMINED BY container
                USING OUTER
        ENCODING twos-complement} ,
    { IF bounded-with-negatives
        ENCODING-SPACE
                SIZE fixed-to-max
        ENCODING twos-complement} ,
    { IF semi-bounded-with-negatives
        ENCODING-SPACE
                SIZE variable-with-determinant
                DETERMINED BY container
                USING OUTER
        ENCODING twos-complement} ,
    { IF semi-bounded-without-negatives
        ENCODING-SPACE
                SIZE variable-with-determinant
                DETERMINED BY container
                USING OUTER
        ENCODING positive-int} ,
    { IF bounded-without-negatives
        ENCODING-SPACE
                SIZE fixed-to-max
        ENCODING positive-int}}}
```

D.1.6 Положительные целые числа, закодированные в BCD

D.1.6.1 Этот пример показывает, как кодировать положительное целое число в BCD (Binary Coded Decimal, двоично-кодированное десятичное число) при помощи последовательных преобразований: из целого числа в цепочку знаков, а затем из цепочки знаков в цепочку битов.

D.1.6.2 Предисловие АСН.1 равно:

```
PositiveIntegerBCD ::= INTEGER(0..MAX)
```

D.1.6.3 Присвоением объекта кодирования (см. 19.4, 24.1 и 23.4.1) является:

```
positiveIntegerBCDEncoding #PositiveIntegerBCD ::= {
        USE #CHARS
        MAPPING TRANSFORMS{{
                INT-TO-CHARS
                -- Мы превращаем в знаки (например, целое число 42 становится цепочкой знаков "42")
                -- и кодируем знаки с объектом кодирования "numeric-chars-to-bcdEncoding"
                        SIZE variable
                        PLUS-SIGN FALSE}}
        WITH numeric-chars-to-bcdEncoding }
numeric-chars-to-bcdEncoding #CHARS ::= {
        ALIGNED TO NEXT nibble
                TRANSFORMS {{
                        CHAR-TO-BITS
                -- Мы превращаем каждый знак в цепочку битов (например, знак "4" становится
                -- '0100'B, а "2" становится '0010'B)
                                AS mapped
                                        CHAR-LIST { "0","1","2","3",
                                            "4","5","6","7",
                                            "8","9"}
                                        BITS-LIST { '0000'B, '0001'B, '0010'B, '0011'B,
                                                '0100'B, '0101'B, '0110'B, '0111'B,
                                                '1000'B, '1001'B }}}
                REPETITION-ENCODING {
                        REPETITION-SPACE
                -- Мы определяем конкатенацию цепочек битов для знаков и добавляем признак конца
                -- (например, '0100'B + '0010'B становится '0100 0010 1111'B)
                                SIZE variable-with-determinant
                                DETERMINED BY pattern
                PATTERN bits:'1111'B}}
```

D.1.6.4 Положительное число сначала преобразуется в цепочку знаков преобразователем «int-to-chars», используя факультативные возможности «переменная длины» и «нет знака плюс», а также факультативную возможность по умолчанию «нет заполнения», что дает цепочку, содержащую знаки от «**0**» до «**9**». Затем цепочка знаков кодируется так, что каждый знак преобразуется в комбинацию битов, **'0000'B** для «**0**», **'0001'B** для «**1**» и т. д. до **'1001'B** для «**9**». Цепочки битов выравниваются по границе полуоктета и заканчиваются специальной комбинацией '1111'B.

D.1.6.5 Более сложные альтернативы, не показанные здесь, но широко используемые, могут встраивать кодирование BCD в цепочку октетов с внешним булевым значением, которое указывает, имеется ли в конце неиспользуемый полуоктет.

D.1.7 Объект кодирования класса #BITS

D.1.7.1 Этот пример определяет объект кодирования класса **#BITS** (см. 23.2.1) для цепочки битов, которая выровнена по октетам, заполнена нулями и заканчивается 8-битовым полем, содержащим **'00000000'B** (предполагается, что абстрактное значение никогда не содержит восемь последовательных нулей).

D.1.7.2 Присвоение ACH.1 равно:

```
Fax ::= BIT STRING (CONSTRAINED BY {-- не должны содержаться восемь последовательных битов нуль --})
```

D.1.7.3 Присвоением объекта кодирования (см. 23.2.1, 23.13.1 и 23.14.1) является:

```
faxEncoding #Fax ::= {
        ALIGNED TO NEXT octet
        REPETITION-ENCODING {
            REPETITION-SPACE
                        SIZE variable-with-determinant
                        DETERMINED BY pattern
                PATTERN bits:'00000000'B}}
```

D.1.7.4 Этот объект кодирования (класса **#BITS**) содержит вложенный объект кодирования класса **#CONDITIONALREPETITION**, который указывает механизм и оканчивающую комбинацию.

D.1.7.5 Многие примеры в этом приложении показывают, что здесь имеется широкая опора на безусловные значения (по умолчанию), предусмотренные в разделе 23, и получаются преимущества от способности определять объекты кодирования инлайновым методом, а не отдельно присваивать их справочным именам, которые затем используются в других присвоениях.

D.1.8 Объект кодирования для типа «цепочка октетов»

D.1.8.1 Присвоение ACH.1 равно:

```
BinaryFile ::= OCTET STRING
```

D.1.8.2 Присвоением объекта кодирования (см. 23.9.1) является:

```
binaryFileEncoding #BinaryFile ::= {
        ALIGNED TO NEXT octet
                PADDING one
        REPETITION-ENCODING {
            REPETITION-SPACE
                    SIZE variable-with-determinant
                DETERMINED BY container
                        USING OUTER}}
```

D.1.8.3 Значение выровнено по октетам с использованием заполнения единицами и заканчивается в конце PDU.

D.1.9 Объект кодирования для типа «цепочка знаков»

D.1.9.1 Присвоение ACH.1 равно:

```
Password ::= PrintableString
```

D.1.9.2 Присвоением объекта кодирования (см. 23.4.1 и 23.14.1) является:

```
passwordEncoding #Password ::= {
        ALIGNED TO NEXT octet
            TRANSFORMS {{CHAR-TO-BITS
                                    AS compact
                                    SIZE fixed-to-max
                                    MULTIPLE OF bit }}
        REPETITION-ENCODING {
            REPETITION-SPACE
                SIZE variable-with-determinant
                DETERMINED BY container
                        USING OUTER}}
```

D.1.9.3 Цепочка выровнена по октетам с использованием заполнения нулями и заканчивается в конце PDU; кодированием знаков указано «**compact**», поэтому каждый знак кодируется в 7 битах, используя **'0000000'B** для первого знака ASCII типа **PrintableString**, **'0000001'B** для следующего и т. д.

D.1.10 Отображение значений знаков в значения битов

D.1.10.1 Присвоение ACH.1 равно:

```
CharacterStringToBit ::= IA5String ("FIRST" | "SECOND" | "THIRD")
```

D.1.10.2 Присвоением объекта кодирования (см. 19.2) является:

```
characterStringToBitEncoding #CharacterStringToBit ::= {
        USE #IntFrom0To2
        MAPPING VALUES {
            "FIRST" TO 0,
            "SECOND" TO 1,
                "THIRD" TO 2}
            WITH integerEncoding}
#IntFrom0To2 ::= #INT (0..2),
```

где «integerEncoding» определен в D.1.5.5.

D.1.10.3 Три возможных абстрактных значения отображаются в три целых числа, а затем эти числа кодируются в двухбитовое поле.

D.1.11 Объект кодирования для типа «последовательность»

D.1.11.1 Здесь мы кодируем тип «последовательность», который имеет поле «a», переносящее семантику приложения (то есть видимую для приложения), но мы хотим также использовать его в качестве детерминанта (определителя) присутствия для второго (факультативного) целочисленного числа «b». Далее имеется цепочка октетов, которая выравнивается по октетам и разграничивается концом PDU. Мы нуждаемся в предоставлении специальных кодовых последовательностей для факультативной возможности «b» и используем специальное кодирование, определенное в D.1.8 (путем ссылки на объект кодирования «binaryFileEncoding»), для цепочки октетов «c». Мы хотим еще кодировать все это методом BASIC-PER-UNALIGNED.

D.1.11.2 Присвоение ACH.1 равно:

```
Sequence1    ::= SEQUENCE {
        a    BOOLEAN,
        b    INTEGER OPTIONAL,
        c    BinaryFile
             -- "BinaryFile" определен в D.1.8.1 --}
```

D.1.11.3 Присвоениями ECN (см. 17.5 и 23.11.1) являются:

```
sequence1Encoding #Sequence1 ::= {
    ENCODE STRUCTURE {
    b USE-SET OPTIONAL-ENCODING parameterizedPresenceEncoding {< a >},
    c binaryFileEncoding
    -- "binaryFileEncoding" определен в D.1.8.2 -- }
WITH PER-BASIC-UNALIGNED}
parameterizedPresenceEncoding {< REFERENCE:reference >} #OPTIONAL ::= {
        PRESENCE
            DETERMINED BY field-to-be-used
                USING reference}
```

D.1.11.4 Заметим, что нам не требуется обеспечивать признак кодирования «**DECODERS-TRANSFORMS**» в объекте кодирования «**parameterizedPresenceEncoding**», так как компонент «**a**» относился к булеву типу, а также предполагается, что **TRUE** означает присутствие «**b**». Если, однако, «**a**» имел целочисленное поле или если прикладное значение **TRUE** для «a» фактически означает, что «**b**» отсутствовал, то мы пожелаем включить признак кодирования «**DECODER-TRANSFORMS**», как в D.2.6.

D.1.12 Объект кодирования для выборочного типа

D.1.12.1 Выборочный тип с тремя альтернативами кодируется с использованием номера тега для контекста класса, кодированного в трехбитовом поле, в качестве селектора. Объект кодирования класса **#ALTERNATIVES** указывает, что идентификационный описатель «**Tag**» используется как детерминант; объект кодирования класса **#TAG** определяет позицию идентификационного описателя (три бита). Для каждой альтернативы значение кодируется методом PER basic unaligned.

D.1.12.2 Присвоение ACH.1 равно:

```
Choice ::= CHOICE {
        boolean [1] BOOLEAN,
integer [3] INTEGER,
string [5] IA5String}
```

D.1.12.3 Присвоениями ECN (см. 23.1.1 и 23.15.1) являются:

```
choiceEncoding #Choice ::= {
            ENCODE STRUCTURE {
                boolean [tagEncoding] USE-SET,
                integer [tagEncoding] USE-SET,
                string [tagEncoding] USE-SET
                STRUCTURED WITH {
                    ALTERNATIVE
                        DETERMINED BY handle
                            HANDLE "Tag"}}
 WITH PER-BASIC-UNALIGNED}
```

```
tagEncoding #TAG ::= {
        ENCODING-SPACE
            SIZE 3
            MULTIPLE OF bit
        EXHIBITS HANDLE "Tag" AT {0 | 1 | 2}}
```

D.1.12.4 Возможно, самым ясным путем обеспечения первого присвоения из D.1.12.3 было бы определение нового набора объектов кодирования и применение его следующим образом:

```
MyEncodings #ENCODINGS ::= { tagEncoding } COMPLETED BY PER-BASIC-UNALIGNED
choiceEncoding #Choice ::= {
        ENCODE STRUCTURE {
            STRUCTURED WITH {
                ALTERNATIVE
                    DETERMINED BY handle
                        HANDLE "Tag"}}
WITH MyEncodings}
```

D.1.13 Кодирование цепочки битов, содержащей другое кодирование

D.1.13.1 Значение цепочки битов, закодированной методом PER basic unaligned, содержит кодирование последовательности из целого числа октетов (с заполнением нулями), но не обязательно выровнена по границе октета.

D.1.13.2 Присвоение ACH.1 равно:

```
Sequence2 ::= SEQUENCE {
        a BOOLEAN,
        b BIT STRING (CONTAINING Sequence3) }
Sequence3 ::= SEQUENCE {
        a INTEGER(0..10),
        b BOOLEAN }
```

D.1.13.3 Присвоениями ECN (см. 25.1) являются:

```
sequence2Encoding #Sequence2 ::= {
        ENCODE STRUCTURE {
                b { REPETITION-ENCODING {
            REPETITION-SPACE
                SIZE 8
                        MULTIPLE OF bit}
            CONTENTS-ENCODING {containerEncoding}
            COMPLETED BY PER-BASIC-UNALIGNED}}
        WITH PER-BASIC-UNALIGNED}

sequence3Encoding #Sequence3 ::= {
    ENCODE STRUCTURE {
            STRUCTURED WITH sequence3StructureEncoding
            }
    WITH PER-BASIC-UNALIGNED }

sequence3StructureEncoding #CONCATENATION ::= {
    ENCODING-SPACE
            MULTIPLE OF octet
            VALUE-PADDING
                    JUSTIFIED left:0
                    POST-PADDING zero
                    UNUSED BITS
                            DETERMINED BY not-needed }
```

D.1.14 Набор объектов кодирования

Этот набор объектов кодирования содержит определения кодирования для некоторых типов, указанных в модуле ACH.1 из D.1.15.

```
Example1Encodings #ENCODINGS ::= {
                marriedEncoding-1
            | integerRightAlignedEncoding
            | evenPositiveIntegerEncoding
```

```
                                    | evenNegativeIntegerEncoding
                                    | integerRightAlignedEncoding
                                    | integerWithHoleEncoding
                                    | positiveIntegerEncoding
                                    | negativeIntegerEncoding
                                        | positiveIntegerBCDEncoding
                                        | faxEncoding
                                        | binaryFileEncoding
                                        | passwordEncoding
                                        | characterStringToBitEncoding
                                        | sequence1Encoding
                                        | choiceEncoding-1
                                        | sequence2Encoding
                                        | sequence3Encoding}
```

D.1.15 Определения ACH.1

D.1.15.1 Этот модуль ACH.1 объединяет все определения ACH.1 от D.1.1 до D.1.13. Они будут кодированы согласно объектам кодирования, определенным в EDM из D.1.16, и правилам кодирования PER basic unaligned.

```
Example1-ASN1-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module1(2)}
        DEFINITIONS AUTOMATIC TAGS : :=
        BEGIN
        MyPDU ::= CHOICE {
                    marriedMessage      Married,
                    altitudeMessage     Altitude
                    -- и т. д.
    }
        Married ::= BOOLEAN
        Altitude ::= INTEGER (0..65535)
        -- и т. д.
        END
```

D.1.16 Определения EDM

D.1.16.1 Этот модуль EDM объединяет все определения ECN из D.1.1—D.1.13.

```
Example1-EDM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module1(3)}
        ENCODING-DEFINITIONS ::=
        BEGIN
        EXPORTS Example1Encodings;
        IMPORTS #Married, #Altitude, #EvenPositiveInteger, #EvenNegativeInteger,
                #IntegerRightAligned, #IntegerWithHole, #PositiveInteger,
                #NegativeInteger, #PositiveIntegerBCD, #Fax, #BinaryFile, #Password, #CharacterStringToBit,
                #Sequence1, #Choice, #Sequence2
                FROM Example1-ASN1-Module { joint-iso-itu-t(2) asn1(1) ecn(4)
                examples(5) asn1-module1(2) };
        Example1Encodings #ENCODINGS ::= {
                    marriedEncoding-1 |
                            -- и т. д.
                    sequence2Encoding}
        -- и т. д.
        END
```

D.1.17 Определения ELM

Приведенный ниже ELM кодирует модуль ACH.1, определенный в D.1.15, с помощью объектов EDM, описанных в D.1.16.

```
Example1-ELM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) elm-module1(1)}
        LINK-DEFINITIONS ::=
        BEGIN
        IMPORTS
                Example1Encodings FROM Example-EDM
                        {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module1(3)}
                #MyPDU FROM Example1-ASN1-Module
```

```
                {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module1(2)};
ENCODE #MyPDU WITH Example1Encodings
        COMPLETED BY PER-BASIC-UNALIGNED
END
```

D.2 Специализированные примеры

Примеры в этом разделе показывают, как можно изменять отдельные части какого-либо кодирования заданного типа с целью минимизировать размер закодированных сообщений. Кодовые последовательности PER basic unaligned обычно создаются в виде сжатых по возможности кодовых последовательностей. Однако имеются случаи, когда желательны специализированные кодирования:

- имеются некоторые специальные виды семантики, связанные с такими компонентами сообщения, которые дают возможность удалить некоторые вспомогательные поля, генерируемые PER;
- пользователи желают применять разные кодирования для вспомогательных полей PER, генерируемых по умолчанию, таких как поля детерминанта с переменной шириной.

D.2.1 Кодирование путем распределения значений по альтернативным структурам кодирования

D.2.1.1 Присвоение ACH.1 равно:

```
NormallySmallValues ::= INTEGER (0..1000)
-- Обычно значения находятся в диапазоне 0—63, но иногда используется полный диапазон значений.
```

D.2.1.2 PER кодировал бы тип с использованием 10 битов. Мы хотим минимизировать размер кодирования так, чтобы нормальный случай кодировался с использованием как можно меньшего числа битов.

Примечание — В этом примере мы применяем простой прямой подход. Более сложный подход с использованием кодирования Хаффмана приведены в Е.1.

D.2.1.3 Присвоением объекта кодирования (см. 19.6) является:

```
normallySmallValuesEncoding-1 #NormallySmallValues ::= {
        USE #NormallySmallValuesStruct-1
        MAPPING DISTRIBUTION {
            0..63 TO small,
            REMAINDER TO large }
    WITH PER-BASIC-UNALIGNED}
```

D.2.1.4 Присвоением структуры кодирования является:

```
#NormallySmallValuesStruct ::= #CHOICE {
        small #INT (0..63),
        large #INT (64..1000)}
```

D.2.1.5 Используемые обычно значения кодируются с помощью поля «**small**», а значения, используемые только изредка, кодируются с помощью поля «**large**». Выбор между этими двумя возможностями осуществляется генерируемым PER однобитовым полем селектора. Длина поля «**small**» равна 6 битам, а длина поля «**large**» — 10 битам, поэтому обычный случай кодируется 7 битами, а редкий случай — 11 битами.

D.2.2 Кодирование путем отображения упорядоченных абстрактных значений в альтернативную структуру кодирования

D.2.2.1 В примере D.2.1 использовано явное определение способа отображения диапазонов значений в поля структуры кодирования. Такой же эффект можно получить более просто, используя «отображение упорядоченных абстрактных значений». Однако в качестве иллюстрации мы здесь еще изменим требование: вспомогательные длинные значения могут появляться изредка, а присвоение ACH.1, по предположению, не содержит ограничений для них.

D.2.2.2 Присвоениями объектов кодирования (см. 19.5) являются:

```
normallySmallValuesEncoding-2 #NormallySmallValues ::= {
    USE #NormallySmallValuesStruct2
    MAPPING ORDERED VALUES
    WITH NormallySmallValuesTag-encoding-plus-PER}
normallySmallValuesTag-encoding #TAG ::= {
    ENCODING-SPACE
                SIZE 1}
```

```
NormallySmallValuesTag-encoding-plus-PER #ENCODINGS ::= {normallySmallValuesTag-encoding}
        COMPLETED BY PER-BASIC-UNALIGNED
```

D.2.2.3 Присвоением структуры кодирования является:

```
#NormallySmallValuesStruct2 ::= #CHOICE {
        small [#TAG(0)] #INT (0..63),
        large [#TAG(1)] #INT (0..MAX) }
```

D.2.2.4 Результат очень похож на D.2.1, но теперь значения свыше 64, которые отображаются в поле «**large**», кодируются от нуля и выше. Две альтернативы различаются с помощью однобитового индекса. Другое отличие состоит в том, что поле «**large**» остается неограниченным, так что этот объект кодирования может кодировать произвольно большие целые числа, но за счет длинного поля в случае «**large**». Этот пример может использоваться также, когда нет верхней границы для значений, которые могли бы появляться изредка («**large**» не ограничено в структурах замены). Это опять иллюстрирует гибкость, доступную для проектировщиков ECN при разработке кодирования, удовлетворяющего их конкретным требованиям.

D.2.3 Компрессия прерывистых диапазонов значений

D.2.3.1 Этот пример также использует отображение упорядоченных абстрактных значений. В этом случае отображение используется для компрессии разбросанных значений в некоторую базовую спецификацию АСН.1. Компрессия могла бы достигаться также путем определения в АСН.1 абстрактного значения «x», имеющего прикладную семантику «2x», а затем используя более простое ограничение целочисленного типа АСН.1. В этом примере, однако, предполагается, что разработчик АСН.1 решил не делать этого, а нам необходимо применить компрессию во время отображения из абстрактных значений в кодовые последовательности.

D.2.3.2 Присвоение АСН.1 равно:

```
SparseEvenlyDistributedValueSet ::= INTEGER (2 | 4 | 6 | 8 | 10 | 12 | 14 | 16)
```

D.2.3.3 Кодирование PER basic unaligned применяет нижние и верхние границы только при определении числа битов, необходимых для кодирования целого числа. Это приводит к неиспользуемым комбинациям битов в кодовой последовательности. Такая кодовая последовательность может быть компрессирована, чтобы неиспользуемые комбинации битов были опущены, а каждое значение кодировалось с использованием минимального числа битов.

D.2.3.4 Присвоением объекта кодирования (см. 19.5) является:

```
sparseEvenlyDistributedValueSetEncoding SparseEvenlyDistributedValueSet ::= {
        USE #IntFrom0To7
        MAPPING ORDERED VALUES
        WITH PER-BASIC-UNALIGNED}
    #IntFrom0To7 ::= #INT (0..7)
```

D.2.3.5 Восемь возможных абстрактных значений отображены в диапазон 0—7 и будут кодироваться в трехбитовом поле.

D.2.4 Компрессия прерывных диапазонов значений с помощью преобразователя

D.2.4.1 В примере D.2.3 использовалось отображение упорядоченных абстрактных значений. Такой же эффект может быть достигнут при использовании класса **#TRANSFORM**.

D.2.4.2 Присвоением объекта кодирования (см. 19.4) является:

```
sparseEvenlyDistributedValueSetEncoding-2 #SparseEvenlyDistributedValueSet ::= {
        USE #IntFrom0To7
        MAPPING TRANSFORMS {{INT-TO-INT divide: 2}, {INT-TO-INT decrement:1}}
        WITH PER-BASIC-UNALIGNED}
```

D.2.4.3 Опять восемь возможных абстрактных значений отображаются в диапазон 0—7 и кодируются в трехбитовом поле.

D.2.5 Компрессия неравномерно распределенного набора значений с помощью упорядоченных абстрактных значений

D.2.5.1 Присвоение АСН.1 равно:

```
SparseUnevenlyDistributedValueSet ::= INTEGER (0|3|5|6|11|8)
-- В беспорядке — для иллюстрации того, что порядок следования не важен в таком ограничении.
```

D.2.5.2 Кодирование должно быть таким, чтобы не было пропусков в используемых кодовых комбинациях.

D.2.5.3 Присвоением объекта кодирования является:

```
sparseUnevenlyDistributedValueSetEncoding #SparseUnevenlyDistributedValueSet ::= {
        USE #IntFrom0To5
        MAPPING ORDERED VALUES
        WITH PER-BASIC-UNALIGNED}
        #IntFrom0To5 ::= #INT (0..5)
```

D.2.5.4 Шесть возможных абстрактных значений отображаются в диапазон 0—5 и кодируются в трехбитовое поле. Отображение будет следующим: 0→0, 3→1, 5→2, 6→3, 8→4 и 11→5.

D.2.6 Присутствие факультативного компонента в зависимости от значения другого компонента

D.2.6.1 Присвоение ACH.1 равно:

```
ConditionalPresenceOnValue ::= SEQUENCE {
        a INTEGER (0..4),
        b INTEGER (1..10),
        c BOOLEAN OPTIONAL
                -- Условие: "c" присутствует, если "a" равно 0, а в остальных случаях "c"
                отсутствует --,
    d BOOLEAN OPTIONAL
            -- Условие: "d" отсутствует, если "a" равно 1, а в остальных случаях "d"
            присутствует -- }
-- Учтем подразумеваемые в комментариях ограничения присутствия.
-- Учтем также, что целочисленное поле "a" переносит прикладную семантику
-- и имеет значения, отличающиеся от нуля и единицы. Если "a" имеет значение 0,
-- то "c" и "d" оба присутствуют. Если "a" имеет значение 1, то "c" и "d" оба отсутствуют.
-- Если "a" имеет значение 3 или 4, то "c" отсутствует, а "d" присутствует. Эти условия
-- являются очень жесткими для формального выражения при использовании только ACH.1.
```

D.2.6.2 Компонент «**a**» выполняет роль детерминанта присутствия для двух компонентов «**c**» и «**d**», а кодирование PER будет вырабатывать два вспомогательных бита для факультативных компонентов. Мы нуждаемся в кодировании, в котором эти вспомогательные биты отсутствуют.

D.2.6.3 Присвоениями объекта кодирования являются:

```
conditionalPresenceOnValueEncoding #ConditionalPresenceOnValue ::= {
        ENCODE STRUCTURE {
            c USE-SET OPTIONAL-ENCODING is-c-present{< a >},
            d USE-SET OPTIONAL-ENCODING is-d-present{< a >}}
        WITH PER-BASIC-UNALIGNED}
is-c-present {< REFERENCE : a >} #OPTIONAL ::= {
        PRESENCE
            DETERMINED BY field-to-be-used
            USING a
                DECODER-TRANSFORMS {{INT-TO-BOOL TRUE-IS {0}}}}
is-d-present {< REFERENCE : a >} #OPTIONAL ::= {
        PRESENCE
            DETERMINED BY field-to-be-used
            USING a
                DECODER-TRANSFORMS {{INT-TO-BOOL TRUE-IS {0 | 2 | 3 | 4}}}}
```

D.2.6.4 Здесь мы имеем простую, формальную и ясную спецификацию условий присутствия «**c**» и «**d**», которая может быть понятна для инструментов кодера-декодера. Комментарии ACH.1 не могут обрабатываться инструментами. Обеспечение кодирования функциональных возможностей для «**c**» и «**d**» означает, что кодирование PER для **OPTIONAL** не используется в этом случае и нет вспомогательных битов.

D.2.6.5 Параметризованные объекты кодирования «**is-c-present**» и «**is-d-present**» указывают, как определяется присутствие компонентов во время кодирования. Заметим, что для кодирования не требуется (и не разрешается) преобразование, так как детерминант имеет прикладную семантику (то есть он виден в определении типа ACH.1). Однако хороший кодирующий инструмент будет контролировать установку «**a**» с помощью приложения, обеспечивая, что ее значение согласуется с наличием или отсутствием «**c**» и «**d**», которое определяется кодом приложения.

D.2.7 Присутствие факультативного компонента в зависимости от некоторого внешнего условия

D.2.7.1 Присвоение ACH.1 равно:

```
ConditionalPresenceOnExternalCondition ::= SEQUENCE {
        a BOOLEAN OPTIONAL
          -- Условие: "a" присутствует, если сохраняется внешнее условие "C",
          -- в противном случае "a" отсутствует -- }
        -- Отметим, что ограничение присутствия может добавляться только в комментарии.
```

D.2.7.2 Код приложения для передатчика и приемника может оценивать условие «C» от некоторой информации, не содержащейся в сообщении. Спецификатор ECN нуждается в инструментах для вызова такого кода при определении присутствия «a», а не в использовании какого-либо бита в кодовой последовательности.

D.2.7.3 Присвоением объекта кодирования является:

```
conditionalPresenceOnExternalConditionEncoding #ConditionalPresenceOnExternalCondition ::= {
    ENCODE STRUCTURE {
        a USE-SET OPTIONAL-ENCODING is-a-present}
    WITH PER-BASIC-UNALIGNED}
is-a-present #OPTIONAL ::=
    NON-ECN-BEGIN {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) user-notation(7)}
    extern C;
    extern channel;
    /* a присутствует только, если канал эквивалентен некоторому значению "C" */
    int is_a_present() {
        if(channel == C) return 1;
        else return 0; }
    NON-ECN-END
```

D.2.7.4 Так как условие является внешним по отношению к сообщению, объект кодирования для определения присутствия компонента «a» может быть описан только не-ECN определением некоторого объекта кодирования. Хотя это экономит биты в линии, многие разработчики, однако, могут считать, что лучше включать бит в сообщение, чтобы уменьшить вероятность ошибки и облегчить тестирование и контроль. Этот выбор остается за разработчиком ECN.

D.2.8 Список с переменной длиной

```
EnclosingStructureForList ::= SEQUENCE {
        list VariableLengthList}
VariableLengthList ::= SEQUENCE (SIZE (0..1023) ) OF INTEGER (1..2)
-- Обычно этот список содержит только небольшое число элементов (0—31), но мог бы содержать много.
```

D.2.8.2 PER basic unaligned кодирует длину списка с помощью 10 битов, даже если обычно длина находится в диапазоне 0—31. Мы хотим минимизировать размер кодирования детерминанта длины в обычных случаях, разрешая, тем не менее, значения, которые появляются реже.

D.2.8.3 Присвоением объекта кодирования является:

```
enclosingStructureForListEncoding #EnclosingStructureForList ::= {
        USE #EnclosingStructureForListStruct
        MAPPING FIELDS
        WITH {
            ENCODE STRUCTURE {
                aux-length list-lengthEncoding,
                list {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH {
                                                    REPETITION-ENCODING {
                                                        REPETITION-SPACE
                                                            DETERMINED BY field-to-be-set
                                                        USING aux-length}}}
                                            WITH PER-BASIC-UNALIGNED }}
WITH PER-BASIC-UNALIGNED}}
                -- Первое отображение: использование структуры кодирования с явным детерминантом
```

длины.

```
list-lengthEncoding #AuxVariableListLength ::= {
        USE #AuxVariableListLengthStruct -- См. D.2.8.4.
        MAPPING ORDERED VALUES
        WITH PER-BASIC-UNALIGNED}
        -- Второе отображение: длина списка кодируется в виде выбора между короткой формой "normally" и
        -- длинной формой "sometimes".
```

D.2.8.4 Присвоением структуры кодирования является:

```
#EnclosingStructureForListStruct ::= #CONCATENATION {
        aux-length #AuxVariableListLength,
        list #VariableLengthList}
#AuxVariableListLength ::= #INT (0..1023)
#AuxVariableListLengthStruct ::= #ALTERNATIVES {
        normally #INT (0..31),
        sometimes #INT (32..1023)}
```

D.2.8.5 Детерминант длины для компонента «**list**» будет переменным. Детерминант длины для значений короткого списка кодируется с помощью 1 бита для детерминанта выбора и 5 битов для детерминанта длины. Детерминант длины для значений длинного списка кодируется с помощью 1 бита для детерминанта выбора и 10 битов для детерминанта длины.

D.2.9 Списки с одинаковой длиной

D.2.9.1 Присвоение ACH.1 равно:

```
EqualLengthLists ::= SEQUENCE {
        list1 List1,
        list2 List2}
        (CONSTRAINED BY {
        -- "list1" и "list2" всегда имеют одно и то же число элементов. --
        })
List1 ::= SEQUENCE (SIZE (0..1023)) OF BOOLEAN
List2 ::= SEQUENCE (SIZE (0..1023)) OF INTEGER (1..2)
```

D.2.9.2 Как «**list1**», так и «**list2**» имеют одинаковое число элементов, а разработчик ECN желает использовать единый детерминант длины для обоих списков (PER может кодировать поля длины для обоих компонентов).

D.2.9.3 Присвоениями объекта кодирования являются:

```
equalLengthListsEncoding #EqualLengthLists ::= {
        USE #EqualLengthListsStruct
        MAPPING FIELDS
        WITH {
            ENCODE STRUCTURE {
                list1 list1Encoding{< aux-length >},
                list2 list2Encoding{< aux-length >}}
                  WITH PER-BASIC-UNALIGNED}}
```

Первый объект кодирования определяется с двумя параметризованными объектами кодирования классов **#List1** и **#List2** соответственно, используя поле длины в качестве реального параметра. Эти два объекта кодирования используют общий параметризованный объект кодирования класса **#REPETITION**.

```
list1Encoding {< REFERENCE : length >} #List1 ::= {
        ENCODE STRUCTURE { USE-SET
                STRUCTURED WITH list-with-determinantEncoding {< length >}}
        WITH PER-BASIC-UNALIGNED}
list2Encoding {< REFERENCE : length >} #List2 ::= {
        ENCODE STRUCTURE { USE-SET
                STRUCTURED WITH list-with-determinantEncoding {< length >}}
        WITH PER-BASIC-UNALIGNED}
list-with-determinantEncoding {< REFERENCE : length-determinant >} #REPETITION ::= {
        REPETITION-ENCODING {
```

```
            REPETITION-SPACE
                SIZE variable-with-determinant
                    MULTIPLE OF repetitions
                DETERMINED BY field-to-be-set
                USING length-determinant}}
```

D.2.9.4 Присвоениями структуры кодирования являются:

```
#EqualLengthListsStruct ::= #CONCATENATION {
            aux-length #AuxListLength,
            list1 #List1,
            list2 #List2}
#AuxListLength ::= #INT (0..1023)
```

D.2.10 Неравные вероятности выборочных альтернатив

D.2.10.1 Присвоение ACH.1 равно:

```
EnclosingStructureForChoice ::= SEQUENCE {
        choice UnevenChoiceProbability }
UnevenChoiceProbability ::= CHOICE {
        frequent1 INTEGER (1..2),
        frequent2 BOOLEAN,
        common1 INTEGER (1..2),
        common2 BOOLEAN,
        common3 BOOLEAN,
        rare1 BOOLEAN,
        rare2 INTEGER (1..2),
        rare3 INTEGER (1..2)}
```

D.2.10.2 Эти альтернативы выборочного типа имеют разные вероятности выбора. Имеются альтернативы, которые появляются очень часто («**frequent1**» и «**frequent2**») или являются достаточно обычными («**common1**», «**common2**» и «**common3**»), или появляются только редко («**rare1**», «**rare2**» и «**rare3**»). Кодирование детерминанта альтернативы должно быть таким, чтобы альтернативы, появляющиеся чаще, имели более короткие поля детерминанта, чем те, которые появляются реже.

D.2.10.3 Присвоениями структуры кодирования являются:

```
#EnclosingStructureForChoiceStruct ::= #CONCATENATION {
            aux-selector #AuxSelector,
            choice #UnevenChoiceProbability }
            -- Явный вспомогательный детерминант альтернативы для "выборочного типа".
#AuxSelector ::= #INT (0..7)
```

D.2.10.4 Присвоениями объекта кодирования являются:

```
enclosingStructureForChoiceEncoding #EnclosingStructureForChoice ::= {
        USE #EnclosingStructureForChoiceStruct
        MAPPING FIELDS
        WITH {
                ENCODE STRUCTURE {
                    aux-selector auxSelectorEncoding,
                    choice {
                        ENCODE STRUCTURE {
                            STRUCTURED WITH {
                                ALTERNATIVE
                                    DETERMINED BY field-to-be-set
                                    USING aux-selector}}
                            WITH PER-BASIC-UNALIGNED }}
                        WITH PER-BASIC-UNALIGNED} }
                    -- Первое отображение: вводит явный вспомогательный
                    -- детерминант альтернативы.
```

```
                    -- Этот объект кодирования указывает, что вспомогательный
                    детерминант используется в качестве
                    -- детерминанта альтернативы.
auxSelectorEncoding #AuxSelector ::= {
        USE #BITS
        -- ECN Хаффман
        -- ДИАПАЗОН (0—7)
        -- (0—1) занимают 60 %
        -- (2—4) занимают 30 %
        -- (5—7) занимают 10 %
        -- Определение конца (End Definition)
        -- Отображения, которые вырабатывает "ECN Public
        -- Domain Software for Huffman encodings, version 1"
        -- (см. E.8).
        MAPPING TO BITS {
            0 .. 1 TO '10'B .. '11'B,
            2 .. 4 TO '001'B .. '011'B,
            5 TO '0001'B,
            6 .. 7 TO '00000'B .. '00001'B}
    WITH PER-BASIC-UNALIGNED }
-- Второе отображение: Отображает индексы детерминанта в цепочки битов.
```

D.2.10.5 Выше мы определили количественно «frequent», «common» и «rare» как 60, 30 и 10 % соответственно и применили генератор ECN Хаффмана для области общего использования (см. E.8) при определении оптимальных комбинаций битов, которые следует использовать для каждого диапазона целых чисел.

D.2.10.6 Вышеприведенное является оптимальным в математическом смысле, но разница, которая получается в процентах от общего трафика, зависит от того, какие другие части содержит протокол. Если попытки реализации обеспечения и использования оптимального кодирования ничего не стоят (так как могут быть использованы инструменты), конечные выгоды могут быть не важны.

D.2.11 Сообщение версии 1

D.2.11.1 Присвоение ACH.1:

```
Version1Message ::= SEQUENCE {
        ie-1 BOOLEAN,
        ie-2 INTEGER (0..20)}
```

Мы хотим использовать PER basic unaligned, но намереваемся добавить еще поля в версию 2 и желаем указать, что системы версии 1 должны принимать и игнорировать любой дополнительный материал в PDU.

D.2.11.2 Мы используем две структуры кодирования для кодирования сообщения: первая является неявно генерируемой структурой кодирования, содержащей только поля версии 1, а вторая является структурой, которую мы определяем и которая содержит поля версии 1 плюс поле заполнения с переменной длиной, которое простирается до конца PDU. Система версии 1 использует первую структуру для кодирования, а вторую — для декодирования. Не считая этот подход к расширяемости, все кодовые последовательности являются PER basic unaligned. Структурой кодирования версии 1 является:

```
#Version1DecodingStructure ::= #CONCATENATION {
        ie-1 #BOOL,
                ie-2 #INT (0..20),
                future-additions #PAD}
```

D.2.11.3 Присвоениями объекта кодирования являются:

```
version1MessageEncoding #Version1Message ::= {
        ENCODE-DECODE
            {ENCODE WITH PER-BASIC-UNALIGNED }
        DECODE AS IF decodingSpecification}
decodingSpecification #Version1Message ::= {
        USE #Version1DecodingStructure
        MAPPING FIELDS
                WITH {
                ENCODE STRUCTURE {
```

```
                    future-additions additionsEncoding{< OUTER >} }
                    WITH PER-BASIC-UNALIGNED}}
    additionsEncoding {< REFERENCE:determinant >} #PAD ::= {
        ENCODING-SPACE
            SIZE encoder-option-with-determinant
            DETERMINED BY container
            USING determinant}
```

D.2.12 Набор объектов кодирования

Этот набор объектов кодирования содержит определения кодирования для некоторых типов, указанных в модуле ACH.1 с именем «**Example2-ASN1-Module**» (остаток кодируется с использованием метода PER basic unaligned).

```
Example2Encodings #ENCODINGS ::= {
        normallySmallValuesEncoding-1                       |
        sparseEvenlyDistributedValueSetEncoding             |
        sparseUnevenlyDistributedValueSetEncoding           |
        conditionalPresenceOnValueEncoding                  |
        conditionalPresenceOnExternalConditionEncoding      |
        enclosingStructureForListEncoding                   |
        equalLenghListsEncoding                             |
        enclosingStructureForChoiceEncoding                 |
        version1MessageEncoding }
```

D.2.13 Определения ACH.1

Этот модуль объединяет все определения ACH.1 от D.2.1 до D.2.11, которые будут кодироваться согласно объектам кодирования, определенным в EDM, и перечисляет также другие определения ACH.1, которые будут кодироваться по правилам кодирования PER basic unaligned.

```
Example2-ASN1-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module2(5)}
        DEFINITIONS AUTOMATIC TAGS : :=
        BEGIN
        ExampleMessages ::= CHOICE {
                firstExample NormallySmallValues,
                secondExample SparseEvenlyDistributedValueSet
                -- и т. д.
}
        NormallySmallValues ::= INTEGER (0..1024)
        SparseEvenlyDistributedValueSet ::= INTEGER (2 | 4 | 6 | 8 | 10 | 12 | 14 | 16)
        -- и т. д.
        END
```

D.2.14 Определения EDM

```
Example2-EDM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module2(6)}
        ENCODING-DEFINITIONS ::=
        BEGIN
        EXPORTS Example2Encodings;
        IMPORTS #NormallySmallValues, #SparseEvenlyDistributedValue,
                #SparseUnevenlyDistributedValueSet, #ConditionalPresenceOnValueSet,
                #ConditionalPresenceOnExternalCondition,
                #EnclosingStructureForList,#EqualLengthLists, EnclosingStructureForChoice,
                #Version1Message
                        FROM Example2-ASN1-Module
                        {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module2(5)};
Example2Encodings #ENCODINGS ::= {
                        normallySmallValuesEncoding |
                        -- и т. д.
                        extensibleMessageEncoding}
        -- и т. д.
        END
```

D.2.15 Определения ELM

Нижеследующий ELM связан с модулем АСН.1, определенным в D.2.13, и с EDM, определенным в D.2.14.

```
Example2-ELM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) elm-module2(4)}
        LINK-DEFINITIONS ::=
        BEGIN
        IMPORTS
                Example2Encodings FROM Example2-EDM
                        {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module2(6)}
                #ExampleMessages FROM Example2-ASN1-Module
                        {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module2(5)};
        ENCODE #ExampleMessages WITH Example2Encodings
                COMPLETED BY PER-BASIC-UNALIGNED
        END
```

D.3 Примеры явно генерируемой структуры

Примеры, описанные в D.3.1—D.3.4, показывают использование явно генерируемых структур для замены класса кодирования в неявно генерируемой структуре кодирования на одноименный класс. Затем мы образуем специализированные типы кодирования путем введения в набор объектов кодирования объекта одноименного класса. Эти примеры даются с использованием следующего формата:

- «присвоение типа АСН.1». Здесь дается исходное определение типа АСН.1;
- требование. Здесь перечисляются требуемые изменения к кодированиям, обеспеченным методом PER basic unaligned;
- изменение неявно генерируемой структуры кодирования для образования новой структуры кодирования;
- присвоения класса кодирования и объекта кодирования.

D.3.1 Последовательность с факультативными компонентами, определенными указателями

D.3.1.1 Присвоение АСН.1 равно:

```
Sequence1 ::= SEQUENCE {
component1 INTEGER OPTIONAL,
component2 INTEGER OPTIONAL,
component3 VisibleString }
```

D.3.1.2 Вместо использования побитового отображения PER для двух компонентов целочисленного типа, обозначенных **OPTIONAL**, присутствие и позиция таких компонентов определяется указателями в начале кодирования последовательности. Каждый указатель содержит 0 (компонент отсутствует) или относительное смещение для кодирования компонента, который начинается на границе октета.

D.3.1.3 Класс кодирования **#INTEGER** заменяется на «**#Integer-with-pointer-concat**» в объекте кодирования «**sequence1-encoding**». Класс «**#Integer-with-pointer-concat**» определен как структура конкатенации, содержащая один элемент, который является замененным элементом, объединенным с классом «**#Integeroptionality**» в категории «функциональные возможности».

D.3.1.4 Затем определены два объекта кодирования. Первый «**integer-with-pointer-concat-encoding**» класса «**#Integer-with-pointer-concat**» получает три параметра: замененный элемент, указатель и текущий комбинированный набор объектов кодирования (см. 22.1.3.7). Второй «**integer-optionality-encoding**» класса «**#Integer-optionality**» получает один параметр: указатель, который используется для определения присутствия компонента. Так как **BASIC-PER-UNALIGNED** не содержит объекта кодирования класса **#CONCATENATION** с факультативными компонентами, требуется определить третий объект кодирования класса **#CONCATENATION**. Этот объект «**concat**» использует безусловную (по умолчанию) установку.

D.3.1.5 Присвоениями класса кодирования и объекта кодирования являются:

```
sequence1-encoding #SEQUENCE ::= {
    REPLACE OPTIONALS
            WITH #Integer-with-pointer-concat
            ENCODED BY integer-with-pointer-concat-encoding
                        INSERT AT HEAD #Pointer
    ENCODING-SPACE
            SIZE variable-with-determinant
            DETERMINED BY container
            USING OUTER }
#Pointer ::= #INTEGER
```

```
#Integer-with-pointer-concat {< #Element >} ::= #CONCATENATION {
        element #Element OPTIONAL-ENCODING #Integer-optionality }
#Integer-optionality ::= #OPTIONAL
integer-optionality-encoding{< REFERENCE: start-pointer>} #Integer-optionality ::= {
  ALIGNED TO ANY octet
  START-POINTER start-pointer
  PRESENCE DETERMINED BY pointer}
integer-with-pointer-concat-encoding {< #Element, REFERENCE:pointer, #ENCOINGS:EncodingObjectSet >}
  #Integer-with-pointer-concat{< #Element >} ::= {
  ENCODE STRUCTURE {
    element USE-SET OPTIONAL-ENCODING integer-optionality-encoding{< pointer >}}
  WITH EncodingObjectSet}
concat #CONCATENATION ::= {
        ENCODING-SPACE }
```

D.3.2 Добавление булева типа в качестве детерминанта присутствия

D.3.2.1 Присвоение ACH.1 равно:

```
Sequence2 ::= SEQUENCE {
  component1 BOOLEAN OPTIONAL,
component2 INTEGER,
component3 VisibleString OPTIONAL }
```

D.3.2.2 Вместо использования побитового отображения PER для компонентов, обозначенных «**OPTIONAL**», присутствие факультативного компонента связывается со значением уникального (однозначного) бита присутствия, который равен 1 (компонент отсутствует) или 0 (компонент присутствует). В этом случае бит присутствия инвертируется.

D.3.2.3 Структуры кодирования и объекты кодирования определяются следующим образом.

Класс кодирования **#OPTIONAL** переименовывается на **#Sequence2-optional** в разделе «**RENAMES**» (см. D.3.7). Поэтому класс «**#Sequence2**» неявно заменяется на:

```
#Sequence2 ::= #SEQUENCE {
    component1 #BOOL OPTIONAL-ENCODING #Sequence2-optional,
    component2 #INTEGER,
component3 #VisibleString OPTIONAL-ENCODING #Sequence2-optional},
```

где:

```
#Sequence2-optional ::= #OPTIONAL
```

Затем определяется объект кодирования класса «**#Sequence2-optional**»; этот объект с помощью группы замены заменяет определение кодирования компонента (см. 23.11.3.2) на класс «**Optional-with-determinant**».

```
sequence2-optional-encoding #Sequence2-optional ::= {
        REPLACE STRUCTURE
        WITH #Optional-with-determinant
                ENCODED BY optional-with-determinant-encoding}
```

Этот класс, параметризованный исходным компонентом, принадлежит к категории «конкатенация» и имеет два компонента: детерминант (булева типа) и исходный компонент.

```
#Optional-with-determinant{< #Element >} ::= #CONCATENATION {
    determinant #BOOLEAN,
    component #Element OPTIONAL-ENCODING #Presence-determinant},
```

где:

```
#Presence-determinant ::= #OPTIONAL
```

Затем определяется объект кодирования класса «**#Optional-with-determinant**»; этот объект имеет два фиктивных параметра: класс компонента и набор объектов кодирования, используемый для кодирования всего, кроме детерминанта и функциональных возможностей компонента:

```
optional-with-determinant-encoding {< #Element, #ENCODINGS: Sequence2-combined-encoding-object-set >}
        #Optional-with-determinant {< #Element >} ::= {
  ENCODE STRUCTURE {
    determinant determinant-encoding,
```

```
component USE-SET
                    OPTIONAL-ENCODING if-component-present-encoding{< determinant >} }
    WITH Sequence2-combined-encoding-object-set }
```

Кодирование полностью указывается описанием объектов кодирования «**if-component-present-encoding**» и «**determinant-encoding**»:

```
if-component-present-encoding {<REFERENCE:presence-bit>} #Presence-determinant ::= {
        PRESENCE
                DETERMINED BY field-to-be-set
                USING presence-bit}
determinant-encoding #BOOLEAN ::= {
        ENCODING-SPACE
            SIZE 1
            MULTIPLE OF bit
        TRUE-PATTERN bits:'0'B
        FALSE-PATTERN bits:'1'B}
```

D.3.3 Последовательность с факультативными компонентами, указанными уникальным тегом и разграниченными полем длины

D.3.3.1 Присвоения ACH.1 равны:

```
Octet3 ::= OCTET STRING (CONTAINING Sequence3)
Sequence3 ::=SEQUENCE {
  component1 [0] BIT STRING (SIZE(0..2047)) OPTIONAL,
  component2 [1] OCTET STRING (SIZE(0..2047)) OPTIONAL,
  component3 [2] VisibleString (SIZE(0..2047)) OPTIONAL }
```

D.3.3.2 Каждый компонент указывается тегом из четырех битов, а общая длина этой последовательности указывается полем из одиннадцати битов, которое предшествует кодированию первого компонента.

D.3.3.3 Классы кодирования **#OCTETS**, **#OPTIONAL** и **#TAG** переименовываются соответственно в **#Octets3**, **#Sequence3-optional** и **#TAG-4-bits** в разделе «**RENAMES**» (см. D.3.7). Затем объекты кодирования новых классов кодирования определяются.

D.3.3.4 Присвоениями класса кодирования и объекта кодирования для цепочки октетов являются:

```
#Octets3 ::= #OCTET-STRING
octets3-encoding #Octets3 ::= {
        REPETITION-ENCODING {
                REPLACE STRUCTURE
                        WITH #Octets-with-length
                ENCODED BY octets-with-length-encoding}}
#Octets-with-length{< #Element >} ::= #CONCATENATION {
        length #INT(0..2047),
        octets #Element}
octets-with-length-encoding{< #Element >} #Octets-with-length{< #Element >} ::= {
        ENCODE STRUCTURE {
            octets octets-encoding{< length >}}
        WITH PER-BASIC-UNALIGNED}
octets-encoding{< REFERENCE:length >} #OCTETS ::= {
        REPETITION-ENCODING {
                    REPETITION-SPACE
                        SIZE variable-with-determinant
                DETERMINED BY field-to-be-set
                USING length} }
```

D.3.3.5 Присвоениями класса кодирования и объекта кодирования для последовательности являются:

```
        sequence3-encoding #Sequence3 ::= {
                ENCODE STRUCTURE {
```

```
                    STRUCTURED WITH sequence3Structure-encoding }
            WITH Sequence3-encodings
                    COMPLETED BY PER-BASIC-UNALIGNED }

        Sequence3-encodings #ENCODINGS ::= {
                sequence3-optional-encoding |
                tag-4-bits-encoding }

        #Sequence3-optional ::= #OPTIONAL
        sequence3-optional-encoding #Sequence3-optional ::= {
          PRESENCE
          DETERMINED BY container
                  USING OUTER}
        #TAG-4-bits ::= #TAG
        tag-4-bits-encoding #TAG-4-bits ::= {
          ENCODING-SPACE
             SIZE 4}
```

Следующий объект кодирования класса #OUTER указывает, что декодер должен игнорировать биты после кодирования последовательности, добавленной, чтобы иметь несколько октетов.

```
outer-encoding #OUTER ::= {
        ADDED BITS DECODING silently-ignore }
```

D.3.4 Тип «последовательность-из» с подсчетом

D.3.4.1 Присвоение ACH.1 равно:

```
        SequenceOfIntegers ::= SEQUENCE(SIZE(0..63)) OF INTEGER(0..1023)
```

D.3.4.2 Число элементов кодируется в шестибитовом поле, которое предшествует кодированию первого элемента.

D.3.4.3 Класс кодирования **#SEQUENCE-OF** переименовывается в **#Sequenceof** в разделе «**RENAMES**» (см. D.3.7). Определяется объект кодирования нового класса кодирования. Присвоениями класса кодирования и объекта кодирования являются:

```
    #SequenceOf ::= #REPETITION
    sequenceOf-encoding #SequenceOf ::= {
            REPETITION-ENCODING {
              REPLACE STRUCTURE
              WITH #SequenceOf-with-count
              ENCODED BY sequenceOf-with-count-encoding}}
#SequenceOf-with-count{< #Element >} ::= #CONCATENATION {
        count #INT(0..63),
        elements #Element }
sequenceOf-with-count-encoding{< #Element >} #SequenceOf-with-count{< #Element >} ::= {
        ENCODE STRUCTURE {
            elements {
                        ENCODE STRUCTURE {
                                STRUCTURED WITH elements-encoding{< count >}}
                        WITH PER-BASIC-UNALIGNED}}
        WITH PER-BASIC-UNALIGNED}
elements-encoding{< REFERENCE:count >} #REPETITION ::= {
        REPETITION-ENCODING {
            REPETITION-SPACE
                SIZE variable-with-determinant
        MULTIPLE OF repetitions
                DETERMINED BY field-to-be-set
                USING count}}
```

D.3.4.4 Поле подсчета кодируется по правилам кодирования PER для целочисленного типа с диапазоном значений (0—63), что дает шестибитовое поле.

D.3.5 Набор объектов кодирования

Набор объектов кодирования содержит объекты кодирования классов, определенные в модуле EDM (только первый из них содержит объект кодирования класса #SEQUENCE).

```
Example3Encodings-1                #ENCODINGS ::= {
                sequence1-encoding                |
Example3Encodings-2                #ENCODINGS ::= {
                concat                            |
                sequence2-optional-encoding       |
                octets3-encoding                  |
                sequenceOf-encoding               |
                sequence3-encoding                |
                        outer-encoding }
```

D.3.6 Определения ACH.1

Этот модуль объединяет определения ACH.1 из D.3.1—D.3.4, которые будут кодироваться согласно объектам кодирования, определенным в EDM из D.3.7.

```
Example3-ASN1-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module3(9)}
        DEFINITIONS
        AUTOMATIC TAGS ::=
        BEGIN
        Sequence1 ::=           SEQUENCE {
                component1      BOOLEAN         OPTIONAL,
                component2      INTEGER         OPTIONAL,
                component3      VisibleString   OPTIONAL }
    -- и т. д.
    END
```

D.3.7 Определения EDM

```
Example3-EDM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module3(10)}
ENCODING-DEFINITIONS ::=
BEGIN
EXPORTS Example3Encodings;
RENAMES
  #OPTIONAL AS #Sequence2-optional
    IN #Sequence2
  #OCTET-STRING AS #Octets3
    IN ALL
  #OPTIONAL AS #Sequence3-optional
    IN #Sequence3
  #TAG AS #TAG-4-bits
    IN #Sequence3
FROM Example3-ASN1-Module { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module3(9)};
Example3Encodings-1 #ENCODINGS ::= {
        sequence1-encoding }
Example3Encodings-2 #ENCODINGS ::= {
        concat |
        -- и т. д.
        sequenceOf-encoding }
-- и т. д.
END
```

D.3.8 Определения ELM

Описанный ниже ELM связан с модулем ACH.1, определенным в D.3.6. и с EDM, определенным в D.3.7.

```
Example3-ELM {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) elm-module3(8)}
LINK-DEFINITIONS ::=
BEGIN
```

```
IMPORTS Example3Encodings, #Sequence1, #Sequence2, #Octet3, #Sequence3, #SequenceOfIntegers
        FROM Example3-EDM
                { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module3(10) };

ENCODE #Sequence1,
        WITH Example3Encodings-1
        COMPLETED BY PER-BASIC-UNALIGNED

ENCODE #Sequence2, #Octet3, #Sequence3, #SequenceOfIntegers
        WITH Example3Encodings-2
        COMPLETED BY PER-BASIC-UNALIGNED
END
```

D.4 Пример кодирования детерминанта «бит-еще»

D.4.1 Описание проблемы

D.4.1.1 Этот пример взят из Рек. МСЭ-Т Q.763 (Система сигнализации № 7 — форматы и коды подсистемы пользователя ЦСИС).

D.4.1.2 Существует потребность вырабатывать описанное ниже кодирование в виде серии октетов:

8	7	6	5	4	3	2	1
Индикатор расширения	Резерв		Профиль протокола				

D.4.1.3 Бит 8 является «индикатором расширения». Если он равен 0, то имеется следующий октет в том же формате. Если он равен 1, то этот октет является последним в серии.

Примечание — Кодирование PER для булевых значений дает 1 для **TRUE** и 0 для **FALSE**, а ECN требует, чтобы последний элемент выдавал **FALSE**, а более ранние элементы — **TRUE**. Поэтому, если мы используем для «бит-еще» булево значение с кодированием PER, то мы должны применить преобразователь «**not**».

D.4.1.4 Это является традиционным использованием «бит-еще», несмотря на возможно необычные нуль для «еще» и единица для «последний».

D.4.1.5 Пример мог бы упроститься, если бы «индикатор расширения» поменял использование нуля и единицы и если бы не было битов «резерв», но здесь предпочтение отдано использованию реального примера.

D.4.1.6 Имеются четыре подхода к решению этой проблемы.

D.4.1.7 Первый подход: включить в спецификацию АСН.1 компонент для обеспечения детерминанта «бит-еще» (см. D.4.2). Этот подход отвергается по двум причинам. Первая состоит в том, что определение типа АСН.1 содержит компонент, который не переносит прикладную семантику. Вторая состоит в том, что этот подход требует применения установки (избыточной) этого поля в правильное состояние в каждом элементе при повторении детерминанта «бит-еще».

D.4.1.8 Второй подход: использовать отображения значения из неявно генерируемой структуры в определяемую пользователем структуру кодирования, которая содержит детерминант «бит-еще» (см. D.4.3).

D.4.1.9 Третий подход: использовать механизм замены для включения детерминанта «бит-еще» (см. D.4.4).

D.4.1.10 Четвертый подход: использовать вставку головного узла в виде детерминанта «бит-еще» (это здесь не иллюстрируется).

D.4.1.11 Каждый из трех последних подходов имеет свои преимущества, а выбор между ними является в значительной степени вопросом стиля.

D.4.2 Использование АСН.1 для обеспечения детерминанта «бит-еще»

D.4.2.1 При этом подходе АСН.1 отражает все поля в кодировании. Подход обычно считается «нечистым», если поля, которые должны быть видимы только при кодировании, будут видимы для приложения, что уменьшает «сокрытие информации», которое является сильной стороной АСН.1. В этом случае АСН.1 будет равно:

```
ProfileIndication ::= SEQUENCE OF
        SEQUENCE {
                more-bit            BOOLEAN,
                reserved            BIT STRING (SIZE (2)),
                protocol-Profile-ID INTEGER (0..31) }
```

D.4.2.2 Неявно генерируемой структурой кодирования является:

```
#ProfileIndication ::= #SEQUENCE-OF {
        #SEQUENCE {
                more-bit                #BOOLEAN,
                reserved                #BIT-STRING (SIZE (2)),
                protocol-Profile-ID     #INTEGER (0..31) } }
```

D.4.2.3 Сначала мы образуем обобщенный объект кодирования для **#SEQUENCE-OF**, который использует «бит-еще» в поле, указанном в виде параметра объекта кодирования и имеющим булево значение **TRUE** (кодируемое одиночным битом «1» в PER) для последнего элемента:

```
more-bit-encoding {< REFERENCE:more-bit >} #SEQUENCE-OF ::= {
        REPETITION-ENCODING {
                REPETITION-SPACE
                        SIZE variable-with-determinant
                        DETERMINED BY flag-to-be-set
                                USING more-bit
                                        ENCODER-TRANSFORMS
                                                { { BOOL-TO-BOOL AS logical:not } } } }
```

D.4.2.4 Этот объект кодирования используется также в D.4.3 и D.4.4, так как он дает фундаментальное описание кодирования, необходимого для повторения.

D.4.2.5 При первом подходе (простом, но «нечистом»!) мы можем теперь определить наш объект кодирования для **#ProfileIndication**, используя **ENCODE STRUCTURE**, и применить этот объект кодирования в ELM, завершая данный пример. Этот объект кодирования определяется как:

```
profileIndicationEncoding #ProfileIndication ::= {
        ENCODE STRUCTURE {
                STRUCTURED WITH
                        more-bit-encoding {< more-bit >} }
        WITH PER-BASIC-UNALIGNED }
```

D.4.3 Использование отображений значения для обеспечения детерминанта «бит-еще»

D.4.3.1 При этом подходе мы скрываем структуру кодирования в определении ECN для определяемой пользователем структуры кодирования и используем отображение значения при помощи полей сопоставления, давая возможность кодированию определяемой пользователем структуры кодирования закодировать упрощенное определение типа ACH.1.

D.4.3.2 Теперь определение типа ACH.1 равно:

```
ProfileIndication2 ::= SEQUENCE OF
        protocol-Profile-ID INTEGER (0..31)
```

D.4.3.3 Это имеет следующую неявно генерируемую структуру кодирования (к которой мы применяем наши кодирования в ELM):

```
#ProfileIndication2 ::= #SEQUENCE-OF {
        protocol-Profile-ID #INTEGER (0..31) }
```

D.4.3.4 Мы определяем структуру кодирования для кодирования, нужного нам, аналогично тому, как мы записали для ACH.1 в первом подходе (см. D.4.2.1), за исключением того, что мы используем **#PAD** для зарезервированных битов:

```
#ProfileIndicationStruct ::= #SEQUENCE-OF {
        #SEQUENCE {
                more-bit-field          #BOOLEAN,
                reserved                #PAD,
                protocol-Profile-ID     #INTEGER (0..31) } }
```

D.4.3.5 Чтобы мы могли завершить кодирование, нам нужен теперь объект кодирования для двухбитового **#PAD**:

```
pad-encoding #PAD ::= {
    ENCODING-SPACE SIZE 2
    PATTERN bits:'00'B }
```

Примечание — В пункте 23.12.4.2 указано, что декодеры должны принимать любое значение для битов **#PAD**, что нам здесь необходимо, поэтому нам не требуются особые кодер-декодер.

D.4.3.6 Мы определяем объект кодирования для нашей структуры почти так же, как в первом подходе (см. D.4.2.5):

```
profileIndicationStructEncoding #ProfileIndicationStruct ::= {
ENCODE STRUCTURE {
STRUCTURED WITH
more-bit-encoding {< more-bit-field >} }
WITH {pad-encoding} COMPLETED BY PER-BASIC-UNALIGNED }
```

D.4.3.7 Наконец, мы используем отображение значения от неявно генерируемой структуры к нашей явно генерируемой структуре, чтобы определить наше окончательное кодирование:

```
profileIndication2Encoding #ProfileIndication2 ::= {
USE #ProfileIndicationStruct
MAPPING FIELDS
WITH profileIndicationStructEncoding }
```

D.4.4 Использование механизма замены для обеспечения детерминанта «бит-еще»

D.4.4.1 При нашем последнем подходе мы определяем обобщенное кодирование «последовательности-из», которое можно применять к любой «последовательности-из». Для этого нам нужна параметризованная структура кодирования:

```
#SequenceOfStruct {< #Component >} ::=
#SEQUENCE {
more-bit-field #BOOLEAN,
reserved #PAD,
sequence-of-component #Component }
```

D.4.4.2 Мы определяем наше кодирование «последовательности-из» для выполнения замены компонента на эту структуру, учитывая кодирование «бит-еще» и используя определенное кодирование PAD:

```
sequence-of-encoding #SEQUENCE-OF ::= {
REPETITION-ENCODING {
REPLACE COMPONENT WITH #SequenceOfStruct
REPETITION-SPACE
SIZE variable-with-determinant
DETERMINED BY flag-to-be-set
USING more-bit-field
ENCODER-TRANSFORMS
{ { BOOL-TO-BOOL AS logical:not } } } }
```

D.4.4.3 Когда это применяется в ELM, то для того, чтобы завершить кодирование, дающее желательный результат, используется «**COMPLETED BY PER-BASIC-UNALIGNED**» в виде комбинированного набора объектов кодирования.

D.5 Традиционный протокол, описанный с помощью табличной нотации

D.5.1 Введение

D.5.1.1 Цель примера в этом разделе — показать, как конструировать определения ECN для протокола, кодирования сообщения которого специфицированы с использованием изображений «биты и байты» и табличной нотации.

Следующие таблицы показывают содержимое сообщений (полностью показано только «Сообщение 1»):

Сообщение 1:

	8	7	6	5	4	3	2	1
Октет 1	Идентификатор сообщения							
Октет 2	A			b-flag	c-len			Резерв

Окончание таблицы

	8	7	6	5	4	3	2	1
Октет 3	b1		b2	Резерв	b3		Резерв	
…								
Октет Y	c1				c2			
Октет Y+1	c3						Резерв	
…								
Октет Z	d1	d2			d3			Резерв

Сообщение 2:

	8	7	6	5	4	3	2	1
Октет 1	Идентификатор сообщения							
Октет 2…	Что-либо — 1							

Сообщение 3:

	8	7	6	5	4	3	2	1
Октет 1	Идентификатор сообщения							
Октет 2…	Что-либо — 2							

D.5.1.2 Все сообщения имеют общую заголовочную часть (затемненную в таблицах). В этом примере она используется только для идентификации сообщения.

D.5.1.3 Сообщение 1 имеет поля трех видов:

- обязательные поля («a»);
- обязательные поля, являющиеся детерминантами для других полей («b-flag», «c-len»);
- факультативные поля («b», «c» и «d»).

D.5.1.4 Все поля «b», «c» и «d» должны начинаться на границе октета.

D.5.1.5 Поля «b», «c» и «d» состоят из подполей («b1», «b2», «b3», «c1» и т. д.) фиксированной длины. Кроме того, поля «c» и «d» могут появляться несколько раз (выше показано только одно появление). Поле «b3» должно начинаться на границе полуоктета.

D.5.1.6 Присутствие факультативного компонента указывается разными методами:

- поле «b» присутствует, если значение поля «b-flag» равно 1;
- поле «d» присутствует, если в сообщении есть неиспользованные октеты.

D.5.1.7 Длина поля, которое может появляться несколько раз, определяется разными методами:

- число повторений поля «c» руководит полем детерминанта «c-len»;
- число повторений поля «d» определяется концом сообщения.

D.5.1.8 Ниже приводится модуль АСН.1, содержащий определения показанных выше структур сообщения. Приняты следующие решения при проектировании:

- имеется один тип с вложением («инкапсуляцией»), который содержит общие определения для всех сообщений;
- вспомогательные поля детерминантов в сообщениях видны на уровне АСН.1. Заметим, что это сделано для простоты описания в этом примере, а в обычной практике такие поля сохраняются вне определения АСН.1, если они не переносят реальную прикладную семантику;
- расширяемость выражается в форме комментариев;
- заполнение не видно.

D.5.1.9 Модуль АСН.1 равен:

```
LegacyProtocol-ASN1-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module4(11)}
    DEFINITIONS AUTOMATIC TAGS ::=
    BEGIN
    LegacyProtocolMessages ::= SEQUENCE {
            message-id ENUMERATED {message1, message2, message3},
            messages CHOICE {
```

```
                    message1 Message1,
                    message2 Message2,
                    message3 Message3}}
            -- CHOICE ограничен значением message-id.
Message1 ::= SEQUENCE {
            a A,
            b-flag BOOLEAN,
            c-len INTEGER (0..max-c-len),
            b B OPTIONAL, -- определяется значением "b-flag"
            c C, -- определяется значением "c-len"
            d D OPTIONAL} -- определяется концом PDU
A ::= INTEGER (0..7) -- Значения 5—7 зарезервированы для будущего
-- использования. Системы версии 1
-- должны рассматривать 5—7 как 4.
B ::= SEQUENCE {
            b1 ENUMERATED { e0, e1, e2, e3 },
            b2 BOOLEAN,
            b3 INTEGER (0..3) }
C ::= SEQUENCE (SIZE (0..max-c-len)) OF C-elem
C-elem ::= SEQUENCE {
            c1 BIT STRING (SIZE (4)),
            c2 INTEGER (0..1024) }
D ::= SEQUENCE (SIZE (0..max-d-len)) OF D-elem
D-elem ::= SEQUENCE {
            d1 BOOLEAN,
            d2 ENUMERATED { f0, f1, f2, f3, f4, f5, f6, f7 },
            d3 INTEGER (0..7) }
max-c-len INTEGER ::= 7
max-d-len INTEGER ::= 20
    Message2 ::= SEQUENCE {
                -- quelque chose 1 -- }
    Message3 ::= SEQUENCE {
                -- quelque chose 2 -- }
    END
```

D.5.1.10 Модуль EDM в D.5.7 содержит определения кодирования для сообщений, указанных в модуле ACH.1 «**LegacyProtocol-ASN1-Module**». Приняты следующие решения при проектировании:

- заполнение внутри октетов явно указывается как поля заполнения;
- заполнение для выравнивания не указывается как явные поля заполнения.

D.5.2 Определение кодирования для структуры сообщения верхнего уровня

D.5.2.1 Объект кодирования «**legacyProtocolMessagesEncoding**» указывает, как кодируются общие части сообщений традиционных протоколов. Идентификатор сообщения указывается в ACH.1 в виде перечислительного типа. PER basic unaligned кодирует «**message-id**» с использованием минимального числа битов (то есть 2), но здесь мы хотели бы иметь их с кодированием 8 битами. Кроме того, мы указываем, что «**message-id**» должен использоваться в качестве детерминанта для «**messages**».

D.5.2.2 Объект кодирования «**legacyProtocolMessagesEncoding**» равен:

```
legacyProtocolMessagesEncoding #LegacyProtocolMessages ::= {
        ENCODE STRUCTURE {
                message-id {
                        ENCODING {
                                ENCODING-SPACE
                                        SIZE 8}},
            messages {
                        ENCODE STRUCTURE {
                                STRUCTURED WITH {
                                    ALTERNATIVE
                            DETERMINED BY field-to-be-used
                            USING message-id}}
        WITH PER-BASIC-UNALIGNED}}
        WITH PER-BASIC-UNALIGNED}
```

D.5.3 Определение кодирования для структуры сообщения

D.5.3.1 Объект кодирования «**message1Encoding**» указывает, как должны кодироваться значения из «**Message1**»:

- поле «**b**» присутствует, если поле «**b-flag**» содержит значение **TRUE**;
- поле «**c**» присутствует, если поле «**c-len**» не содержит значения 0. Поле «**c-len**» руководит также числом элементов в «**c**»;
- поле «**d**» присутствует, если имеются еще октеты в кодировании сообщения.

D.5.3.2 Объектом кодирования для «Message1» является:

```
message1Encoding #Message1 ::= {
        ENCODE STRUCTURE {
            b b-encoding
              OPTIONAL-ENCODING {
                PRESENCE
                    DETERMINED BY field-to-be-used
                    USING b-flag},
            c octet-aligned-seq-of-with-ext-determinant{< c-len >},
            d octet-aligned-seq-of-until-end-of-container
                                OPTIONAL-ENCODING USE-SET}
        WITH PER-BASIC-UNALIGNED}
```

D.5.4 Кодирование для типа последовательности «B»

D.5.4.1 Заполнение в один бит вводится между полями «**b2**» и «**b3**» («**aux-reserved**»). Кодирование для «**B**» выравнивается по октетам.

D.5.4.2 Кодированием для «**B**» является:

```
b-encoding #B ::= {
        ENCODE STRUCTURE {
            -- Компоненты
            b3 {
                ALIGNED TO NEXT nibble
                ENCODING {
                    ENCODING-SPACE
                        SIZE 2
                            MULTIPLE OF bit }}
            -- Структура
            STRUCTURED WITH {
                ALIGNED TO NEXT octet
                ENCODING-SPACE
                    SIZE self-delimiting-values
                        MULTIPLE OF bit }}
        -- Остаток
        WITH PER-BASIC-UNALIGNED}
```

D.5.5 Кодирование выровненного по октетам типа «последовательность-из» с детерминантом длины

D.5.5.1 Один из типов «последовательность-из», используемых в традиционном протоколе, имеет явный детерминант длины.

D.5.5.2 Кодирование выравнивается по октетам. Подсчет числа элементов определяется полем «**len**».

```
octet-aligned-seq-of-with-ext-determinant{< REFERENCE : len >} #REPETITION ::= {
        REPETITION-ENCODING {
            ALIGNED TO NEXT octet
            REPETITION-SPACE
                SIZE variable-with-determinant
                DETERMINED BY field-to-be-used
                USING len}}
```

D.5.6 Кодирование выровненного по октетам типа «последовательность-из», которое продолжается до конца PDU

D.5.6.1 Кодирование выравнивается по октетам. Число элементов определяется концом PDU.

D.5.6.2 Объектом кодирования является:

```
octet-aligned-seq-of-with-ext-determinant{< REFERENCE : len >} #REPETITION ::= {
        REPETITION-ENCODING {
            ALIGNED TO NEXT octet
            REPETITION-SPACE
                SIZE variable-with-determinant
                DETERMINED BY field-to-be-used
                USING len}}
```

D.5.7 Определения EDM

Определениями EDM являются:

```
LegacyProtocol-EDM-Module {joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module4(13)}
    ENCODING-DEFINITIONS ::=
    BEGIN
    EXPORTS LegacyProtocolEncodings;
    IMPORTS #LegacyProtocolMessages
    FROM LegacyProtocol-ASN1-Module
    { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module4(11) };
    LegacyProtocolEncodings #ENCODINGS ::= {
            legacyProtocolMessagesEncoding |
            message1Encoding }
    -- и т. д.
    END
```

D.5.8 Определения ELM

ELM для традиционного протокола будет следующим:

```
LegacyProtocol-ELM-Module { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) elm-module4(12) }
LINK-DEFINITIONS ::=
BEGIN
IMPORTS
    LegacyProtocolEncodings FROM LegacyProtocol-EDM-Module
            { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) edm-module4(13) }
    #LegacyProtocolMessages FROM LegacyProtocol-ASN1-Module
            { joint-iso-itu-t(2) asn1(1) ecn(4) examples(5) asn1-module4(11) };
ENCODE #LegacyProtocolMessages WITH LegacyProtocolEncodings
    COMPLETED BY PER-BASIC-UNALIGNED
  END
```

Приложение Е
(справочное)

Поддержка кодовых последовательностей Хаффмана

Е.1 Кодовые последовательности Хаффмана являются оптимальными кодовыми последовательностями для ограниченного набора целочисленных значений, при условии, что частота, с которой каждое значение будет передаваться, известна.

Е.2 Такие кодовые последовательности являются саморазграничивающими (не нуждаются в детерминанте длины) и используют малое число битов для частых значений и большее число битов для менее частых значений.

Е.3 Имеется много возможных кодов Хаффмана. Например, взяв любое такое кодирование, просто измените все «1» на «0» и наоборот, при этом вы получите другое кодирование Хаффмана (но также эффективное). Можно еще делать более сложные изменения для образования других кодов Хаффмана, которые будут тоже эффективными.

Е.4 Для кодирования Хаффмана, которое будет эффективным для декодеров, желательно, чтобы при кодировании последовательных целочисленных значений в одно и то же число битов такие биты определяли бы последовательные целочисленные значения, рассматривая кодирование как кодирование положительного целого числа.

Е.5 Один из кодов ECN Хаффмана был определен как имеющий это свойство, и был разработан макрос Microsoft Word 97, который будет генерировать синтаксис для отображения «MappingIntToBits» (см. 19.7), который как оптимален, так и прост для декодирования.

Е.6 Доступна версия этого приложения, содержащая кнопку «macro», которая будет выдавать спецификацию целочисленных значений, которые должны кодироваться, и частоты их передачи и будет генерировать в режиме инлайн формальную спецификацию отображения, соответствующую нотации ECN (эта версия настоящего приложения с соответствующим макросом свободно доступна на web-сайте МСЭ http://www.itu.int/ITU-T/publications/recs.html под названием «X.692 Recommendation», а также на web-сайте ИСО http://www.iso.ch/iso/en/ittf/-PubliclyAvailableStandards/ c034390_ISO_8825-3_2003(E)_Annex_E.html).

Е.7 Ниже в тексте содержатся три примера спецификации ECN Хаффмана.

Е.8 В версии с макросом двойной щелчок мышью на:

будет добавлять к тексту спецификации отображения ECN Хаффмана.

Е.9 Пользователь версии с макросом может пожелать изменить спецификацию значений, подлежащих отображению, и частот их передачи, чтобы увидеть кодирования, которые вырабатываются в разных случаях.

Примечание — В версии с макросом, когда спецификации кодирования были созданы, они могут быть удалены, спецификация ECN Хаффмана может быть изменена, а кнопка макроса — щелкнута снова.

Е.10 Неформальный синтаксис для спецификации ECN Хаффмана будет ясен из последующих примеров. Все строчки начинаются указателем комментария ACH.1 («--»).

Е.11 Первая строка (если макрос будет использоваться) должна содержать только «ECN Huffman», перед которым имеются два дефиса и один пробел, а последующие строки не чувствительны к регистру клавиатуры и могут содержать больше или меньше пробелов.

Е.12 Вторая строка необходима, она указывает наименьшее и наибольшее значения, подлежащие отображению. Диапазон (верхняя граница минус нижняя граница) ограничен до 1000, но может содержать отрицательные значения. Нет необходимости отображать все значения в диапазоне.

Е.13 Проценты даются либо для одиночных значений, либо для диапазонов значений. Нет необходимости добавлять проценты вплоть до 100 %, но если это не сделано, то выдается предупреждение.

Е.14 Строка «REST» факультативна, она дает частоту передачи для каких-либо явно не перечисленных значений диапазона. Если этой строки нет, то отображенными значениями будут только те, которые явно указаны.

Е.15 Последняя строка обязательна, она должна содержать «End Definition» (прописными или строчными буквами). Формальная спецификация кодирования ECN вводится (макросом) после этой строки.

Е.15.1 Первым примером является:

```
my-int-encoding1 #My-Special-1 ::=
{ USE #BITS
    -- ECN Хаффман
```

```
        -- ДИАПАЗОН (–1—10)
        -- –1 занимает 20 %
        -- 0 занимает 15 %
        -- 1 занимает 25 %
        -- (3—6) занимают 10 %
        -- Оставшиеся занимают 2 %
        -- Определение конца
        -- Отображения, созданные "ECN Public Domain
        -- Software for Huffman encodings, version 1"
            MAPPING TO BITS {
                -1 TO '11'B,
                0 .. 1 TO '01'B .. '10'B,
                2 TO '0000001'B ,
                3 .. 5 TO '0001'B .. '0011'B,
                6 TO '00001'B,
                7 .. 8 TO '0000010'B .. '0000011'B,
                9 .. 10 TO '00000000'B .. '00000001'B
            }
    WITH my-self-delim-bits-encoding }
```

E.15.2 Вторым примером является:

```
    my-int-encoding2 #My-Special-2 ::=
    { USE #BITS
            -- ECN Хаффман
            -- ДИАПАЗОН (–10—10)
            -- –10 занимает 20 %
            -- 1 занимает 25 %
            -- 5 занимает 15 %
            -- (7—10) занимают 10 %
            -- Определение конца
            -- Отображения, созданные "ECN Public Domain
            -- Software for Huffman encodings, version 1"
              MAPPING TO BITS {
                    -10 TO '11'B ,
                    1 TO '10'B ,
                    5 TO '01'B ,
                    7 .. 10 TO '0000'B .. '0011'B
              }
    WITH my-self-delim-bits-encoding }
```

E.15.3 Третьим примером является:

```
    my-int-encoding3 #My-Special-3 ::=
    { USE #BITS
        -- ECN Хаффман
        -- ДИАПАЗОН (0—1000)
        -- (0—63) занимают 100 %
        -- Оставшиеся занимают 0 %
        -- Определение конца
        -- Отображения, созданные "ECN Public Domain
        -- Software for Huffman encodings, version 1"
            MAPPING TO BITS {
                0 .. 62 TO '000001'B .. '111111'B,
                63 TO '0000001'B ,
                64 .. 150 TO '0000000110101001'B .. '0000000111111111'B,
                151 .. 1000 TO '0000000000000000'B .. '0000000101010001'B
            }
    WITH my-self-delim-bits-encoding }
```

Приложение F
(справочное)

Дополнительная информация о нотации управления кодированием (ECN)

Дополнительную информацию и ссылки, относящиеся к нотации управления кодированием, можно найти на следующем web-сайте:

http://www.itu.int/itu-t/asn1/ecn

Приложение G
(справочное)

Сводный перечень нотации ECN

G.1 Терминальные символы

В настоящем стандарте использованы следующие терминальные символы.

G.1.1 Следующие элементы определяются в разделе 8:

anystringexceptnonecend
encodingobjectreference
encodingobjectsetreference
encodingclassreference
"::="
".."
"{"
"}"
"("
")"
","
"."
"|"
ALL
AS
BEGIN
BER
IF
IMPORTS
IN
LINK-DEFINITIONS
MAPPING
MAX
MIN
MINUS-INFINITY
NON-ECN-BEGIN
NON-ECN-END
NULL
OPTIONAL-ENCODING
OPTIONS
ORDERED
OUTER
PER-BASIC-ALIGNED
PER-BASIC-UNALIGNED

BITS
BY
CER
COMPLETED
DECODE
DER
DISTRIBUTION
ENCODE
ENCODE-DECODE
ENCODING-CLASS
ENCODING-DEFINITIONS
END
EXCEPT
EXPORTS
FALSE
FIELDS
FROM
GENERATES
PER-CANONICAL-ALIGNED
PER-CANONICAL-UNALIGNED
PLUS-INFINITY
REFERENCE
REMAINDER
RENAMES
SIZE
STRUCTURE
STRUCTURED
TO
TRANSFORMS
TRUE
UNION
USE
USE-SET
VALUES
WITH

G.1.2 Следующий элемент определен в приложении А:

REFERENCE

G.1.3 Следующие элементы определены в ИСО/МЭК 8824-1:

bstring
cstring
hstring
identifier
modulereference
number
realnumber
typereference
"-"
NULL
PLUS-INFINITY
";"
":"
ALL
EXCEPT
EXPORTS
FALSE
FROM
IMPORTS
MINUS-INFINITY
TRUE

G.1.4 Следующие элементы определены в ИСО/МЭК 8824-2 :

word
valuefieldreference
valuesetfieldreference

G.1.5 Следующие элементы определены в ИСО/МЭК 8824-4:

"{<"
">}"

G.2 Продукции

G.2.1 В настоящем стандарте использованы следующие продукции с элементами, определенными в G.1 в качестве терминальных символов:

```
        ELMDefinition ::=
                ModuleIdentifier
                LINK-DEFINITIONS
                "::="
                BEGIN
                ELMModuleBody
                END
        ELMModuleBody ::=
                Imports ?
                EncodingApplicationList
        EncodingApplicationList ::=
                EncodingApplication
                EncodingApplicationList ?
        EncodingApplication ::=
                ENCODE
                SimpleDefinedEncodingClass "," +
                CombinedEncodings
        CombinedEncodings ::=
                WITH
                PrimaryEncodings
                CompletionClause ?
        CompletionClause ::=
            COMPLETED BY
            SecondaryEncodings
PrimaryEncodings ::= EncodingObjectSet
SecondaryEncodings ::= EncodingObjectSet
EDMDefinition ::=
            ModuleIdentifier
            ENCODING-DEFINITIONS
            "::="
            BEGIN
            EDMModuleBody
            END
EDMModuleBody ::=
            Exports ?
            RenamesAndExports ?
            Imports ?
            EDMAssignmentList ?
EDMAssignmentList ::=
            EDMAssignment
            EDMAssignmentList ?
EDMAssignment ::=
            EncodingClassAssignment
          | EncodingObjectAssignment
          | EncodingObjectSetAssignment
          | ParameterizedAssignment
RenamesAndExports ::=
            RENAMES
            ExplicitGenerationList ";"
ExplicitGenerationList ::=
            ExplicitGeneration
            ExplicitGenerationList ?
ExplicitGeneration ::=
            OptionalNameChanges
            FROM GlobalModuleReference
OptionalNameChanges ::=
            NameChanges | GENERATES
        NameChanges ::= NameChange NameChanges ?
        NameChange ::=
                OriginalClassName
                AS
```

```
                NewClassName
                IN
                NameChangeDomain
        OriginalClassName ::= SimpleDefinedEncodingClass | BuiltinEncodingClassReference
        NewClassName ::= encodingclassreference
        NameChangeDomain ::=
                IncludedRegions
                Exception ?
        Exception ::=
                EXCEPT
                ExcludedRegions
        IncludedRegions ::=
                ALL | RegionList
        ExcludedRegions ::= RegionList
        RegionList ::=
                Region "," +
        Region ::=
                SimpleDefinedEncodingClass |
                ComponentReference
        ComponentReference ::=
                SimpleDefinedEncodingClass
                "."
                ComponentIdList
        ComponentIdList ::=
                identifier "." +
        EncodingClassAssignment ::=
                encodingclassreference
                "::="
                EncodingClass
        EncodingClass ::=
                BuiltinEncodingClassReference |
            EncodingStructure
    EncodingObjectAssignment ::=
            encodingobjectreference
            DefinedOrBuiltinEncodingClass
            "::="
            EncodingObject
    EncodingObjectSetAssignment ::=
            encodingobjectsetreference
            #ENCODINGS
            "::="
            EncodingObjectSet
            CompletionClause ?
    EncodingObjectSet ::=
            DefinedOrBuiltinEncodingObjectSet |
            EncodingObjectSetSpec
    EncodingStructure ::=
            TaggedStructure |
            UntaggedStructure
    TaggedStructure ::=
            "["
            TagClass
            TagValue ?
            "]"
            UntaggedStructure
    UntaggedStructure ::=
                DefinedEncodingClass
            | EncodingStructureField
            | EncodingStructureDefn
    TagClass ::=
            DefinedEncodingClass
```

```
                | TagClassReference
    TagValue ::=
            "(" number ")"
    EncodingStructureDefn ::=
            AlternativesStructure
                    | RepetitionStructure
                    | ConcatenationStructure
            AlternativesStructure ::=
                    AlternativesClass
                    "{"
                    NamedFields
                    "}"
            AlternativesClass ::=
                     DefinedEncodingClass
                    | AlternativesClassReference
            NamedFields ::= NamedField "," +
            NamedField ::=
                    identifier
                    EncodingStructure
            RepetitionStructure ::=
                    RepetitionClass
                    "{"
                    identifier ?
                    EncodingStructure
                    "}"
                    Size?
            RepetitionClass ::=
                     DefinedEncodingClass
                    | RepetitionClassReference
            ConcatenationStructure ::=
                    ConcatenationClass
                    "{"
                    ConcatComponents
                    "}"
            ConcatenationClass ::=
                     DefinedEncodingClass
                    | ConcatenationClassReference
            ConcatComponents ::=
                    ConcatComponent "," *
            ConcatComponent ::=
            NamedField
            ConcatComponentPresence ?
ConcatComponentPresence ::=
            OPTIONAL-ENCODING
            OptionalClass
  OptionalClass ::=
             DefinedEncodingClass
            | OptionalityClassReference
  DefinedEncodingClass ::=
             encodingclassreference
            | ExternalEncodingClassReference
            | ParameterizedEncodingClass
  DefinedOrBuiltinEncodingClass ::=
             DefinedEncodingClass
            | BuiltinEncodingClassReference
  DefinedEncodingObject ::=
             encodingobjectreference
            | ExternalEncodingObjectReference
            | ParameterizedEncodingObject
  DefinedEncodingObjectSet ::=
             encodingobjectsetreference
```

```
            | ExternalEncodingObjectSetReference
            | ParameterizedEncodingObjectSet
DefinedOrBuiltinEncodingObjectSet ::=
              DefinedEncodingObjectSet
            | BuiltinEncodingObjectSetReference
BuiltinEncodingObjectSetReference ::=
              PER-BASIC-ALIGNED
            | PER-BASIC-UNALIGNED
            | PER-CANONICAL-ALIGNED
            | PER-CANONICAL-UNALIGNED
            | BER
            | CER
            | DER
ExternalEncodingClassReference ::=
                modulereference "." encodingclassreference
              | modulereference "." BuiltinEncodingClassReference
          ExternalEncodingObjectReference ::=
              modulereference "." encodingobjectreference
          ExternalEncodingObjectSetReference ::=
              modulereference "." encodingobjectsetreference
          EncodingObjectSetSpec ::=
              "{"
              EncodingObjects UnionMark *
              "}"
          EncodingObjects ::=
                DefinedEncodingObject
              | DefinedEncodingObjectSet
          UnionMark ::=
              "|" |
              UNION
          EncodingObject ::=
                DefinedEncodingObject
              | DefinedSyntax
              | EncodeWith
              | EncodeByValueMapping
              | EncodeStructure
              | DifferentialEncodeDecodeObject
              | EncodingOptionsEncodingObject
              | NonECNEncodingObject
          EncodeWith ::=
              "{" ENCODE CombinedEncodings "}"
          EncodeByValueMapping ::=
              "{"
              USE
              DefinedOrBuiltinEncodingClass
              MAPPING
              ValueMapping
              WITH
              ValueMappingEncodingObjects
          "}"
ValueMappingEncodingObjects ::=
              EncodingObject
            | DefinedOrBuiltinEncodingObjectSet
DifferentialEncodeDecodeObject ::=
          "{"
          ENCODE-DECODE
          SpecForEncoding
          DECODE AS IF
          SpecForDecoders
          "}"
SpecForEncoding ::= EncodingObject
```

```
SpecForDecoders ::= EncodingObject
EncodingOptionsEncodingObject ::=
        "{"
        OPTIONS
        EncodingOptionsList
        WITH
        AlternativesEncodingObject
        "}"
EncodingOptionsList ::= OrderedEncodingObjectList
AlternativesEncodingObject ::= EncodingObject
NonECNEncodingObject::=
        NON-ECN-BEGIN
        AssignedIdentifier
        anystringexceptnonecnend
        NON-ECN-END
EncodeStructure ::=
        "{"
        ENCODE STRUCTURE
        "{"
        ComponentEncodingList
        StructureEncoding ?
        "}"
        CombinedEncodings ?
        "}"
StructureEncoding ::=
        STRUCTURED WITH
        TagEncoding ?
        EncodingOrUseSet
ComponentEncodingList ::=
        ComponentEncoding "," *
ComponentEncoding ::=
          NonOptionalComponentEncodingSpec
        | OptionalComponentEncodingSpec
NonOptionalComponentEncodingSpec ::=
        identifier ?
        TagAndElementEncoding
OptionalComponentEncodingSpec ::=
        identifier
        TagAndElementEncoding
        OPTIONAL-ENCODING
        OptionalEncoding
TagAndElementEncoding ::=
        TagEncoding ?
        EncodingOrUseSet
TagEncoding ::= "[" EncodingOrUseSet "]"
OptionalEncoding ::= EncodingOrUseSet
EncodingOrUseSet ::=
          EncodingObject
        | USE-SET
BuiltinEncodingClassReference ::=
          BitfieldClassReference
        | AlternativesClassReference
        | ConcatenationClassReference
        | RepetitionClassReference
        | OptionalityClassReference
        | TagClassReference
        | EncodingProcedureClassReference
BitfieldClassReference ::=
          #NUL
        | #BOOL
        | #INT
```

```
                | #BITS
                | #OCTETS
                | #CHARS
                | #PAD
                | #BIT-STRING
                | #BOOLEAN
                | #CHARACTER-STRING
                | #EMBEDDED-PDV
                | #ENUMERATED
                | #EXTERNAL
                | #INTEGER
                | #NULL
                | #OBJECT-IDENTIFIER
                | #OCTET-STRING
                | #OPEN-TYPE
                | #REAL
                | #RELATIVE-OID
                | #TIME
                | #DATE
                | #DATE-TIME
                | #TIME-OF-DAY
                | #DURATION
                | #GeneralizedTime
                | #UTCTime
                | #ObjectDescriptor
                | #BMPString
                | #GeneralString
                | #GraphicString
                | #IA5String
                | #NumericString
                | #PrintableString
                | #TeletexString
                | #UniversalString
                | #UTF8String
                | #VideotexString
                | #VisibleString
    AlternativesClassReference ::=
                  #ALTERNATIVES
                | #CHOICE
    ConcatenationClassReference ::=
                  #CONCATENATION
                | #SEQUENCE
                | #SET
    RepetitionClassReference ::=
                  #REPETITION
                | #SEQUENCE-OF
                | #SET-OF
    OptionalityClassReference ::=
                  #OPTIONAL
    TagClassReference ::=
                  #TAG
    EncodingProcedureClassReference ::=
                  #TRANSFORM
                | #CONDITIONAL-INT
                | #CONDITIONAL-REPETITION
                | #OUTER
    EncodingStructureField ::=
                  #NUL
                | #BOOL
                | #INT Bounds?
                | #BITS Size?
```

```
            | #OCTETS Size?
            | #CHARS Size?
            | #PAD
            | #BIT-STRING Size?
            | #BOOLEAN
            | #CHARACTER-STRING
            | #EMBEDDED-PDV
            | #ENUMERATED Bounds?
            | #EXTERNAL
            | #INTEGER Bounds?
            | #NULL
            | #OBJECT-IDENTIFIER
            | #OCTET-STRING Size?
            | #OPEN-TYPE
            | #REAL
            | #RELATIVE-OID
            | #TIME
            | #DATE
            | #DATE-TIME
            | #TIME-OF-DAY
            | #DURATION
            | #GeneralizedTime
            | #UTCTime
            | #ObjectDescriptor Size?
            | #BMPString Size?
            | #GeneralString Size?
            | #GraphicString Size?
            | #IA5String Size?
            | #NumericString Size?
            | #PrintableString Size?
            | #TeletexString Size?
            | #UniversalString Size?
            | #UTF8String Size?
            | #VideotexString Size?
            | #VisibleString Size?
Bounds ::= "(" EffectiveRange ")"
EffectiveRange ::=
              MinMax
            | Fixed
Size ::= "(" SIZE SizeEffectiveRange ")"
SizeEffectiveRange ::=
            "(" EffectiveRange ")"
MinMax ::=
            ValueOrMin
            ".."
            ValueOrMax
ValueOrMin ::=
            SignedNumber
          | MIN
ValueOrMax ::=
              SignedNumber
            | MAX
Fixed ::= SignedNumber
ValueMapping ::=
              MappingByExplicitValues
            | MappingByMatchingFields
            | MappingByTransformEncodingObjects
            | MappingByAbstractValueOrdering
            | MappingByValueDistribution
            | MappingIntToBits
MappingByExplicitValues ::=
```

```
                VALUES
                "{"
                MappedValues "," +
                "}"
    MappedValues ::=
                MappedValue1
                TO
                MappedValue2
    MappedValue1 ::= Value
    MappedValue2 ::= Value
    MappingByMatchingFields ::=
                FIELDS
    MappingByTransformEncodingObjects ::=
                TRANSFORMS
                "{"
                OrderedTransformList
                "}"
    OrderedTransformList ::= Transform "," +
    Transform ::= EncodingObject
    MappingByAbstractValueOrdering ::=
                ORDERED VALUES
    MappingByValueDistribution ::=
                DISTRIBUTION
                "{"
                Distribution "," +
                "}"
    Distribution ::=
                SelectedValues
                TO
                identifier
    SelectedValues ::=
                  SelectedValue
                | DistributionRange
                | REMAINDER
    DistributionRange ::=
                DistributionRangeValue1
                ".."
                DistributionRangeValue2
    SelectedValue ::= SignedNumber
    DistributionRangeValue1 ::= SignedNumber
    DistributionRangeValue2 ::= SignedNumber
    MappingIntToBits ::=
                TO BITS
                "{"
                MappedIntToBits "," +
                "}"
    MappedIntToBits ::=
                  SingleIntValMap
                | IntValRangeMap
    SingleIntValMap ::=
                IntValue
                TO
                BitValue
    IntValue ::= SignedNumber
    BitValue ::=
                bstring |
                hstring
    IntValRangeMap ::=
                IntRange
                TO
                BitRange
```

```
IntRange ::=
            IntRangeValue1
            ".."
            IntRangeValue2
BitRange ::=
            BitRangeValue1
            ".."
            BitRangeValue2
IntRangeValue1 ::= SignedNumber
IntRangeValue2 ::= SignedNumber
BitRangeValue1 ::=
            bstring |
            hstring
BitRangeValue2 ::=
            bstring |
            hstring
```

G.2.2 Следующие продукции определяются в ИСО/МЭК 8824-1 с учетом изменений по приложению А и с элементами, определенными в G.1 в качестве терминальных символов:

Примечание — Вычеркнутые продукции не разрешены в ECN.

```
ModuleIdentifier ::=
    modulereference
    DefinitiveIdentifier ?
DefinitiveIdentifier ::=
            "{" DefinitiveObjIdComponentList "}"
DefinitiveObjIdComponentList ::=
     DefinitiveObjIdComponent
   | DefinitiveObjIdComponent DefinitiveObjIdComponentList
DefinitiveObjIdComponent ::=
     NameForm
   | DefinitiveNumberForm
   | DefinitiveNameAndNumberForm
NameForm ::= identifier
DefinitiveNumberForm ::= number
DefinitiveNameAndNumberForm ::= identifier "(" DefinitiveNumberForm ")"
Exports ::=
EXPORTS SymbolsExported? ";" |
EXPORTS ALL ";"
SymbolsExported ::= SymbolList
Imports ::= IMPORTS SymbolsImported? ";"
SymbolsImported ::= SymbolsFromModuleList
SymbolsFromModuleList ::=
     SymbolsFromModule |
     SymbolsFromModuleList SymbolsFromModule
SymbolsFromModule ::=
   SymbolList
            FROM
            GlobalModuleReference
GlobalModuleReference ::=
     modulereference AssignedIdentifier
AssignedIdentifier ::=
          DefinitiveIdentifier
        | empty
SymbolList ::=
     Symbol |
     SymbolList "," Symbol
Symbol ::=
          Reference
        | ParameterizedReference
        | BuiltinEncodingClassReference
```

```
Reference ::=
        encodingclassreference
      | ExternalEncodingClassReference
      | encodingobjectreference
      | encodingobjectsetreference
Value ::=
        BuiltinValue
      | ReferencedValue
      | ObjectClassFieldValue
BuiltinValue ::=
        BitStringValue
      | BooleanValue
      | CharacterStringValue
      | ChoiceValue
      | EmbeddedPDVValue
      | EnumeratedValue
      | ExternalValue
      | InstanceOfValue
      | IntegerValue
      | NullValue
      | ObjectIdentifierValue
      | OctetStringValue
      | RealValue
      | RelativeOIDValue
      | SequenceValue
      | SequenceOfValue
      | SetValue
      | SetOfValue
      | TaggedValue
BitStringValue ::=
        bstring
      | hstring
      | "{" IdentifierList "}"
      | "{" "}"
BooleanValue ::=
        TRUE
      | FALSE
CharacterStringValue ::=
        RestrictedCharacterStringValue
      | UnrestrictedCharacterStringValue
RestrictedCharacterStringValue ::=
          cstring
        | CharacterStringList
        | Quadruple
        | Tuple
CharacterStringList ::= "{" CharSyms "}"
CharSyms ::=
          CharsDefn
        | CharSyms "," CharsDefn
CharsDefn ::=
          cstring
        | Quadruple
        | Tuple
        | AbsoluteCharReference
Quadruple ::= "{" Group "," Plane "," Row "," Cell "}"
Group ::= number
Plane ::= number
Row ::= number
Cell ::= number
Tuple ::= "{" TableColumn "," TableRow "}"
TableColumn ::= number
```

```
TableRow ::= number
AbsoluteCharReference ::=
            ModuleIdentifier
            "."
            valuereference
~~UnrestrictedCharacterStringValue ::= SequenceValue~~
ChoiceValue ::= identifier ":" Value
~~EmbeddedPDVValue ::= SequenceValue~~
EnumeratedValue ::= identifier
~~ExternalValue ::= SequenceValue~~
IntegerValue ::=
              SignedNumber
            ~~| identifier~~
SignedNumber ::=
            number |
           "-" number
NullValue ::= NULL
ObjectIdentifierValue ::=
"{" ObjIdComponentsList "}"
      ~~| "{" DefinedValue ObjIdComponentsList "}"~~
ObjIdComponentsList ::=
            ObjIdComponents |
            ObjIdComponents ObjIdComponentsList
ObjIdComponents ::=
            NameForm |
            NumberForm |
            NameAndNumberForm
NameForm ::= identifier
NumberForm ::=
             number
            ~~| DefinedValue~~
NameAndNumberForm ::= identifier "(" NumberForm ")"
OctetStringValue ::=
            bstring |
            hstring
RealValue ::=
             NumericRealValue
           | SpecialRealValue
NumericRealValue ::=
            ~~–0~~
            ~~| +~~realnumber
            | "-" realnumber
            | SequenceValue
SpecialRealValue ::=
             PLUS-INFINITY
           | MINUS-INFINITY
RelativeOIDValue ::= "{" RelativeOidComponentsList "}"
RelativeOidComponentsList ::=
      RelativeOidComponents
| RelativeOidComponents RelativeOidComponentsList
RelativeOidComponents ::=
NumberForm
| NameAndNumberForm
~~| DefinedValue~~
SequenceValue ::=
            "{" ComponentValueList "}" |
            "{" "}"
ComponentValueList ::=
              NamedValue
            | ComponentValueList "," NamedValue
NamedValue ::=
```

```
            identifier Value
SequenceOfValue ::=
            -- "{" ValueList "}"
            | "{" "}"
ValueList ::=
    -- Value
    | ValueList "," Value
SetValue ::=
            "{" ComponentValueList "}" |
            "{" "}"
SetOfValue ::=
            "{" ValueList "}" |
            "{" "}"
ValueSet ::= "{" ElementSetSpecs "}"
ElementSetSpecs ::=
              RootElementSetSpec
            | RootElementSetSpec "," "..."
            | "..." "," AdditionalElementSetSpec
            | RootElementSetSpec "," "..." "," AdditionalElementSetSpec
RootElementSetSpec ::= ElementSetSpec
ElementSetSpec ::=
            Unions
          | ALL Exclusions
Exclusions ::= EXCEPT Elements
Unions ::=
            Intersections
          | UElems UnionMark Intersections
UElems ::= Unions
Intersections ::=
            IntersectionElements
          | IElems IntersectionMark IntersectionElements
IntersectionElements ::= Elements | Elems Exclusions
UnionMark ::=
            "|" |
            UNION
Elements ::=
            SubtypeElements
          | ObjectSetElements
          | "(" ElementSetSpec ")"
SubtypeElements ::=
            SingleValue
          | ContainedSubtype
          | ValueRange
          | PermittedAlphabet
          | SizeConstraint
          | TypeConstraint
          | InnerTypeConstraints
SingleValue ::= Value
```

G.2.3 Следующие продукции определяются в ИСО/МЭК 8824-2 с учётом изменений по приложению B и с элементами, определенными в G.1 в качестве терминальных символов:

```
DefinedSyntax ::= "{" DefinedSyntaxList ? "}"
DefinedSyntaxList ::= DefinedSyntaxToken DefinedSyntaxList ?
DefinedSyntaxToken ::=
    Literal
  | Setting
Literal ::=
    word
    | ","
Setting ::=
```

```
        Value
      | ValueSet
      | OrderedValueList
      | EncodingObject
      | EncodingObjectSet
      | OrderedEncodingObjectList
      | DefinedOrBuiltinEncodingClass
      | OUTER
OrderedValueList ::= "{" Value "," + "}"
OrderedEncodingObjectList ::= "{" EncodingObject "," + "}"
InstanceOfValue ::= Value
EncodingClassFieldType ::=
      DefinedEncodingClass
      "."
      FieldName
FieldName ::= PrimitiveFieldName "." +
PrimitiveFieldName ::=
       valuefieldreference
      | valuesetfieldreference
| orderedvaluelistfieldreference
```

G.2.4 Следующие продукции определяются в ИСО/МЭК 8824-4 с учетом изменений по приложению С и с элементами, определенными в G.1 в качестве терминальных символов:

```
ParameterizedAssignment ::=
  ParameterizedEncodingObjectAssignment
| ParameterizedEncodingClassAssignment
| ParameterizedEncodingObjectSetAssignment
ParameterizedEncodingObjectAssignment ::=
    encodingobjectreference
    ParameterList
      DefinedOrBuiltinEncodingClass
      "::="
      EncodingObject
ParameterizedEncodingClassAssignment ::=
      encodingclassreference
      ParameterList
      "::="
      EncodingClass
ParameterizedEncodingObjectSetAssignment ::=
      encodingobjectsetreference
      ParameterList
      #ENCODINGS
      "::="
      EncodingObjectSet
ParameterList ::= "{<" Parameter "," + ">}"
Parameter ::=
             ParamGovernor ":" DummyReference
           | DummyReference
ParamGovernor ::=
             Governor
           | DummyGovernor
Governor ::=
       EncodingClassFieldType
      | REFERENCE
      | DefinedOrBuiltinEncodingClass
      | #ENCODINGS
      | Type
DummyGovernor ::= DummyReference
DummyReference ::=
         encodingclassreference
```

```
            | valuereference
            | typereference
            | identifier
            | encodingobjectreference
            | encodingobjectsetreference
ParameterizedReference ::=
                    Reference
                  | Reference "{<" ">}"
ParameterizedEncodingObject ::=
                SimpleDefinedEncodingObject
                ActualParameterList
SimpleDefinedEncodingObject ::=
        ExternalEncodingObjectReference
      | encodingobjectreference
ParameterizedEncodingObjectSet ::=
      SimpleDefinedEncodingObjectSet
      ActualParameterList
SimpleDefinedEncodingObjectSet ::=
        ExternalEncodingObjectSetReference
      | encodingobjectsetreference
ParameterizedEncodingClass ::=
      SimpleDefinedEncodingClass
      ActualParameterList
SimpleDefinedEncodingClass ::=
        ExternalEncodingClassReference
      | encodingclassreference
ActualParameterList ::= "{<" ActualParameter "," + ">}"
ActualParameter ::=
            Value
          | ValueSet
          | OrderedValueList
          | DefinedOrBuiltinEncodingClass
          | EncodingObject
          | EncodingObjectSet
          | OrderedEncodingObjectList
          | identifier
          | STRUCTURE
          | OU
```

Приложение ДА
(справочное)

Сведения о соответствии ссылочных международных стандартов национальным стандартам

Таблица ДА.1

Обозначение ссылочного международного стандарта	Степень соответствия	Обозначение и наименование соответствующего национального стандарта
ISO/IEC 9834-1:12005	IDT	ГОСТ Р ИСО/МЭК 9834-1—2009 «Информационная технология. Взаимосвязь открытых систем. Процедуры действий уполномоченных по регистрации ВОС. Часть 1. Общие процедуры и верхние дуги дерева идентификатора объекта АСН.1»
ISO/IEC 8824-1:2008	—	*
ISO/IEC 8824-2:2008	—	*
ISO/IEC 8824-3:2008	—	*
ISO/IEC 8824-4:2008	—	*
ISO/IEC 8825-1:2008	—	*
ISO/IEC 8825-2:2008	—	*
ISO/IEC 10646:2003	—	*

* Соответствующий национальный стандарт отсутствует.

Примечание — В настоящей таблице использовано следующее условное обозначение степени соответствия стандарта:

- IDT — идентичный стандарт.

УДК 681.324:006.354 ОКС 35.100.60

Ключевые слова: информационные технологии, правила кодирования, абстрактно-синтаксическая нотация, спецификация нотации контроля кодирования, кодирование, декодирование

Редактор *К.В. Колесникова*
Корректор *Е.Р. Ароян*
Компьютерная верстка *Ю.В. Половой*

Сдано в набор 11.11.2016. Подписано в печать 15.12.2016. Формат 60 × 84 $^{1}/_{8}$. Гарнитура Ариал.
Усл. печ. л. 25,58. Уч.-изд. л. 23.27. Тираж 25 экз. Зак. 3331.
Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

Набрано в ИД «Юриспруденция», 115419, Москва, ул. Орджоникидзе, 11
www.jurisizdat.ru y-book@mail.ru

Издано и отпечатано во ФГУП «СТАНДАРТИНФОРМ», 123995, Москва, Гранатный пер., 4.
www.gostinfo.ru info@gostinfo.ru