ICS 35.100.60
L 79

中华人民共和国国家标准

GB/T 16262.1—2006/ISO/IEC 8824-1:2002
代替 GB/T 16262—1996

信息技术　抽象语法记法一(ASN.1)
第1部分:基本记法规范

Information technology—Abstract syntax notation one (ASN.1)—Part 1:Specification of basic notation

(ISO/IEC 8824-1:2002,IDT)

2006-03-14 发布　　2006-07-01 实施

中华人民共和国国家质量监督检验检疫总局
中国国家标准化管理委员会　发布

目　次

前言 …… Ⅲ
引言 …… Ⅳ
1 范围 …… 1
2 规范性引用文件 …… 1
3 术语和定义 …… 2
4 缩略语 …… 10
5 记法 …… 10
6 类型扩展的 ASN.1 模块 …… 13
7 编码规则的可扩展性要求 …… 13
8 标记 …… 14
9 ASN.1 记法的使用 …… 15
10 ASN.1 字符集 …… 15
11 ASN.1 词项 …… 16
12 模块定义 …… 24
13 引用类型和值定义 …… 28
14 支持引用 ASN.1 成分的记法 …… 29
15 类型和值的赋值 …… 30
16 类型和值的定义 …… 32
17 布尔类型记法 …… 35
18 整数类型的记法 …… 35
19 枚举类型的记法 …… 36
20 实数类型的记法 …… 38
21 位串类型的记法 …… 39
22 八位位组串类型的记法 …… 40
23 空类型记法 …… 41
24 序列类型的记法 …… 41
25 单一序列类型的记法 …… 45
26 集合类型的记法 …… 47
27 单一集合类型的记法 …… 48
28 选择类型的记法 …… 49
29 精选类型的记法 …… 51
30 已标记类型的记法 …… 52
31 客体标识符类型的记法 …… 53
32 相对客体标识符类型记法 …… 54
33 嵌入式 pdv 类型的记法 …… 56
34 外部类型的记法 …… 57
35 字符串类型 …… 58
36 字符串类型的记法 …… 59

37 受限制字符串类型的定义 …… 59
38 GB/T 13000.1 中定义的命名字符和集 …… 63
39 字符的正则顺序 …… 66
40 无限制字符串类型的定义 …… 67
41 第 42 至 44 章中定义的类型的记法 …… 69
42 通用时间 …… 69
43 世界时间 …… 69
44 客体描述符类型 …… 70
45 受约束类型 …… 70
46 元素集规范 …… 72
47 子类型元素 …… 74
48 扩展标志 …… 78
49 例外标识符 …… 80
附录 A(规范性附录) ASN.1 常规表达式 …… 81
附录 B(规范性附录) 类型和值兼容的规则 …… 84
附录 C(规范性附录) 指派的客体标识符值 …… 93
附录 D(资料性附录) 给客体标识符成分赋值 …… 95
附录 E(资料性附录) 举例和提示 …… 97
附录 F(资料性附录) ASN.1 字符串的辅导附录 …… 120
附录 G(资料性附录) 类型扩展 ASN.1 的辅助附录 …… 123
附录 H(资料性附录) ASN.1 记法总结 …… 129

前　言

GB/T 16262 在《信息技术　抽象语法记法一(ASN.1)》总标题下，目前包括以下 4 个部分：

第 1 部分(即 GB/T 16262.1)：基本记法规范；

第 2 部分(即 GB/T 16262.2)：信息客体规范；

第 3 部分(即 GB/T 16262.3)：约束规范；

第 4 部分(即 GB/T 16262.4)：ASN.1 规范的参数化。

本部分为 GB/T 16262 的第 1 部分，等同采用国际标准 ISO/IEC 8824-1:2002《信息技术　抽象语法记法一(ASN.1)：基本记法规范》(英文版)。与该项国际标准的等同文本是 ITU-T 建议 X.680。

按照 GB/T 1.1—2000 的规定，本部分对 ISO/IEC 8824-1:2002 作了下列编辑性修改：

——“本标准”一词改为“本部分”；

——在引用的标准中，凡已转化成我国标准的各项标准，均用我国的相应标准编号代替。对“规范性引用文件”一章中的标准，按 GB/T 1.1 的规定重新进行了排序。

本部分代替 GB/T 16262—1996《信息处理系统　开放系统互连　抽象语法记法一(ASN.1)规范》。与 GB/T 16262—1996 相比，本次修订在内容上作了如下变化：

——将“0　引言”变为独立的“引言”；

——在“规范性引用文件”一章中增加了所涉及到的有关标准；

——在“术语和定义”一章中增加了所涉及到的有关术语及其定义，并对个别术语进行了修改；

——第 5 章“本标准中使用的记法”改为第 5 章“记法”和第 8 章“标记”，并对叙述内容作了适当修改；

——增加了“类型扩展的 ASN.1 模块”和“编码规则的可扩展性要求”两章；

——增加了“支持引用 ASN.1 成分的记法”、“相对客体标识符的记法“、“嵌入式 pdv 类型的记法”、“字符串类型”、“字符的正则顺序”、“无限制字符串类型的定义”、“受约束类型”、“元素集类型”、“扩展标志”和“例外标识符”的记法规范，删除了“任意类型的记法”；

——在 GB/T 16262—1996 中，将各种记法都译成了中文，在本修订版中，将记法按原文列出；

——增加了“ASN.1 常规表达式”、“类型和值兼容的规则”、“ASN.1 字符串的辅助附录”和“类型扩展 ASN.1 的辅助附录”；

——对部分条款的叙述作了适当修改。

本部分的附录 A、附录 B 和附录 C 是规范性附录，附录 D、附录 E、附录 F、附录 G 和附录 H 是资料性附录。

本部分由中华人民共和国信息产业部提出。

本部分由中国电子技术标准化研究所归口。

本部分起草单位：中国电子技术标准化研究所。

本部分主要起草人：郑洪仁、徐云驰、安金海。

引　言

GB/T 16262 的本部分为定义数据类型和值提出标准记法。*数据类型*(简称*类型*)是信息范畴(例如,数字、文本、静止图像或视频信息)。*数据值*(简称*值*)是这种类型的实例。本部分定义一些基本类型和它们对应的值,以及将它们组合成更复杂的类型和值的规则。

在某些协议结构中,每条消息规定为八位位组序列的二进制值。然而,标准的制定者需要定义十分复杂的数据类型来携带它们的消息,而不考虑它们的二进制表示法。为了规定这些数据类型,它们需要一个不必确定每个值表示法的记法。ASN.1 就是这样一种记法。该记法由一个或多个确定携带应用语义(称为*传送语法*)的八位位组值、称为*编码规则*的运算法则规范来补充。ISO/IEC 8825-1、ISO/IEC 8825-2 和 ISO/IEC 8825-4 规定标准化编码规则的三个族,它们分别称为基本编码规则(BER)、紧缩编码规则(PER)、XML 编码规则(XER)。

某些用户希望用 ASN.1 重新定义它们的遗留协议,但是由于他们需要保留它们已有的二进制表示法而不能使用标准化编码规则。其他用户希望更完整地控制线上各位的精确布局(传送语法)。为 ASN.1 规定编码控制记法(ECN)的 ISO/IEC 8825-3 可以解决这些要求。ECN 使设计者能用 ASN.1 形式上规定协议的抽象语法,但是,(如果他们也希望的话)通过写出补充 ENC 规范(可能引用编码某些部分的标准化编码规则)完全或部分控制线上的位。

在抽象层定义复杂类型的非常普遍的技术是通过定义简单类型的所有可能值定义少量的*简单类型*,然后以多种方式组合这些简单类型。定义新类型的一些方式如下:

a) 给出已有类型的(有序)列表,作为取自每个已有类型的值的(有序)序列能形成一个值;按本方式获得的所有可能值的集合是一个新类型(如果列表中的已有类型都不同,这一机制能扩展到允许省略取自列表中的某些值);

b) 给出(不同)已有类型的无序集,作为取自每个已有类型的值的(无序)序列能形成一个值;按本方式获得的值的所有可能无序集的集合是一个新类型(机制能再扩展到允许省略某些值);

c) 给出单个已有类型,作为取自每个已有类型的(有序)列表或零个、一个或多个值的(无序)集能形成一个值;按本方式获得的值的所有可能列表或集的集合是一个新类型;

d) 给出(不同)类型的列表,能从它们中的任一个选择一个值;按本方式获得的所有可能值的集是一个新类型;

e) 给出类型,作为它的子集,通过采用某些结构或值之间的顺序关系能形成一个新类型。

以这种方式组合类型的重要方面是编码规则应该认可组合结构,提供基本类型值集合的无歧义编码。因此,用本部分中规定的记法定义的每个基本类型在值的无歧义编码中被赋予一个作为帮助的标记。

标记主要为了给机器使用,而对本部分中定义的人记法并不必需。然而,必须要求某些类型不同时,就通过要求它们有不同的标记来表达。因此,分配标记是使用本记法的重要部分,但是,(自 1994 年以来)可以规定自动分配标记。

注:在本部分内,指派了标记值给所有的简单类型和构造机制。对使用记法的约束保证标记能用于传送中值的无歧义标识。

ASN.1 规范最初用完全定义的 ASN.1 类型的集产生。然而,在随后的阶段里,可能必须改变这些类型(通常通过在序列或集类型中附加额外成分)。如果下面的方式有可能:采用旧类型定义的实现能以定义的方式与采用新类型定义的实现互工作,那么,编码规则需要提供合适的支持。ASN.1 记法支持包括类型数上的*扩展标志*。这给编码规则发出设计者意图的信号:这个类型是称为扩展系列的系列

相关类型(也就是,相同初始类型的版本)之一,及要求编码规则能使信息在使用因是相同扩展系列一部分而相关的不同类型的实现之间传送。

第 10 至 31 章(含)定义 ASN.1 支持的简单类型,并规定用于引用简单类型和用它们定义新类型的记法。第 10 至 31 章也规定用于规定用 ASN.1 定义的类型值的记法。提供了两个值记法,第一个称为基本 ASN.1 值记法,并且自它引进以来就一直是 ASN.1 记法的一部分。第二个称为 XML ASN.1 值记法,并提供使用可扩展置标语言(XML)的值记法。

注:XML 值记法提供使用 XML 表示 ASN.1 值的方法。因此,ASN.1 类型定义也规定 XML 元素的结构和内容。这使 ASN.1 成为 XML 的简单模式语言。

第 33 至 34 章(含)定义 ASN.1 支持的类型以便在其内携带 ASN.1 类型的完整编码。

第 35 至 40 章(含)定义字符串类型。

第 41 值 44 章(含)定义认为是通用的、但没有要求附加编码规则的某些类型。

第 45 至 47 章(含)定义子类型能从双亲类型值定义的记法。

第 48 章定义允许"版本 1"规范中规定的 ASN.1 类型标识为可能在"版本 2"中扩展,而且对于后续版本中带来的附加分别列出并用它们的版本号标识的记法。

第 49 章定义允许 ASN.1 类型定义包含如果收到位于当前标准化定义中规定的值之外值的编码时预计错误处理的指示的记法。

附录 A 构成本部分的完整部分,并规定 ASN.1 的正常表达式。

附录 B 构成本部分的完整部分,并规定类型和值兼容性的规则。

附录 C 构成本部分的完整部分,并记录 ASN.1 系列标准中指派的客体标识符和客体描述符值。

附录 D 不构成本部分的完整部分,它描述客体标识符注册树的顶级弧。

附录 E 不构成本部分的完整部分,它提供使用 ASN.1 记法的示例和提示。

附录 F 不构成本部分的完整部分,它提供 ASN.1 字符串的辅导。

附录 G 不构成本部分的完整部分,它提供类型扩展 ASN.1 模块的辅导。

附录 H 不构成本部分的完整部分,它提供使用第 5 章记法的 ASN.1 汇总。

信息技术　抽象语法记法一(ASN.1)
第1部分:基本记法规范

1　范围

GB/T 16262的本部分提供一个称为抽象语法记法一(ASN.1)的标准记法,该记法用来定义数据类型、值及数据类型的约束。

本部分

——定义了一些简单的类型及其标记,也规定了引用这些类型和规定这些类型值的记法;

——定义了从多个基本类型构造新类型的机制,也规定了定义这些类型及为他们指派标记和规定这些类型值的记法;

——定义了ASN.1内使用的字符集(通过引用其他标准);

无论何时需要定义信息的抽象语法,都可应用ASN.1记法。

ASN.1记法供其他定义ASN.1类型编码规则的标准引用。

2　规范性引用文件

下列文件中的条款通过GB/T 16262的本部分的引用而成为本部分的条款。凡是注日期的引用文件,其随后所有的修改单(不包括勘误的内容)或修订版均不适用于本部分,然而,鼓励根据本部分达成协议的各方研究是否可使用这些文件的最新版本。凡是不注日期的引用文件,其最新版本适用于本部分。

GB/T 1988—1988　信息技术　信息交换用七位编码字符集(eqv ISO/IEC 646:1991)

GB/T 2311—2000　信息技术　字符代码结构与扩充技术(idt ISO/IEC 2022:1994)

GB/T 2659—2000　世界各国和地区名称代码(eqv ISO 3116-1:1997)

GB/T 7408—1994　数据元和交换格式　信息交换　日期和时间表示法(eqv ISO 8601:1988)

GB/T 13000.1—1993　信息技术　通用多八位编码字符集(UCS)　第一部分:体系结构与基本多文种平面(idt ISO/IEC 10646-1:1993)

GB/T 16262.2—2006　信息技术　抽象语法记法一(ASN.1)　第2部分:信息客体规范(ISO/IEC 8824-2:2002,IDT)

GB/T 16262.3—2006　信息技术　抽象语法记法一(ASN.1)　第3部分:约束规范(ISO/IEC 8824-3:2002,IDT)

GB/T 16262.4—2006　信息技术　抽象语法记法一(ASN.1)　第4部分:ASN.1规范的参数化(ISO/IEC 8824-4:2002,IDT)

GB/T 16263.1—2006　信息技术　ASN.1编码规则　第1部分:基本编码规则(BER)、正则编码规则(CER)和非典型编码规则(DER)规范(ISO/IEC 8825-1:2002,IDT)

GB/T 16263.2—2006　信息技术　ASN.1编码规则　第2部分:紧缩编码规则(PER)规范(ISO/IEC 8825-2:2002,IDT)

GB/T 17969.1—2000　信息技术　开放系统互连　OSI登记机构的操作规程　第1部分:一般规程序(eqv ISO/IEC 9834-1:1993)

GB/T 18793—2002　信息技术　可扩展置标语言(XML)1.0

SJ/Z 9090—1987　数据互换　组织标识用的结构(idt ISO 6523:1984)

ISO/IEC 8825-3:2002 信息技术 ASN.1 编码规则:ASN.1 的编码控制记法(ECN)

ISO/IEC 8825-4:2002 信息技术 ASN.1 编码规则:XML 编码规则规范(XER)

ISO/IEC 7350:1991 信息技术 ISO 10367 图形字符集的登记

ISO 与转义序列使用的编码字符集国际注册

ITU-T Rec. TF.460-5:1997 标准频率和时间信号发射

ITU-T 建议 T.101:1994 可视文本服务的国际互工作

CCITT 建议 T.100:1988 交互式可视文本的国际信息交换

Unicode 标准,版本 3.2.0:2002,Unicode 联盟(读物,MA,Addison-Wesley)

注:因为上面的参考文件提供控制字符的名称,因此包括这项文件。

3 术语和定义

下列术语和定义适用于 GB/T 16262 的本部分。

3.1 信息客体规范

本部分使用 GB/T 16262.2—2006 中定义的下列术语:

a) 信息客体 information object;

b) 信息客体类别 information object class;

c) 信息客体集 information object set;

d) 单一实例类型 instance-of type;

e) 客体类别字段类型 object class field type。

3.2 约束规范

本部分使用 GB/T 16262.3—2006 中定义的下列术语:

a) 成分关系约束 component relation constraint;

b) 表约束 table constraint。

3.3 ASN.1 规范的参数化

本部分使用 GB/T 16262.4—2006 中定义的下列术语:

c) 参数化类型 parameterized type;

d) 参数化值 parameterized value。

3.4 组织标识的结构

本部分使用 SJ/Z 9090—1987 中定义的下列术语:

a) 发布组织 issuing organization;

b) 组织代码 organization code;

c) 国际代码指定者 International Code Designator。

3.5 通用多八位编码字符集(UCS)

本部分使用 GB/T 13000.1—1993 中定义的下列术语:

a) 基本多文种平面 (BMP) Basic Multilingual Plane(BMP);

b) 字符元 cell;

c) 组合用字符 combining character;

d) 图形符号 graphic symbol;

e) 组 group;

f) 有限子集 limited subset;

g) 平面 plane;

h) 行 row;

i) 选择子集 selected subset。

3.6 附加定义

3.6.1

抽象字符 abstract character

用于组织、控制和表示文本数据的抽象值。

注：附录F提供术语抽象字符更完整的描述。

3.6.2

抽象值 abstract value

其定义仅基于用来携带某些语义的类型，而与其在任何编码中的表达方式无关的值。

注：抽象值的示例是整数类型、布尔类型、字符串类型或者整数和布尔的序列(或选择)类型等的值。

3.6.3

ASN.1字符集 ASN.1 character set

第10章中规定的用于ASN.1记法的字符集。

3.6.4

ASN.1规范 ASN.1 specification

一个或多个ASN.1模块的集合。

3.6.5

关联类型 associated type

仅用于定义类型的值及子类型记法的类型。

注：当必须清楚ASN.1中怎样定义类型与它怎样编码之间可能有重大差别时，在本部分中定义了关联类型。用户规范中不出现关联类型。

3.6.6

位串类型 bitstring type

其非典型值是零个、一个或多个二进制位的有序序列的简单类型。

注：当需要携带抽象值的嵌入编码时，不赞成使用没有内容约束(见GB/T 16262.3—2006的第11章)的位串(或八位位组串)类型。而使用嵌入pdv类型(见第33章)提供更灵活的机制，允许声明抽象语法及嵌入的抽象值的编码。

3.6.7

布尔类型 boolean type

具有两个非典型值的简单类型。

3.6.8

字符性质 character property

与定义字符汇的表中的字符元相关的信息集。

注：信息通常将包含部分或全部下面的项：

a) 图形符号；

b) 字符名称；

c) 在特定环境下使用时，与字符相关的功能的定义；

d) 它是否表示数字；

e) 只有在(大/小)写中不同的关联字符。

3.6.9

字符抽象语法 character abstract syntax

其值规定为零个、一个或多个字符的字符串集的任何抽象语法，这些字符取自字符的某些已规定的集合。

3.6.10

字符汇 character repertoire

对字符怎样编码没有任何隐含说明的字符集中的那些字符。

3.6.11

字符串类型 character string types

其值取自某些已定义字符集的字符串的简单类型。

3.6.12

字符传送语法 character transfer syntax

字符抽象语法的任何传送语法。

注：ASN.1 不支持未将所有字符串编码成 8 位整数倍的字符传送语法。

3.6.13

选择类型 choice types

通过引用不同类型列表来定义的类型；选择类型的每个值从其中一个成分类型的值衍生而来。

3.6.14

成分类型 component type

定义 CHOICE、SET、SEQUENCE、SET OF 或 SEQUENCE OF 时引用的某种类型。

3.6.15

约束 constraint

可与类型一起使用来定义该类型的子类型的记法。

3.6.16

内容约束 contents constraint

规定内容为规定的 ASN.1 类型的编码、或规定的程序用来产生和处理内容的位串或八位位组串类型的约束。

3.6.17

控制字符 control character

出现在某些给出名称(和也许是有关某些环境的已定义功能)，但没有分配图形符号，也不是间隔字符的字符汇中的字符。

注：HORIZONTAL TABULATION(9)和 LINE FEED(10)是打印环境里指派了格式化功能的控制字符示例。DATA LINK ESCAPE(16)是通信环境中指派了功能的控制字符示例。

3.6.18

国际协调时 Coordinated Universal Time(UTC)

国际时间局所保持的时标，构成标准频率和时间信号协调传播的基础。

注 1：此定义的来源是 ITU-R Rec. TF.460-5。ITU-R 也用 UTC 作为国际协调时的缩写。

注 2：UTC 和格林尼治标准时间(GMT)是两个可替换的，在大部分实用情况下确定同一时间的时间标准。

3.6.19

元素 element

支配类型的值或支配信息客体类别的信息客体，能分别从相同类型或相同类别信息客体的所有其他值中区别开来。

3.6.20

元素集 element set

所有支配类型的值或支配类别的信息客体的元素集合。

注：GB/T 16262.2 的 3.4.7 中定义了支配类别。

3.6.21

嵌入 pdv 类型 embedded-pdv type

其值集是所有可能的抽象语法中值集形式上合并的类型。这一类型可用于希望其协议中携带其类型可能在那个 ASN.1 规范外部定义的抽象值的 ASN.1 规范中。它也携带被携带抽象值的抽象语法

(类型)的标识以及用来编码那个抽象值的编码规则的标识。

3.6.22

编码 encoding

编码规则集应用于抽象值上产生的位图。

3.6.23

(ASN.1)编码规则 (ASN.1) encoding rules

在传送 ASN.1 类型值期间规定其表示的规则。编码规则也能使值从给出类型知识的表示法中恢复。

注:为了规定编码规则,能为固有类型(和值)提供替换记法的各种被引用类型(和值)记法没有关系。

3.6.24

枚举类型 enumerated types

一个简单类型,其值是类型记法一部分的给定不同标识符。

3.6.25

扩展附加 extension addition

扩展序列中增加的记法之一。对于集合、序列和选择类型,每个扩展附加是单个扩展附加组或单个成分类型的附加。对于枚举类型,它是单个进一步枚举的附加。对于约束,它是(只有)一个子类型元素的附加。

注:扩展附加既按文本排列(紧接扩展标志),也按逻辑排列(有递增枚举值和在 CHOICE 替换时递增标记)。

3.6.26

扩展附加组 extension addition group

在版本括号内分组的集合、序列或选择类型的一或多个成分。扩展附加组用来清楚地标识加在 ASN.1 模块特定版本中集合、序列或选择类型的成分,也能用简单整数标识那个版本。

3.6.27

扩展附加类型 extension addition type

包含于扩展附加组中的类型或本身是扩展附加(这时它不包含在扩展附加组中)的单个成分类型。

3.6.28

可扩展约束 extensible constraint

在较外层带扩展标志,或使用带值的可扩展集的集算法中可扩展的子类型约束。

3.6.29

扩展插入点(或插入点) entension insertion point (or insertion point)

类型定义中插入扩展附加的位置。如果类型定义中有单个省略号,该位置在扩展序列中紧贴前面类型的类型记法末尾,或者如果类型定义中有扩展标志对,该位置紧贴在第二个省略号之前。

注:在任何选择、序列或集合类型成分内最多能有一个插入点。

3.6.30

扩展标志 extension marker

包含在形成扩展序列部分的所有类型中的语法标志(省略号)。

3.6.31

扩展标志对 extension marker pair

插入扩展附加之间的一对扩展标志。

3.6.32

相关扩展 extension-related

有相同扩展根,通过在一个上加零个或更多个扩展附加而产生另一个的两个类型。

3.6.33

扩展根 extension root

扩展序列中是第一个类型的可扩展类型。它携带不含附加记法、只有注解和扩展标志与相配的"}"

或“)”之间空白的扩展标志;或者不含附加记法,只有单个逗号、注解和扩展标志之间空白的扩展标志对。

注:只有扩展根能是扩展序列中的第一个类型。

3.6.34

扩展序列 extension series

能按通过在扩展插入点附加文本形成序列中每个连续类型的方式排列的 ASN.1 类型序列。

3.6.35

可扩展类型 extensible type

有扩展标志或者应用了扩展约束的类型。

3.6.36

外部引用 external reference

类型引用、值引用、信息客体类别引用、信息客体引用或(可能参数化的)信息客体集引用,他们在某些其他而不是正在被它引用的模块中定义,并且通过在被引用项上加模块名作为前缀来引用。

例如:ModuleName.TypeReference

3.6.37

外部类型 external type

携带对该 ASN.1 规范而言其类型可能是外部定义值的 ASN.1 规范一部分的类型。他也携带被携带值的类型标识。

3.6.38

假 false

布尔类型中的非典型值的一个(也见“真”)。

3.6.39

支配(类型);支配者 governing (type);governor

影响部分 ASN.1 语法解释、要求那部分 ASN.1 语法引用支配类型中的值的类型定义或引用。

3.6.40

等同类型定义 identical type definition

完成附录 B 中规定的转换后,如果 ASN.1“Type”产生式中的两个实例(见第 16 章)是相同词项的相同顺序列表(见第 11 章),则它们定义为相同类型定义。

3.6.41

整数类型 integer type

具有非典型值的简单类型,值是正整数或负整数,包括零(作为单一值)。

注:当特定的编码规则限制整数的范围时,选择这种限制不至影响 ASN.1 任何用户。

3.6.42

词项 lexical item

取自 ASN.1 字符集(在第 11 章中规定),用来形成 ASN.1 记法的字符的已命名序列。

3.6.43

模块 module

类型、值、值集、信息客体类别、信息客体和信息客体集(以及它们的参数化变体)采用 ASN.1 记法的一个或多个实例,用 ASN.1 模块记法来定界(见第 12 章)。

注:术语信息客体类别(等)在 GB/T 16262.2—2006 中规定,而参数化在 GB/T 16262.4—2006 中规定。

3.6.44

空类型 null type

由单一值组成的简单类型,也称为空。

3.6.45

客体 object

精确定义的一段信息、定义或规范，它要有名称以便标识其在通信实例中的用途。

注：这种客体可能像 GB/T 16262.2—2006 中定义的信息客体。

3.6.46

客体描述符类型 object descriptor type

其非典型值是提供对信息客体简要描述的人可读的文本的类型。

注：客体描述符值常常与单个客体相关，只有客体标识符值无歧义地标识一个客体。

3.6.47

客体标识符 object identifier

与无歧义地标识它的客体相关的全局唯一值。

3.6.48

客体标识符类型 object identifier type

其值是按标准 GB/T 17969 系列的规则分配的所有客体标识符的集合的简单类型。

注：GB/T 17969.1 的规则允许各种机构独立地将客体标识符与信息客体相联系。

3.6.49

八位位组串类型 octetstring type

其非典型值是零个、一个或多个八位位组的有序序列的简单类型。每个八位位组是八个二进制位的有序序列。

3.6.50

开放系统互连 open systems interconnection

提供许多以缩写“OSI”开始、用于本部分的术语的计算机通信结构。

注：如果需要，这些术语的意义能从 ITU-T Rec. X.200 系列和相当的 ISO/IEC 标准中获得。如果 ASN.1 用于 OSI 环境，这些术语是唯一适用的。

3.6.51

开放类型记法 open type notation

用来表示取自不只一个 ASN.1 类型的值的集合的 ASN.1 记法。

注 1：在本部分的正文中，术语“开放类型”和“开放类型记法”同义使用。

注 2：所有 ASN.1 编码规则为单个 ASN.1 类型值提供无歧义编码，他们不必为“开放类型记法”提供无歧义的编码，因为“开放类型记法”携带取自在规范时刻通常没有确定的 ASN.1 类型的值。能无歧义确定那一字段的抽象值前需要“开放类型记法”中被编码值的类型的知识。

注 3：本部分中是“开放类型记法”的唯一记法是 GB/T 16262.2—2006 第 14 章中规定的“ObjectClassFieldType”。这里的“FieldName”指明类型字段或者可变类型值字段。

3.6.52

(子类型的)双亲类型 parent type (of a subtype)

定义子类型时受约束的、并支配子类型记法的类型。

注：双亲类型本身可能是某些其他类型的子类型。

3.6.53

产生式 production

用于规定 ASN.1 的形式记法的一部分(也叫做语法规则或 Backus-Naur Form，BNF)。

3.6.54

实数类型 real type

一个简单类型，其非典型值(第 20 章中规定)是实数集合的一个成员。

3.6.55

(类型的)递归定义 recursive definition (of a type)

ASN.1 的定义的一个集合,不能对这些定义重新排序,因此,结构中使用的所有类型在定义构造之前定义。

注:ASN.1 中允许递归定义:记法的用户有责任保证所用(产生类型的)这些值有限定的表示法并且与类型相关的值集至少包含一个值。

3.6.56

相对客体标识符 relative object identifier

通过其相对某些已知客体标识符(见 3.6.47)的位置标识客体的值。

3.6.57

相对客体标识符类型 relative object identifier type

其值是所有可能相对客体标识符集的简单类型。

3.6.58

受限制字符串类型 restricted character string type

其字符取自类型规范中标识的固定字符汇的字符串类型。

3.6.59

精选类型 selection types

通过引用选择类型的成分类型定义的类型,而且其值精确地为那个成分类型的值。

3.6.60

序列类型 sequence types

通过引用固定的、有序的类型列表(有些可能声明是可选的)定义的类型;序列类型的每个值是取自每个成分类型的值的有序列表。

注:当一个成分类型声明为可选时,序列类型的值不必包含那个成分类型的值。

3.6.61

单一序列类型 sequence-of types

通过引用单个成分类型定义的类型;单一序列类型中的每个值是成分类型的零个、一个或多个值的有序列表。

3.6.62

(约束的)连续应用 serial application (of constraints)

约束应用在已经受约束的双亲类型。

3.6.63

集合运算 set arithmetic

使用如 46.2 中规定的并集、交集和差集(使用 EXCEPT)等运算的值或信息客体的新集的形式。

注:术语"值算法"没有覆盖约束连续应用的结果。

3.6.64

集合类型 set types

通过引用固定的、无序的、类型(有些声明是可选的)列表定义的类型。集合类型中的每个值是一个无序的值列表,列表中的各个值取自相应的成分类型。

注:当成分类型声明为可选时,集合类型的值不必包含那个成分类型的值。

3.6.65

单一集合类型 set-of types

通过引用单个成分类型定义的类型,单一集合类型中的每个值是成分类型的零个、一个或多个值的无序列表。

3.6.66

简单类型 simple types

通过直接规定其值集来定义的类型。

3.6.67

间隔字符 spacing character

字符汇中的字符，它用来包括字符串打印中的图形字符，但用空间隔在物理解释中表示。通常，不认为它是控制字符(见 3.6.17)。

注：字符汇中可能有单个间隔字符，或宽度可变的多个间隔字符。

3.6.68

(双亲类型的)子类型 subtype (of a parent type)

其值是某些其他类型(双亲类型)值的子集(或完整集)的类型。

3.6.69

标记 tag

与每个 ASN.1 类型相关的类型记号。

3.6.70

已标记类型 tagged type

通过引用单个现存类型和标记来定义的类型，新类型与该现存类型同构，但与它不等同。

3.6.71

置标记 tagging

用规定的标记替换某个类型现有(可能是默认)的标记。

3.6.72

传送语法 transfer syntax

用来交换抽象语法中抽象值的位串的集合，通常是将编码规则应用在抽象语法上获得。

注：术语“传送语法”与“编码”同义。

3.6.73

真 true

布尔类型中的非典型值的一个(也见“假”)。

3.6.74

类型 type

已命名的值集合。

3.6.75

类型引用名 type reference name

在某些上下文中唯一与类型相联系的名称。

注：指派引用名给本部分中定义的类型，在 ASN.1 中是普遍存在的。其他引用名在其他标准中定义，并只适用于那个标准的上下文中。

3.6.76

未限制字符串类型 unrestricted character string type

其抽象值取自字符抽象语法、并且带字符抽象语法及用于其编码的字符传送语法的标识的值的类型。

3.6.77

(ASN.1)用户 user (of ASN.1)

用 ASN.1 定义一段特定信息的抽象语法的个人或组织。

3.6.78

值映射 value mapping

能使引用那些值的一个用来引用其他值的两个类型中的值之间的 1-1 对应关系。例如，这能用在

规定子类型和默认值中(见附录 B)。

3.6.79

值引用名 value reference name

在某些上下文中唯一与值相联系的名称。

3.6.80

值集(合) value set

类型的值的集合,语义上相当于子类型。

3.6.81

版本括号 version brackets

一对用来描画扩展附加组开始和结束的相邻左括号和右括号([[或]])。紧接在左括号对后面可以有选择地给出扩展附加组版本号的数字。

3.6.82

版本号 version number

能与版本括号相联系的数字(见 G.1.8)。

注:版本号不能加在不是扩展附加组一部分的扩展附加上,也不能加在非选择、序列或集合的任意类型的扩展附加上。

3.6.83

空白 white-space

任何在打印页上产生间隔的格式化动作,例如:空格或制表。

4 缩略语

本部分采用下列缩略语:

ASN.1	抽象语法记法一
BER	ASN.1 基本编码规则
BMP	基本多文种平面
DCC	数据国家代码
DNIC	数据网络标识代码
ECN	ASN.1 编码控制记法
ICD	国际代码指定者
IEC	国际电工委员会
ISO	国际标准化组织
ITU-T	国际电信联盟—电信标准化部
OID	客体标识符
OSI	开放系统互连
PER	ASN.1 的紧缩编码规则
ROA	公认的运营机构
UCS	通用多八位编码字符集
UTC	国际协调时
XML	可扩展置标语言

5 记法

5.1 概述

5.1.1 ASN.1 记法由取自第 10 章规定的 ASN.1 字符集的字符序列组成。

5.1.2 每次使用 ASN.1 记法包括从 ASN.1 字符集中抽取字符并组合为词项。第 11 章规定了组成词项的字符的所有序列，并命名了每个项。

5.1.3 在第 12 章(以及以下几章中)，ASN.1 记法的规定是通过对组成 ASN.1 记法有效实例的词项的那些序列的规定和命名，及对每个序列的 ASN.1 语义的规定来实现的。

5.1.4 为了规定词项的允许序列，本部分使用下面各条中定义的形式记法。

5.2 产生式

5.2.1 所有的词项都已命名(见第 11 章)，并且词项的允许序列也都已命名。

5.2.2 一个新的(更复杂的)词项的允许序列是通过产生式是来定义的。它使用词项的名称和词项的允许序列，并形成词项的新已命名允许序列。

5.2.3 每个产生式由下面几个部分组成，占一行或几行，次序是：

a) 词项新允许序列的名字；

b) 字符

```
::=
```

c) 一个或多个 5.3 中所定义的词项的替换序列，使用下面字符分隔

```
|
```

5.2.4 一个词项序列若在一个或多个替换项中出现，则它在词项的新允许序列中出现。在本部分中，词项的新允许序列用上面 5.2.3a)中的名字引用。

注：若词项同一序列出现在多个替换项中，产生的记法中任何语义上的歧义性由相关文本解决。

5.3 替换项集

5.3.1 产生式(见 5.2.3c)中的每个替换项由名称列表来规定。每一个名称或者是一个词项名，或者是一个由某些其他产生式定义和命名的词项的允许序列的名称。

5.3.2 每个替换项定义的词项允许序列由所有这样获得的序列组成，取任何一个与第一个名称相关的序列(或词项)，(然后)和任何一个与第二个名称相关的序列(或词项)组合，(然后)和任何一个与第三个名称相关的序列(或词项) 组合，等等，直到包括替换项中最后的一个名称(或词项)。

5.4 非间隔指示符

如果产生式序列的这些项之间插入非间隔指示符“&”(AMPERSAND)，那么它前面的词项和它后面的词项不应该用空白隔开。

注：这个指示符仅用于描述 XML 值记法的产生式中。例如，它用来规定词项“<”是紧接 XML 标记名的。

5.5 产生式的示例

5.5.1 产生式：

```
ExampleProduction ::=
        bstring                 |
        hstring                 |
        " {"IdentifierList " }"
```

使名称“ExampleProduction”与词项的下列序列相关：

a) 任何“bstring”(词项)；或

b) 任何“hstring”(词项)；或

c) 任何与“IdentifierList”相关的词项序列，用" {" 开始，用" }" 结束。

注：" {" 和" }" 是含有单个字符{和}的词项名称(见 11.26)。

5.5.2 本例中，“IdentifierList”由进一步的产生式定义，可以在定义“ExampleProduction”的产生式之前或之后。

5.6 样式

本部分中使用的每个产生式前面或后面都有一个空行。产生式中没有空行。产生式可以在一行上

或者分布在几行上。样式并不重要。

5.7 递归

本部分中的产生式通常是递归的。在这种情况下，只要有新的序列产生，产生式就要继续重复。

注：在很多情况下，这种反复应用导致词项的允许序列的无穷集，集合中的某些或者所有序列本身可能包含的词项数目不限定，这没有错。

5.8 词项允许序列的引用

本部分通过引用产生式中出现在 ::= 之前的名称来引用词项的允许序列(ASN.1 记法的一部分)；除非它就作为产生式的一部分出现，否则，这个名称用 QUOTATION MARK(34)字符(")括起来，以便把它和自然语言文本区分开来。

5.9 词项的引用

本部分通过使用词项的名称来引用词项，当名称在自然语言文本中出现，并且可能与该文本混淆时，那么，把它用 QUOTATION MARK (34)字符(")括起来。

5.10 缩写记法

为了使产生式更加准确和更可读，下面的缩写记法用于本部分的词项的允许序列定义中，且也用在 GB/T 16262.2，GB/T 16262.3，GB/T 16262.4 中。

a) "A"和"B"两个名称之后的星号(*)表示 "empty"词项(见 11.7)，或与"A"相关的词项的允许序列之一，或与 "A" 相关的词项序列之一和与"B"相关的词项序列之一的替换序列，两者都以与"A"相关一个开始和结束。因此：

C::=AB*

相当于：

C::=D|empty

D::=A|ABD

"D"是不会在产生式中其他地方出现的辅助名。

例如："C::=AB*"是 C 的下列替换项的缩写记法：

empty

A

ABA

ABABA

ABABABA

...

b) 加号(+)与 a)中的星号类似，只是不包括"empty"词项。因此：

E::=AB+

相当于：

E::=A|ABE

例如："E::=AB+"是 E 的下列替换项的缩写记法：

A

ABA

ABABA

ABABABA

...

c) 名称后的问号(?)表示"empty"词项(见 11.7)或与"A"相关的词项的允许序列，因此：

F::=A?

相当于：

F::=empty|A

注：这些缩写记法优先于产生式序列中的词项并置(见 5.2.2)。

5.11 值引用和值的类型

5.11.1 ASN.1 值赋值记法能把名称给与规定类型的值。只要需要引用那个值时就能采用这个名称。附录 B 描述和规定了允许用一个类型值的值引用名称来标识第二个(相似)类型的值的值映射机制。因此，只要要求引用第二个类型中的值就能使用引用第一个值。

5.11.2 在 ASN.1 标准正文中，用正常的英语文本来规定包含不止一个类型构造的合法性(或其他)。这些合法性规范通常要求两个或更多个类型"兼容"。例如，使用值引用时，要求用于定义值引用的类型与支配类型"兼容"。附录 B 采用值映射概念来为任意给出的 ASN.1 构造是否合法给出精确的描述。

6 类型扩展的 ASN.1 模块

当解码可扩展类型时，解码器可检测：

a) 序列或集合类型中缺少希望的扩展附加；或

b) 序列或集合类型(若有)中定义的那些扩展附加之上出现任意不希望的扩展附加，或者选择类型中出现未知的替换项，或者枚举类型中出现未知的枚举，或可扩展其约束的类型出现不希望长度或值。

在形式术语中，可扩展类型"X"定义的抽象语法包含的不仅是类型"X"的值，而且有与"X"扩展有关的所有类型的值。因此，无论发现上面 a)或 b)的哪种情形，解码过程从不会发出错误的信号。每种情形中采取的动作由 ASN.1 规定者确定。

注：通常，动作将忽略出现不希望的扩展附加，而且会为缺少的、希望扩展附加使用默认值或"丢失"指示器。

可扩展类型中解码器检测到的不希望扩展附加，后来能包含在那一类型的随后编码中(以转发回发送者，或某些第三方)，只要在随后的转发中使用相同的传送语法。

7 编码规则的可扩展性要求

注：这些要求适用于标准化的编码规则。它们不适用于用 ECN 定义的编码规则(见 ISO/IEC 8825-3)。

7.1 所有 ASN.1 编码规则应允许可扩展类型"X"的值以它们能用与"X"扩展有关的可扩展类型"Y"解码的方式编码。而且，编码规则应允许用"Y"解码的值(用"Y")重新编码和用与"Y"(而且由此也有"X")扩展有关的第三个可扩展类型"Z"解码。

注：类型"X"、"Y"、"Z"可能在扩展序列中以任何顺序出现。

如果可扩展类型"X"的值被编码，并且随后被传递(直接或通过用扩展有关的类型"Z"的传递应用)到另一个用与"X"扩展有关的可扩展类型"Y"解码的应用，然后使用类型"Y"的解码器获得由如下构成的抽象值：

a) 扩展根类型的抽象值；

b) 在"X"和"Y"中都出现的每个扩展附加的抽象值；

c) 只有"X"中有而"Y"中没有的每个扩展附加(如果有的话)的无限制编码。

如果应用层也这样要求，c)中的编码应该能包含在"Y"值后面的编码中。那个编码应该是"X"值的有效编码。

指导示例：如果系统 A 使用带是可选整数类型扩展附加的序列类型或集合类型的可扩展根类型(类型"X")，而系统 B 使用有每个是可选整数类型的两个扩展附加的、与扩展有关的类型(类型"Y")，那么，用 B 传输忽略第一个扩展附加的整数值而包括第二个的"Y"值决不能用出现它了解的 X 的第一个(只有)扩展附加与 A 混淆。而且，如果应用协议也这样要求的话，A 必须能重新编码带有作为第一个整数类型出现的值、随后跟有从 B 收到的第二个整数值的"X"值。

7.2 所有的 ASN.1 编码规则应该规定以如下方式编码和解码枚举类型和选择类型的值：如果转送的

值是在由编码器和解码器共同保持的扩展附加集中，那么，它被成功地解码，否则，解码器将可能限定它的编码并把它标识为(未知)扩展附加值。

7.3 所有的 ASN.1 编码规则应该规定以如下方式编码和解码有可扩展约束的类型：如果转送的值是在由编码器和解码器共同保持的扩展附加集中，那么，它被成功地解码，否则，解码器将可能限定它的编码并把它标识为(未知)扩展附加值。

在所有的情况下，出现扩展附加不应该影响当带扩展标志的类型套入某些其他类型内时认识后面材料的能力。

注 1：ASN.1 基本编码规则的所有变量和 ASN.1 的紧缩编码规则满足所有这些要求。用 ECN 定义的编码规则不必满足所有这些要求，但是也可能满足。

注 2：PER 和 BER 不标识扩展附加编码中的版本号。用 ECN 规定的编码可能提供也可能没提供这样的标识。

8 标记

8.1 标记是通过给出它的类别和类别中的号码来说明的，类别是下列之一：

通用

应用

专用

上下文规定

8.2 号码是一个非负整数，用十进制记法规定。

8.3 第 30 章中规定了 ASN.1 用户指派的标记的限制。

注：第 30 章中包括不允许本记法的用户显式地规定他们的 ASN.1 规范中通用类别标记的限制。标记的用法之间与另外三个类别没有形式上的差别。当采用应用类别标记时，通常可以应用专用或上下文规定类别标记替代，作为用户选择和风格。出现三个类别主要是出于历史原因，但是 E.2.12 中以通常采用类别的方式给出了指南。

8.4 表 1 总结了在本部分中规定的通用类别里标记的分配。

表 1 通用类别标记分配

通用类别 0	保留为编码规则使用
通用类别 1	布尔类型
通用类别 2	整数类型
通用类别 3	位串类型
通用类别 4	八位位组串类型
通用类别 5	空类型
通用类别 6	客体标识符类型
通用类别 7	客体描述符类型
通用类别 8	外部类型和单一实例类型
通用类别 9	实数类型
通用类别 10	枚举类型
通用类别 11	嵌入 pdv 类型
通用类别 12	UTF8 串类型
通用类别 13	相对客体标识符类型
通用类别 14-15	为本部分的将来版本保留
通用类别 16	序列和单一序列类型
通用类别 17	集合和单一集合类型
通用类别 18-22,25-30	字符串类型
通用类别 23-24	时间类型
通用类别 31-...	为本部分的附录保留

8.5 有些编码规则要求标记有正则顺序。为了一致，8.6 中定义了标记的正则顺序。

8.6 标记的正则顺序以每个类型的最外层标记为基础并定义如下：

a) 那些带通用类别标记的元素或替换项应首先出现，接着是那些带应用类别标记的，再是那些带上下文规定标记的，然后是那些带专用类别标记的。

b) 在标记的每个类别内，元素或替换项应以其标记数字的递增顺序出现。

9 ASN.1 记法的使用

9.1 类型定义的 ASN.1 记法为“Type”(见 16.1)。

9.2 类型的值的 ASN.1 记法为“Value”(见 16.7)。

注：在不知道类型的有关知识时，通常不能解释值的记法。

9.3 将类型指派给类型引用名的 ASN.1 记法应为“TypeAssignment”(见 15.1)、“ValueSetTypeAssignment”(见 15.6)、“ParameterizedTypeAssignment”(见 GB/T 16262.4，8.2)，或者“ParameterizedValueSetTypeAssignment”(见 GB/T 16262.4—2006 的 8.2)。

9.4 将值指派给值引用名的 ASN.1 记法应为“ValueAssignment”(见 15.2)或者“ParameterizedValueAssignment”(见 GB/T 16262.4—2006 的 8.2)。

9.5 记法“Assignment”的产生式替换项应只用在记法“ModuleDefinition”(12.1 注 2 中规定的除外)内。

10 ASN.1 字符集

10.1 除 10.2 和 10.3 的规定外，词项应由表 2 中列出的字符的序列组成。在表 2 中，字符由 GB/T 13000.1—1993 给出的名称标识。

表 2 ASN.1 字符

A 至 Z	拉丁大写字母 A 至拉丁大写字母 Z (LATIN CAPITAL LETTER A 到 LATIN CAPITAL LETTER Z)
a 至 z	拉丁小写字母 A 至拉丁小大写字母 Z (LATIN SMALL LETTER A 到 LATIN SMALL LETTER Z)
0 至 9	数字 0 至数字 9 (DIGIT ZERO 至 DIGIT 9)
!	感叹号 (EXCLAMATION MARK)
"	双引号 (QUOTATION MARK)
&	和 (AMPERSAND)
'	撇号 (APOSTROPHE)
(	左圆括号 (LEFT PARENTHESE)
)	右圆括号 (RIGHT PARENTHESE)
*	星号 (ASTERISK)
,	逗号 (COMMA)
-	负号 (HYPHEN-MINUS)
.	句号 (FULL STOP)
/	斜线 (SOLIDUS)
:	冒号 (COLON)
;	分号 (SEMICOLON)
<	小于符号 (LESS-THAN SIGN)
=	等于符号 (EQUALS SIGN)
>	大于符号 (GREATER-THAN SIGN)
@	商用 AT 符号 (COMMERCIAL AT)

表 2(续)

[	左方括号 (LEFT SQUARE BRACKET)
]	右方括号 (RIGHT SQUARE BRACKET)
^	向上箭号 (CIRCUMFLEX BRACKET)
_	下横线 (LOW LINE)
{	左花括号 (LEFT CURLY BRACKET)
|	竖线 (VERTICAL LINE)
}	右花括号 (RIGHT CURLY BRACKET)

注：等价的有关标准由我国标准化组织给出，附加字符可能在下面的词项中出现：

—typerreference(见 11.2)；

—identifier(见 11.3)；

—valuereference(见 11.4)；

—modulereference(见 11.5)；

当附加字符用在一种大小写无区别的文种时，由以上某些词项的第一个字符的不同情况导致的语义区别将用别的办法来处理。这就允许有效的 ASN.1 规范以各种文种书写。

10.2 当用该记法来规定字符串类型的值时，在 ASN.1 记法中可以出现已定义字符集中的所有字符，括以双引号 QUOTATION MARK(34) 字符(")(见 11.14)。

10.3 附加(任意)的图形符号可出现在"comment"词项中(见 11.6)。

10.4 印刷的字体、大小、色彩、亮度或其他显示特性无关紧要。

10.5 大写字母和小写字母应该视为不同。

10.6 ASN.1 定义在词项之间也可包含空白字符(见 11.1.6)。

11 ASN.1 词项

11.1 一般原则

11.1.1 下列各条规定词项中的字符。在每种情况下，都给出词项的名称，以及形成词项的字符序列的定义。

11.1.2 本第 11 章的各条中规定的每个词项(除多行"comment"、"bstring"、"hstring"和"cstring"外)不应包含空白(见 11.6、11.10、11.12 和 11.14)。

11.1.3 行的长度不受限制。

11.1.4 当使用非间隔指示符"&"(见 5.4)时，词项可以用一或多次空白(见 11.1.6)或注解(11.6)事件隔开。在"XMLTypeValue"产生式(见 15.2)中，词项之间可出现空白，但不应该出现"comment"词项。

注：这将避免"xmlcstring"词项内出现相邻连字符或星号和斜线产生的歧义。当"XMLTypeValue"产生式内出现这样的字符时，它们从不指明"comment"词项的开始点。

11.1.5 若后续词项的起始字符(或多个字符)是前项词项中字符末尾包括的容许字符(或多个字符)时，词项与后续词项间要用一个或多个空白或注解实例隔开。

11.1.6 本部分使用术语"新行"和"空白"。在机器可读的规范中的表示(行尾)空白和新行中，任意一个或多个下面的字符可用在任意组合中(对每个字符，给出 Unicode 编码标准规定的字符名称和字符代码)：

对于空白：

HORIZONTAL TABULATION (9)

LINE FEED (10)

VERTICAL TABULATION (11)

FORM FEED (12)

CARRIAGE RETURN (13)

SPACE (32)

对于新行：

LINE FEED (10)

VERTICAL TABULATION (11)

FORM FEED (12)

CARRIAGE RETURN (13)

注：任何是有效新行的字符或字符序列也是有效的空白。

11.2 类型引用

词项名——typereference

11.2.1 “typereference”应由任意个(一个或多个)字母、数字和连字符组成。以大写字母开头。不能用连字符结尾。一个连字符不能紧接另一个连字符。

注：有关连字符的规则是为了避免与(可能后随的)注解歧义。

11.2.2 “typereference”不应是 11.27 中列出的保留字符序列之一。

11.3 标识符

词项名——identifier

“identifier”应由任意个(一个或多个)字母、数字和连字符组成。以小写字母开头。不能用连字符结尾。一个连字符不能紧接另一个连字符。

注：有关连字符的规则是为了避免与(可能后随的)注解歧义。

11.4 值引用

词项名——valuereference

“valuereference”应由 11.3 中为“identifier”规定的字符序列组成。分析使用这一记法的实例时，“valuereference”由其出现的上下文来与“identifier”区别。

11.5 模块引用

词项名——modulereference

“modulereference”应由 11.2 中规定为“typereference”的字符序列组成。分析使用这一记法的实例时，“modulereference”由其出现的上下文来与“typereference”区别。

11.6 注解

词项名——comment

11.6.1 在 ASN.1 记法的定义中不引用“comment”。但它可以在任意时候出现在别的词项之间，而且没有语法意义。

注：虽然如此，在使用 ASN.1 的本部分上下文中，ASN.1 注解可含有与应用语义有关的标准文本或对语法的约束。

11.6.2 词项“comment”能有两种形式：

a) 如 11.6.3 定义的以“--”开始的一行注解；

b) 如 11.6.4 定义的以“/*”开始的多行注解。

11.6.3 只要“comment”以一对相连的连字符开始，它就应该以下一对相连的连字符或行尾为结束，无论哪一个在前面。注解除了开始的一对连字符及结尾的一对连字符(若有的话)外，不能有一对相连的连字符。如果以“--”开始的注解包括相连的字符“/*”或“*/”，那么它们没有特别的意义并且认为是注解的一部分。注解可以包括没有在 10.1 中规定的字符集中的图形符号(见 10.3)。

11.6.4 只要“comment”以“/*”开始，它就应该以对应的“*/”结束，不管这个“*/”是否在同一行。如果在“*/”之前发现另一个“/*”，那么当为每个“/*”发现配对的“*/”时注解应终止。如果以“/*”开始的注解包含两个相连的连字符“--”，这些连字符没有特别的意义并且认为是注解的一部分。注解可以包括在 10.1 中规定的字符集中没有的图形符号(见 10.3)。

注：这就允许用户注解已经含有注解的 ASN.1 模块部分。(不管它们是以"--"或"/*"开始)，只要被注解出的部分内没有含有"/*"或"*/"的字符串值，通过简单地在被注解的部分开头插入"/*"和其结尾插入"*/"。

11.7 空词项

词项名——empty

"empty"项不包括字符，当规定了产生式序列的替换项集时，它用于第 5 章的记法中以指明可能所有替换项缺失。

11.8 数

词项名——number

"number"应由一个或多个数字组成。除非"number"是单个数字，否则第一位数字不应该是 0。

注：通过将它解释成十进制记法，"number"词项总是映射成整数值。

11.9 实数

词项名——realnumber

"realnumber"应该由是一串一位或多位数字、而且可以选择十进制小数点(.)的整数部分组成。十进制小数点可以选择地后接一位或多位数字的分数部分。整数部分、十进制小数点或分数部分(不管哪个先出现)可以有选择地后接 e 或 E 和是一位或多位数字的选择签名指数。除非指数是单个数字，否则，它的起始数字不应该是零。

11.10 二进制数串

词项名——bstring

"bstring"应由任意个(可能 0 个)字符 0、1 组成，可能混杂着空白、前置 APOSTROPHE(39)字符(')，后随一对字符 'B。

例如：'011011100'B

二进制数串词项中的空白事件没有意义。

11.11 XML 二进制数串项

项名——xmlbstring

"xmlbstring"应该由任意个(可能零个)0、1 或者空白组成。二进制数串项中出现的任何空白字符没有意义。

例如：01101100

这一字符序列也是有效的"xmlhstring"和"xmlcstring"实例。分析使用这一记法的实例时，"xmlbstring"由其出现的上下文来与"xmlhstring"或"xmlcstring"区别。

11.12 十六进制数串

词项名——hstring

11.12.1 "hstring"应由任意个(可能 0 个)字符：A B C D E F 0 1 2 3 4 5 6 7 8 9 组成，可能混杂着空白，前置 APOSTROPHE(39)字符"'"，后随一对字符 'H。

例如：'AB0196'H 。

十六进制数串词项中的空白事件没有意义。

11.12.2 每个字符用十六进制表示法来表示 4 位的值。

11.13 XML 十六进制数串项

项名——xmlhstring

11.13.1 "xmlhstring"应该由任意个(可能零个)0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f 或者空白组成。十六进制数串项中出现的任何空白字符没有意义。

例如：Ab0196

11.13.2 用十六进制表示的每个字符来表示 4 位的值。

11.13.3 某些"xmlhstring"实例也是有效的"xmlbstring"和"xmlcstring"实例。分析使用这一记法的

实例时，“xmlhstring”由其出现的上下文来与“xmlbstring”或“xmlcstring”区别。

11.14 字符串

词项名——cstring

11.14.1 “cstring”应由任意个(可能0个)图形符号和来自字符串类型引用的字符集中的间隔字符组成，前、后置 QUOTATION MARK(34)字符(")。如果字符集中含有 QUOTATION MARK(34)字符，这一字符(如果出现在用“cstring”表示的字符串中)应用在不插入间隔字符的同一行中的一对 QUOTATION MARK(34)字符在“cstring”中表示。“cstring”可能跨过几行文字，在这种情况下，描述的字符串不应在“cstring”中行尾的前或后位置上包含有间隔字符。紧接“cstring”中行尾的前或后出现的空白没有意义。

注1：“cstring”只能用来无歧义地(在打印页上)表示字符串，该被描述的字符串中的每个字符要么指派一个图形符号，要么是一个间隔字符。当要在打印表示中指明包含控制字符的字符串时，可用替换项 ASN.1 语法(见第35章)。

注2：“cstring”表示的字符串应由与图形符号相关的字符和间隔字符组成。紧接“cstring”中任意行尾前或后的间隔字符不是所表示的字符串的一部分(它们被忽略掉)，当“cstring”中含有间隔字符串时，或字符汇中的图形符号在打印表示中不是无歧义时，“cstring”指示的字符串可能在那个打印表示中有歧义。

例1：“屎 屁 市 弒”

例2：“cstring”

"ABCDE FGH

IJK " " XYZ "

可用来表示类型 IA5String 的字符串值。所表示的值由下列字符组成：

ABCDE FGHIJK"XYZ

如果打印规范中使用成比例的间隔字体(如上所用的)，或者如果字符汇包含多个不同宽度的间隔字符，则 E 和 F 之间预计空间的准确数在打印表示中可能有歧义。

11.14.2 当字符是组合字符时，它应在“cstring”的打印表示中指明为一个单独的字符。它不应该用组成它的字符套印。(这样保证了打印版本中无歧义地定义了串中组成字符的顺序。)

例如：小写的“e”和重音组成字符在 GB/T 13000.1 中是两个字符，因此，在对应的“cstring”中打印为两个字符而不是单个字符 é 。

11.15 XML 字符串项

项名——xmlcstring

11.15.1 “xmlcstring”应该由任意个(可能0个)下面的 GB/T 13000.1 字符组成：

a) HORIZONTAL TABULATION (9)；

b) LINE FEED (10)；

c) CARRIAGE RETURN (13)；

d) 其 GB/T 13000.1 字符代码在 32(十六进制 20)至 55295(十六进制 D7FF)(含)范围内的任意字符；

e) 其 GB/T 13000.1 字符代码在 57344(十六进制 E000)至 65533(十六进制 FFFD)(含)范围内的任意字符；

f) 其 GB/T 13000.1 字符代码在 65536(十六进制 10000)至 1114111(十六进制 10FFFF)(含)范围内的任意字符；

注：要求强加的附加限制是采用的实例中“xmlcstring”应该只包含支配字符串类型允许的字符。

11.15.2 字符“&”(AMPERSAND)、“<”(LESS-THAN SIGN)或“>”(GREATER-THAN SIGN)应该只作为 11.15.4 或 11.15.5 规定的字符序列之一的一部分出现。

11.15.3 “xmlcstring”用来表示受限制字符串(见 37.9)的值，并且能用来直接或通过用下面规定的转

义序列表示 GB/T 13000.1 字符的所有组合。

注 1："xmlcstring"不能用来表示不在 GB/T 13000.1 中出现的字符，例如，能在 GeneralString 中出现的某些控制字符，也不能表示用代码超过 10FFFF 十六进制的 GB/T 13000.1 字符定义的字符。

注 2：当用符合 XML 处理器处理时，不能区分字符 LINE FEED (10)和 CARRIAGE RETURN(13)及一对 CARRIAGE RETURN+ LINE FEED。

11.15.4 如果字符"&"(AMPERSAND)、"<"(LESS-THAN SIGN)或">"(GREATER-THAN SIGN)在用"xmlcstring"(见 37.9)表示的抽象字符串值中出现，在"xmlcstring"中他们应该以下列方式之一表示：

a) 11.15.8 中规定的转义序列；或

b) 各自的转义序列"&"、"<"、">"。这些转义序列不应该包含空白(见 11.1.6)。

11.15.5 如果带表 3 中第一列的 GB/T 13000.1 的字符在用"xmlcstring"(见 37.9)表示的抽象字符串值中出现，它应该用表 3 中第二列的字符序列表示。这些字符序列不应该包含空白(见 11.1.6)。

注：这里不包括带十进制字符代码 9、10 和 13 及这些字符序列中所有小写的字母。

表 3 xmlcstring 中控制字符的转义序列

GB/T 13000.1 字符编码	xmlcstring 表示法
0(十六进制 0)	<nul/>
1(十六进制 1)	<soh/>
2(十六进制 2)	<stx/>
3(十六进制 3)	<etx/>
4(十六进制 4)	<eot/>
5(十六进制 5)	<enq/>
6(十六进制 6)	<ack/>
7(十六进制 7)	<bel/>
8(十六进制 8)	<bs/>
11(十六进制 B)	<vt/>
12(十六进制 C)	<ff/>
14(十六进制 E)	<so/>
15(十六进制 F)	<si/>
16(十六进制 10)	<dle/>
17(十六进制 11)	<dc1/>
18(十六进制 12)	<dc2/>
19(十六进制 13)	<dc3/>
20(十六进制 14)	<dc4/>
21(十六进制 15)	<nak/>
22(十六进制 16)	<syn/>
23(十六进制 17)	<etb/>
24(十六进制 18)	<can/>
25(十六进制 19)	
26(十六进制 1A)	<sub/>

表 3(续)

GB/T 13000.1 字符编码	xmlcstring 表示法
27(十六进制 1B)	<esc/>
28(十六进制 1C)	<is4/>
29(十六进制 1D)	<is3/>
30(十六进制 1E)	<is2/>
31(十六进制 1F)	<is1/>

11.15.6 当"xmlcstring"用在形成部分 XER 编码(见 ISO/IEC 8825-4)的"XMLTypedValue"中(见 15.2)时,它可能包含相连的 HYPHEN-MINUS(45)字符。当用在 ASN.1 模块中的 XML 值记法实例内时,它不应该包含两个相连的 HYPHEN-MINUS(45)字符。如果这一字符序列在用 ASN.1 模块中的"xmlcstring"表示的抽象字符串值中出现,那么,11.15.8 中规定的转义序列应该表示至少一个相连的 HYPHEN-MINUS(45)字符。

11.15.7 当"xmlcstring"用在形成部分 XER 编码(见 ISO/IEC 8825-4)的"XMLTypedValue"中时,它可能包含以任意顺序相连的 ASTERISK(42)和 SOLIDUS(47)字符。当用在 ASN.1 模式中的 XML 值记法实例内时,它不应该包含(以任意顺序)相连的 ASTERISK(42)和 SOLIDUS(47)字符。如果这一字符序列在用"xmlcstring"(见 37.9)表示的抽象字符串值中出现,那么,11.15.8 中规定的转义序列应该表示至少一个相连的 ASTERISK(42)和 SOLIDUS(47)字符。

11.15.8 能直接在"xmlcstring"中出现的任何字符也能在"xmlcstring"中用形式"&#n;"(其中 n 是十进制记法中的 GB/T 13000.1 字符代码)或形式"&#xn;"(其中 n 是十六进制记法中的 GB/T 13000.1 字符代码)的转义序列表示。这些转义序列不应该包含空白(见 11.1.6)。

注 1:在"n"的十进制和十六进制值中允许以 0 开始,而且"A"~"F"的大小写都能用在十六进制值中。

注 2:如果转义序列"&#n"和"&#xn"用于不在基本多文种平面(BMP)中的 GB/T 13000.1 字符,"n"值将比 65535(十六进制 FFFF)大。

示例——"xmlcstring":

ABCDéFGHîJK&XYZ

能用来表示类型 UTF8String 的字符串值。表示的值由下面字符组成:

ABCDé FGHîJK&XYZ

如果在规范中采用成比例的间隔字体(例如上面),其中 é 和 F 之间的精确间隔字符在印刷媒介上可能有歧义。

11.16 赋值词项

词项名——"::="

该词项应由字符序列 ::= 组成。

注:该序列不包含任何空白字符(见 11.1.2)。

11.17 范围分隔符

词项名——".."

该词项应由字符序列 .. 组成。

注:该序列不包含任何空白字符(见 11.1.2)。

11.18 省略号

词项名——"..."

该词项应由字符序列 ... 组成。

注:该序列不包含任何空白字符(见 11.1.2)。

11.19 左版本括号

词项名——"[["

该词项应由字符序列 [[组成。

注：该序列不包含任何空白字符(见 11.1.2)。

11.20 右版本括号

词项名——"]]"

该词项应由字符序列]] 组成。

注：该序列不包含任何空白字符(见 11.1.2)。

11.21 XML 结尾标记开始项

项名——"</"

该项应由字符序列 </ 组成。

注：该序列不包含任何空白字符(见 11.1.2)。

11.22 XML 单个标记结尾项

项名——"/>"

该项应由字符序列 /> 组成。

注：该序列不包含任何空白字符(见 11.1.2)。

11.23 XML 布尔真项

项名——"true"

11.23.1 该项应由字符序列 true 组成。

11.23.2 分析使用这一记法的实例时，"true"由其出现的上下文来与"valuereference"或"identifier"区别。

注：该序列不包含任何空白字符(见 11.1.2)。

11.24 XML 布尔假项

项名——"false"

11.24.1 该项应由字符序列 false 组成。

11.24.2 分析使用这一记法的实例时，"false"由其出现的上下文来与"valuereference"或"identifier"区别。

注：该序列不包含任何空白字符(见 11.1.2)。

11.25 ASN.1 类型的 XML 标记名称

项名——xmlasn1typename

11.25.1 当 ASN.1 固有类型将用作 XML 标记名称时，本部分采用项"xmlasn1typename"。

11.25.2 表 4 为 16.2 中列出的每个 ASN.1 固有类型列出形成"xmlasn1typename"的字符序列。ASN.1 固有类型通过它的产生式名称在表 4 的第一列中标识。应该用于"xmlasn1typename"的字符序列在表 4 的第二列中标识，这些字符序列的前或后没有空白。

11.25.3 "UsefulType"的"xmlasn1typename"应该是用于它们定义的"typereference"。

11.25.4 "ObjectClassFieldType"和"InstanceOfType"的"xmlasn1typename"项中的字符序列在 GB/T 16262.2的 14.1 和附录 C 中规定。

11.25.5 如果 ASN.1 固有类型是"TaggedType"，那么，确定"xmlasn1typename"的类型应该是"TaggedType"中的"Type"(见 30.1)。如果这本身是"TaggedType"，那么，应该递归地用本条。

表 4 xmlasn1typename 的字符

ASN.1 类型产生式名称	xmlasn1typename 中的字符
BitStringType	BIT_STRING
BooleanType	BOOLEAN
ChoiceType	CHOICE

表 4(续)

ASN.1 类型产生式名称	xmlasn1typename 中的字符
EmbeddedPDVType	SEQUENCE
EnumeratedType	ENUMERATED
ExternalType	SEQUENCE
InstanceOfType	SEQUENCE
IntegerType	INTEGER
NullType	NULL
ObjectClassFieldType	见 GB/T 16262.2 的 14.10 和 14.11
ObjectIdentifierType	OBJECT_IDENTIFIER
OctetStringType	OCTET_STRING
RealType	REAL
RelativeOIDType	RELATIVE_STRING
RestrictedCharacterStringType	类型名称(如:IA5Sting)
SequenceType	SEQUENCE
SequenceOfType	SEQUENCE_OF
SetType	SET
SetOfType	SET_OF
TaggedType	见 11.25.5
UnrestrictedCharacterStringType	SEQUENCE

11.26 单个字符词项

词项名——

"{" "}" "<" ">""," "." "(" ")" "[" "]" "-"(HYHEN-MINUS) ":" "=" """(QUOTATION MARK) "'"(APOSTROPHE) " "(SPACE)";" "@" "|" "!" "^"

带任意上面所列名称的词项应由无引号的单个字符组成。

11.27 保留字

保留字名——

ABSENT	ENCODED	INTEGER	RELATIVE-OID
ABSTRACT-SYNTAX	END	INTERSECTION	SEQUENCE
ALL	ENUMERATED	ISO646String	SET
APPLICATION	EXCEPT	MAX	SIZE
AUTOMATIC	EXPLICIT	MIN	STRING
BEGIN	EXPORTS	MINUS-INFINITY	SYNTAX
BIT	EXTENSIBILITY	NULL	T61String
BMPString	EXTERNAL	NumericString	TAGS
BOOLEAN	FALSE	OBJECT	TeletexString
BY	FROM	ObjectDescriptor	TRUE
CHARACTER	GeneralizedTime	OCTET	TYPE-IDENTIFIER

CHOICE	GeneralString	OF	UNION
CLASS	GraphicString	OPTIONAL	UNIQUE
COMPONENT	IA5String	PATTERN	UNIVERSAL
COMPONENTS	IDENTIFIER	PDV	UniversalString
CONSTRAINED	IMPLICIT	PLUS-INFINITY	UTCTime
CONTAINING	IMPLIED	PRESENT	UTF8String
DEFAULT	IMPORTS	PrintableString	VideotexString
DEFINITIONS	INCLUDES	PRIVATE	VisibleString
EMBEDDED	INSTANCE	REAL	WITH

带上面名称的词项应由名称中的字符序列组成，且是保留的字符序列。

注1：这些序列中没有空白。

注2：本部分中不用关键字：CLASS、CONSTRAINED、CONTAINING、ENCODED、INSTANCE、SYNTAX 和 UNIQUE，它们在 GB/T 16262.2、GB/T 16262.3、GB/T 16262.4 中使用。

12 模块定义

12.1 “ModuleDefinition”由下面的产生式规定：

```
ModuleDefinition::=
        ModuleIdentifier
        DEFINITIONS
        TagDefault
        ExtensionDefault
        " ::= "
        BEGIN
        ModuleBody
        END
ModuleIdentifier::=
        modulereference
        DefinitiveIdentifier
DefinitiveIdentifier::=
        "{" DefinitiveObjIdComponentList " }"   |
        empty
DefinitiveObjIdComponentList::=
        DefinitiveObjIdComponent                |
        DefinitiveObjIdComponent DefinitiveObjIdComponentList
DefinitiveObjIdComponent::=
        NameForm                                |
        DefinitiveNumberForm                    |
        DefinitiveNameAndNumberForm
DefinitiveNumberForm::= number
DefinitiveNameAndNumberForm::=identifier " (" DefinitiveNumberForm " )"
TagDefault::=
        EXPLICIT TAGS                           |
        IMPLICIT TAGS                           |
```

```
        AUTOMATIC TAGS                          |
        empty
ExtensionDefault::=
        EXTENSIBILITY IMPLIED                   |
        empty
ModuleBody::=
        Exports Imports AssignmentList          |
        empty
Exports::=
        EXPORTS SymbolsExported " ; "           |
        EXPORTS ALL " ; "                       |
        empty
SymbolsExported::=
        SymbolList                              |
        empty
Imports::=
        IMPORTS SymbolsImported " ; "           |
        empty
SymbolsImported::=
        SymbolsFromModuleList                   |
        empty
SymbolsFromModuleList::=
        SymbolsFromModule                       |
        SymbolsFromModuleList SymbolsFromModule
SymbolsFromModule::=
        SymbolList FROM GlobalModuleReference
GlobalModuleReference::=
        modulereference AssignedIdentifier
AssignedIdentifier::=
        ObjectIdentifierValue                   |
        DefinedValue                            |
        empty
SymbolList::=
        Symbol                                  |
        SymbolList " ," Symbol
Symbol::=
        Reference                               |
        ParameterizedReference
Reference::=
        typereference                           |
        valuereference                          |
        objectclassreference                    |
        objectreference                         |
```

```
        objectsetreference
AssignmentList::=
        Assignment                  |
        AssignmentList Assignment
Assignment::=
        TypeAssignment              |
        ValueAssignment             |
        XMLValueAssignment          |
        ValueSetTypeAssignment      |
        ObjectClassAssignment       |
        ObjectAssignment            |
        ObjectSetAssignment         |
        ParameterizedAssignment
```

注 1：在 GB/T 16262.4 中规定了“Exports”和“Imports”列表里的“ParameterizedReference”的用法；

注 2：例如(及对于带通用类别标记类型的本部分中的定义)，“ModuleBody”可用在“ModuleDefinition”的外部。

注 3：第 15 章中规定了“TypeAssignment”、“ValueAssignment”、“XMLValueAssignment”和“ValueSetTypeAssignment”产生式。

注 4：为模块定义的“TagDefault”值只影响那些在模块中显式已定义类型，它不影响引入类型的解释。

注 5：字符分号不在赋值列表规范或它的任何下一级产生式中出现，并保留为 ASN.1 工具开发者使用。

12.2 如果“TagDefault”是“empty”，它取“EXPLICIT TAGS”。

注：第 30 章给出了 EXPLICIT TAGS、IMPLICIT TAGS 和 AUTOMATIC TAGS 的意义。

12.3 当选择“TagDefault”的替换项 AUTOMATIC TAGS 时，就是说选择了为模块自动加标记，否则就说没有选择。自动加标记是(带附加条件)应用于模块定义内的“ComponentTypeLists”和“AlternativeTypeLists”产生式上的语法转换。24.7 至 24.9、26.3 和 28.2 至 28.5 形式上分别规定了有关序列类型、集合类型和选择类型记法的这一转换。

12.4 EXTENSIBILITY IMPLIED 选项相当于允许的模块中每个类型定义中文本插入扩展标志(“...”)。隐式扩展标志的位置是允许有显式规定扩展标志的类型中的最后位置。无 EXTENSIBILITY IMPLIED 意味着只为扩展标志显式存在的模块内的那些类型提供可扩展性。

注：EXTENSIBILITY IMPLIED 只影响类型。它不影响客体集和子类型约束。

12.5 在“ModuleIdentifier”产生式中出现的“modulereference”称为模块名称。

注：用赋予相同“modulereference”的“ModuleBody”的几个事件定义单个 ASN.1 模块的可能性在早期的规范中是(可论证)允许的。本部分中不允许这样。

12.6 模块名在模块定义的应用范围内应只能使用一次(12.9 规定的除外)。

12.7 如果“DefinitiveIdentifier”不是空的，指示客体标识符值无歧义和唯一地标识所定义的模块。未定义值可用于定义客体标识符值。

注：本部分不阐述模块变化需要新“DefinitiveIdentifier”的问题。

12.8 如果“AssignedIdentifier”不是空，则“ObjectIdentifierValue”和“DefinedValue”替换项无歧义和唯一地标识引入引用名的模块。当使用“AssignedIdentifier”的“DefinedValue”替换项时，它应是类型客体标识符值。“AssignedIdentifier”内文本出现的每个“valuereference”应满足下列规则之一：

a) 它在所定义模块的“AssignmentList”中定义，且所有在赋值描述右边文本出现的“valuereference”也满足本规则(“a”规则)或下一个规则(“b”规则)。

b) 在“AssignmentIdentifier”不能文本包含任何“valuereference”的“SymbolsFromModule”中，它表现为“Symbol”。

注：建议赋予一个客体标识符以便其他地方能无歧义地引用模块。

12.9 “SymbolsFromModule”中的“GlobalModuleReference”应在另一个模块的“ModuleDefinition”中出现，如果它包含非空的“DefinitiveIdentifier”，“modulereference”可能在两种情况下有所不同例外。

注：当符号是从两个名字相同（不考虑 12.6 已命名模块）的模块中引入时，应只能用不同于用于其他模块中的“modulereference”。采用替换项的不同名字使这些名字可用于模块体中（见 12.15）。

12.10 当“modulereference”和非空“AssignedIdentifier”都用来引用一个模块时，后者应该认为是确定性的。

12.11 当引用的模块有非空的“DefinitiveIdentifier”时，引用那个模块的“GlobalModuleReference”不应有空的“AssignedIdentifier”。

12.12 当选择“SymbolsExported”替换“Exports”时：

a) “SymbolsExported”中的每个“Symbol”应满足一个且只能是一个下面的条件：

i) 只在所构建的模块中定义；或

ii) 在“Imports”的“SymbolsImported”替换项中，只准确地出现一次；

b) 每个适合从模块外面引用的“Symbol”应包含在“SymbolsExported”中且只有这些“Symbol”能从模块外部引用（根据 12.13 的规定放宽限制）；和

c) 如果没有这类“Symbol”，那么应选择空替换“SymbolsExported”（不是替换“Exports”）。

12.13 当选择“empty”或者 EXPORTS ALL 替换“Exports”时，模块中定义或模块引入的每个“Symbol”可从其他受到 12.12 a)中规定限制的模块中引用。

注：包含“empty”替换“Exports”是为了向后兼容。

12.14 如果定义它们的 typereference 是输出的或作为输出类型内的成分（或子成分）出现，则“NamedNumberList”、“Enumeration”或“NamedBitList”中出现的标识符是隐式输出。

12.15 当选择“SymbolsImported”替换“Imports”时：

a) 在“SymbolsFromModue”中的每个“Symbol”应在“SymbolsFromModue”中的“GlobalModuleReference”指明模块的模块体中定义，或者在“Imports”章出现。如果在那一章中只有一次“Symbol”事件，则只允许引入被引用模块的“Imports”章中出现的“Symbol”，而且“Symbol”不能在被引用模块中定义。

注 1：不禁止在两个来自被引入到另一个模块的不同模块中定义相同的符号名。然而，如果相同的“Symbol”名字在模块 A 的“Imports”章中出现不止一次，那个“Symbol”名就不能从 A 输出以引入到另一个模块 B。

b) 如果在“SymbolsFromModule”中的“GlobalModuleReference”指明的模块的定义中选择了“SymbolsExported”替换“Exports”，则“Symbol”应出现在其“SymbolsExported”中。

c) 只有那些出现在“SymbolsFromModue”的“SymbolList”之中的“Symbol”可在任何有由那个“SymbolsFromModue”的“GlobalModuleReference”指明的“modulereference”的“External＜X＞Reference”（这里＜X＞是“Value”、“Type”、“Object”、“Objectclass”或“Objectset”）中作为符号出现。

d) 如果没有这样的“Symbol”，那么应选择“empty”替换“SymbolsImported”。

注 2：c)和 d)的作用是语句 IMPORTS；意味着模块不能包含“External＜X＞Reference”。

e) 在“SymbolsFromModueList”中的所有“SymbolsFromModue”应包含“GlobalModuleReference”出现，因此：

i) 其中的“modulereference”彼此完全不同，而且不同于与引用模块相关的“modulereference”；和

ii) 非空时，“AssignedIdentifier”指明彼此完全不同的客体标识符值，而且不同于（若有的话）与引用模块相关的客体标识符值。

12.16 当选择“empty”替换“Imports”时，模块仍可通过“External＜X＞Reference”的方法引用其他模块中定义的“Symbol”。

注：含有“empty”替换“Imports”是为了向后兼容。

12.17 如果定义它们的 typereference 是引入的或作为引入类型的成分(或子成分)出现的，则在“NamedNumberList”、“Enumeration”或“NamedBitList”中出现的标识符是隐式引入。

12.18 “SymbolsFromModule”中的“Symbol”可以在“ModuleBody”中作为“Reference”出现，与“Symbol”相关的意义是它在对应的“GlobalModuleReference”指明的模块中出现。

12.19 当“Symbol”也出现在“AssignmentList”(不赞成)，或者在“SymbolsFromModule”的一个或多个其他实例中出现时，它应仅用在“External<X>Reference”中。它没有这样出现时，应直接用作“Reference”。

12.20 本部分中随后的几章里定义了“Assignment”的各种替代式，另外标注的除外：

赋值替代式	定义章节
“TypeAssignment”	15.1
“ValueAssignment”	15.2
“XMLValueAssignment”	15.2
“ValueSetTypeAssignment”	15.6
“ObjectClassAssignment”	GB/T 16262.2 的 9.1
“ObjectAssignment”	GB/T 16262.2 的 11.1
“ObjectSetAssignment”	GB/T 16262.2 的 12.1
“ParameterizedAssignment”	GB/T 16262.4 的 8.1

每个“Assignment”的第一个符号是“Reference”的替换项之一，指明被定义的引用名。“AssignmentList”内的两个赋值中不应有相同的引用名。

13 引用类型和值定义

13.1 被定义类型和值产生式

```
DefinedType::=
        ExternalTypeReference       |
        Typereference               |
        ParameterizedType           |
        ParameterizedValueSetType
DefinedValue::=
        ExternalValueReference      |
        Valuereference              |
        ParameterizedValue
```

规定应用于引用类型和值定义的序列。GB/T 16262.4 中规定了“ParameterizedType”和“ParameterizedValueSetType”标识的类型和“ParameterizedValue”标识的值。

13.2 当需要 XML 标记名表示 ASN.1 类型时，采用“NonParameterizedTypeName”产生式：

```
NonParameterizedTypeName::=
        ExternalTypeReference       |
        typereference               |
        xmlasn1typename
```

13.3 如果“xmlasn1typename”是“CHOICE”、“ENUMERATED”、“SEQUENCE”、“SEQUENCE_OF”、“SET”或“SET_OF”，当 XML 值记法用于 ASN.1 模块中时，第三个替换项不应该用作“XMLValueAssignment”(见 15.2)或“XMLOpenTypeFeildVal”(见 GB/T 16262.2 的 14.6)的“XMLTypeValue”中的“NonParameterizedTypeName”。

注：因为“xmlasn1typename”不定义 ASN.1 类型，因此，在用于 ASN.1 模块中的 XML 值记法中这一限制是强制的。因为不用从“xmlasn1typename”形成的 XML 标记来确定所编码的类型，因此，对于编码规则（例如 XER，见 ISO/IEC 8825-4）中使用这一记法不出现该限制。

13.4 除 12.18 规定的外，除非引用是在类型或值指派给（见 15.1 和 15.2）“typereference”或“valuereference”的“ModuleBody”内，否则不应使用“typereference”、“valuereference”、“ParameterizedType”、“ParameterizedValueSetType”或“ParameterizedValue”替代式。

13.5 除非对应的“typereference”或“valuereference”在用于定义对应的“modulereference”的“ModuleBody”内

a) 分别被赋予类型或值（见 15.1 和 15.2）；或

b) 出现在“Imports”章中，

否则不应使用“ExternalTypeReference”和“ExternalValueReference”。如果在那一章中，“Symbol”事件不超过一次，则仅允许引用另一个模块的“Imports”章中的名称。

注：这不禁止在两个来自被引入到另一模块的不同模块中定义的相同“Symbol”，然而，如果相同的“Symbol”在模块 A 的 IMPORTS 章中出现不止一次，那么不能在外部引用中用模块 A 引用那个“Symbol”。

13.6 在只引用在不同模块中定义的模块名的模块中应采用外部引用，而且用下面的产生式规定：

```
ExternalTypeReference::=
        modulereference
        "."
        typereference
ExternalValueReference::=
        modulereference
        "."
        valuereference
```

注：GB/T 16262.2 中规定了附加外部引用产生式（“ExternalClassReference”、“ExternalObjectReference”和“ExternalObjectSetReference”）。

13.7 当用“Imports”的“SymbolsImported”替换项定义引用模块时，“SymbolsImported”里只有一个“SymbolsFromModule”的“GlobalModuleReference”中应出现外部引用中的“modulereference”。当用“Imports”的“empty”替换项定义引用模块时，定义“Reference”的模块（不同于引用模块）的“ModuleDefinition”中应出现外部引用的“modulereference”。

13.8 在“DefinedType”用作“Type”支配的部分记法时（例如，在“SubtypeConstraint”中），那么，“DefinedType”应该与附录 B.6.2 中规定的支配“Type”相兼容。

13.9 “DefinedValue”的 ASN.1 规范内的每次出现由“Type”支配，而且“DefinedValue”应该引用与附录 B.6.2 中规定的支配“Type”相兼容的类型的值。

14 支持引用 ASN.1 成分的记法

14.1 在很多情况下，需要形式引用 ASN.1 类型、值等的成分。一个这样的实例是要写出文本来标识某些 A SN.1 模块内的规定类型。本章定义能用来提供这些引用的记法。

14.2 记法使集或序列类型（强制或可选地出现在类型中）的任何成分能被标识。

14.3 通过使用“AbsoluteReference”语法结构：

```
AbsoluteReference::= "@" ModuleIdentifier
        "."
        ItemSpec
ItemSpec::=
```

```
                typereference           |
                ItemId "." "ComponentId
ItemId ::= ItemSpec
ComponentId ::=
                Identifier              |
                Number
                "*"
```

能引用任何 ASN.1 类型定义的任何部分。

注：AbsoluteReference 产生式不可用在本部分的其他地方。提供它是为 14.1 的陈述。

14.4 "ModuleIdentifier"标识 ASN.1 模块(见 12.1)。

14.5 当"DefinitiveIdentifier"的第一个替换项用作部分"ModuleIdentifier"时，"DefinitiveIdentifier"无歧义和唯一地标识名称被引用的模块。

14.6 "typereference"引用由"ModuleIdentifier"标识的模块中定义的任何 ASN.1 类型。

14.7 每个"ItemSpec"中的"ComponentId"标识由"ItemId"标识的类型的成分。如果它标识的成分不是集、序列、单一集合、单一序列或选择类型，它应该是最后的"ComponentId"。

14.8 如果双亲"ItemId"是集或序列类型，并且要求是该集或序列"ComponentTypeLists"中"NamedType"的"identifier"之一，可以使用"ComponentId"的"identifier"形式。如果"ItemId"标识选择类型，且然后要求是那个选择类型"AlternativeTypeLists"中"NamedType"的"identifier"之一，也可使用"ComponentId"的"identifier"形式。它不能在任何其他情况下使用。

14.9 只有"ItemId"是单一序列或单一集合类型的条件下能使用"ComponentId"的数字形式。数字值标识单一序列或单一集合中的类型实例，用值"1"标识类型的第一个实例。值零标识概念整数类型成分(不在传送中显式出现)，该成分包含封闭类型的值中出现的单一序列或单一集合中的类型实例数字的计数。

14.10 只有"ItemId"是单一序列或单一集合时可以使用"ComponentId"的"*"形式。任何与使用"ComponentId"的"*"形式相关的语义适用于单一序列和单一集合的所有成分。

注：下面的示例中：

```
M DEFINITIONS ::= BEGIN
    T ::= SEQUENCE {
        a    BOOLEAN
        b    SET OF INTEGER
    }
END
```

成分"T"能通过 ASN.1 模块外(或注解中)的文本引用，如：

```
-- if (@M.T.b.0 is odd) then:
--      (@M.T.b.* shall be an odd integer)
```

用来陈述如果"b"里的成分数字是奇数，"b"的所有成分必须是奇数。

15 类型和值的赋值

15.1 通过"TypeAssignment"产生式规定的记法将"typereference"赋予类型：

```
TypeAssignment ::=
                typereference
                "::="
                Type
```

"typereference"不应该是 ASN.1 的保留字(见 11.27)。

15.2 通过“ValueAssignment”或“XML ValueAssignment”产生式规定的记法将“valuereference”赋予值：

```
ValueAssignment ::=
        valuereference
        Type
        " ::= "
        Value
XML ValueAssignment ::=
        valuereference
        " ::= "
        XMLTypedValue
XMLTypedValue ::=
        "<"&NonParameterizedTypeName ">"
        XMLValue
        "</"& NonParameterizedTypeName ">"
        " <"& NonParameterizedTypeName " />"
```

赋给“ValueAssignment”中“valuereference”的值应该是“Value”，并且由“Type”支配和应该是由“Type”定义的类型的值的记法(如 15.3 中规定的)。赋给“XMLValueAssignment”中“valuereference”的值应该是“XMLValue”(见 16.7)，并且应该是“NonParameterizedTypeName”定义的类型的值的记法(如 15.4 中规定的)。如果这是“xmlasn1typename”项，那么它标识表 4 中对应行的 ASN.1 固有类型(也见 13.3)。

15.3 “Value”是 16.7 中规定类型的值的记法。

15.4 如果“XMLValue”是类型的“XMLBuiltinValue”记法(见 16.10)，则“XMLValue”是类型的值的记法。

15.5 只有“XMLValue”产生式实例为空时，可以使用“XMLTypedValue”的第二个替换项(采用 XML 空元素标记)。

注：如果“XMLValue”产生式是只包含空白的“xmlcstring”，这将不能为空，而且，不能使用第二个替换项。

15.6 通过“ValueSetTypeAssignment”产生式规定的记法可将“typereference”赋给值集：

```
ValueSetTypeAssignment ::=
        typereference
        Type
        " ::= "
        ValueSet
```

这一记法将“typereference”赋给定义为“Type”指示的类型的子类型，并且确定包含“ValueSet”中规定或允许的值的类型。“typereference”不应该是 ASN.1 的保留字(见 11.27)，且可引用作为类型。在 15.7 中定义了“ValueSet”。

15.7 某些类型支配的值集应由记法“ValueSet”规定：

```
ValueSet::= " {" ElementSetSpecs "}"
```

值集包含“ElementSetSpecs”规定的所有值(见第 46 章)，至少应该包含有一个。

15.8 “ValueSetTypeAssignment”产生式扩大成：

```
typereference
        Type
        " ::= "
```

"(" ElementSetSpecs")"

为了所有目的，包括应用编码规则，定义它是准确地相当于使用具有相同的"Type"和"ElementSetSpecs"规范的产生式：

typereference
"::="
Type
"(" ElementSetSpecs ")"

16 类型和值的定义

16.1 类型由下列记法"Type"规定：

Type ::= BuiltinType | ReferencedType | ConstrainedType

16.2 记法"BuiltinType"规定 ASN.1 的固有类型，定义如下：

BuiltinType ::=

BitStringType	|
BooleanType	|
CharacterStringType	|
ChoiceType	|
EmbeddedPDVType	|
EnumeratedType	|
ExternalType	|
InstanceOfType	|
IntegerType	|
NullType	|
ObjectClassFieldType	|
ObjectIdentifierType	|
OctetStringType	|
RealType	|
RelativeOIDType	|
SequenceType	|
SequenceOfType	|
SetType	|
SetOfType	|
TaggedType	

以下几章(除非另有说明，否则是指本部分中的)定义各种"固有类型"记法：

BitStringType	21
BooleanType	17
CharacterStringType	36
ChoiceType	28
EmbeddedPDVType	33
EnumeratedType	19
ExternalType	34
InstanceOfType	GB/T 16262.2—2006 的附录 C
IntegerType	18

NullType	23
ObjectClassFieldType	GB/T 16262.2—2006 的 14.1
ObjectIdentifierType	31
OctetStringType	22
RealType	20
RelativeOIDType	32
SequenceType	24
SequenceOfType	25
SetType	26
SetOfType	27
TaggedType	30

16.3 记法“ReferencedType”规定 ASN.1 引用的类型：

```
ReferencedType ::=
        DefinedType             |
        UsefulType              |
        SelectionType           |
        TypeFromObject          |
        ValueSetFromObject      |
```

“ReferencedType”记法提供了引用某些其他类型(最终到固有类型)的替换方法。各种“ReferencedType”记法以及确定它们引用的类型的方法在下列位置规定(除非另有说明，否则在本部分中)：

DefinedType	13.1
UsefulType	41.1
SelectionType	29
TypeFromObject	GB/T 16262.2—2006 的第 15 章
ValueSetFromObject	GB/T 16262.2—2006 的第 15 章

16.4 “ConstrainedType”在第 45 章中定义。

16.5 本部分要求在规定集类型、序列类型和选择类型的成分时采用记法“NamedType”。“NamedType”的记法是：

```
NamedType ::= identifier Type
```

16.6 “identifier”用来无歧义地引用值记法、内部子类型约束及成分关系约束中的集类型、序列类型或选择类型的成分(见 GB/T 16262.3)。它不是类型的一部分，因此，不影响类型。

16.7 有些类型的值应该由下列记法“Value”或记法“XMLValue”来规定：

```
Value ::=
        BuiltinValue                |
        ReferencedValue             |
        ObjectClassFieldValue
XMLValue ::=
        XMLBuiltinValue             |
        XMLObjectClassFieldValue
```

注 1：“ObjectClassFieldValue”和“XML ObjectClassFieldValue”在 GB/T 16262.2—2006 的 14.6 中定义。

注 2：“XMLValue”只用于“XMLTypedValue”中。

16.8 如果“XMLValue”产生式的任意部分导致 XML 开始标记后紧接着 XML 结束标记，可能由以 11.1.4 允许插入的空白隔开(例如，<字段 1></字段 1>)，这两个 XML 标记以及任意插入的空白能

用单个 XML 空-元素标记(<字段 1/>)代替。

注：如果任意空白字符，11.1.4 允许插入的空白除外，出现在开始标记的最后“>”字符和结束标记的初始“<”字符之间，就不满足上面的条件。

16.9 ASN.1 的固有类型值能用记法“XMLBuiltinValue”(见 16.10)或“BuiltinValue”规定，定义如下：

```
BuiltinValue::=
        BitStringValue
        BooleanValue
        CharacterStringValue
        ChoiceValue
        EmbeddedPDVValue
        EnumeratedValue
        ExternalValue
        InstanceOfValue
        IntegerValue
        NullValue
        ObjectIdentifierValue
        OctetStringValue
        RealValue
        RelativeOIDValue
        SequenceValue
        SequenceOfValue
        SetValue
        SetOfValue
        TaggedValue
```

各种“BuiltinValue”记法的每一个在上面 16.2 中列出的，对应的“BuiltinType”记法的同一条中定义。

16.10 “XML BuiltinValue”定义如下：

```
XMLBuiltinValue::=
        XMLBitStringValue
        XMLBooleanValue
        XMLCharacterStringValue
        XMLChoiceValue
        XMLEmbeddedPDVValue
        XMLEnumeratedValue
        XMLExternalValue
        XMLInstanceOfValue
        XMLIntegerValue
        XMLNullValue
        XMLObjectIdentifierValue
        XMLOctetStringValue
        XMLRealValue
        XMLRelativeOIDValue
```

XMLSequenceValue |
XMLSequenceOfValue |
XMLSetValue |
XMLSetOfValue |
XMLTaggedValue

各种"XMLBuiltinValue"记法的每一个在上面 16.2 中列出的,对应的"BuiltinType"记法的同一章中定义。

16.11 用记法"ReferencedValue"规定 ASN.1 引用的值:

ReferencedValue ::=
DefinedValue |
ValueFromObject

"ReferencedValue"记法提供了引用某些其他值(最终到固有值)的替换方法。各种"ReferencedValue"记法以及确定它们引用的值的方法在下列位置规定(除非另有说明,否则在本部分中):

DefinedValue	13.1
ValueFromObject	GB/T 16262.2—2006 的第 15 章

16.12 不管类型是"BuiltinType"、"ReferencedType"或"ConstrainedType",其值能够用那个类型的"BuiltinValue"或"ReferencedValue"规定。

16.13 用记法"NamedType"引用的类型的值应该用记法"NamedValue"定义,或者当用作部分"XMLValue"时,用记法"XMLNamedValue"。这些产生式是:

NamedValue ::= identifier Value

XMLNamedValue ::= " <"& identifier " >"XMLValue " </ "& identifier " >"

这里,"identifier"与"NamedType"记法中使用的一样。

注:"identifier"是记法的一部分,它不形成值本身的部分。它用来无歧义地引用集类型、序列类型或选择类型的成分。

16.14 类型定义中隐式(见 12.4)或显式出现的扩展标志(见第 6 章)不影响值记法。也就是,有扩展标志的类型的值记法准确地和好像缺少扩展标志一样。

注:46.8 条禁止用于取自引用不在双亲类型扩展根中的值的子类型约束的值记法。

17 布尔类型记法

17.1 应该通过记法"BooleanType"引用布尔类型(见 3.6.7):

BooleanType ::= BOOLEAN

17.2 用本记法定义类型的标记是通用类别,编号为 1。

17.3 布尔类型(见 3.6.73 和 3.6.38)的值应该由记法"BooleanValue"定义,或者当用作"XMLValue"时,用记法"XMLBooleanValue"。这些产生式是:

BooleanValue ::= TRUE | FALSE

XMLBooleanValue ::=
" <"& " true" " />" |
" <"& " false" " />"

18 整数类型的记法

18.1 整数类型应该由记法"IntegerType"来引用(见 3.6.41):

IntegerType ::=
INTEGER |

```
                INTEGER"{" NamedNumberList" }"
NamedNumberList ::=
                NamedNumber             |
                NamedNumberList "," NamedNumber
NamedNumber ::=
                identifier " ("SignedNumber" )"   |
                identifier " ("DefinedValue " )"
SignedNumber ::=
                number                  |
                "-" number
```

18.2 如果“number”是零，则不应该使用“SignedNumber”的第二个替换项。

18.3 “NamedNumberList”在类型定义中没有意义，它仅用于 18.9 中规定的值记法。

18.4 “DefinedValue”中的“valuereference”应该是类型整数的。

注：由于“identifier”不能用来规定与“NamedNumber”相关的值，“DefinedValue”永远不会被错认为是“IntegerValue”。在下面的情况下：

```
a INTEGER ::= 1
T1 ::= INTEGER{a(2)}
T2 ::= INTEGER{a(3),b(a)}
c T2 ::= b
d T2 ::= a
```

其中，c 指示值 1，由于它既不能引用 a 的第二、也不能第三个出现，而 d 指示值 3。

18.5 “NamedNumberList”中出现的每个“SignedNumber”或“DefinedValue”的值应该不同，而且代表整数类型的不同值。

18.6 在“NamedNumberList”中出现的每个“identifier”应该不同。

18.7 在“NamedNumberList”中的“NamedNumber”的顺序无关紧要。

18.8 由本记法定义的类型标记是通用类别，编号为 2。

18.9 整数类型的值应该由记法“IntegerValue”来定义，或者当用作“XMLValue”时，则用记法“XMLIntegerValue”。这些产生式是：

```
IntegerValue ::=
                SignedNumber      |
                identifier
XMLIntegerValue ::=
                SignedNumber      |
                " <"& identifier " />"
```

18.10 在“IntegerValue”及“XMLIntegerValue”中的“identifier”应是与值相关，且应表示对应的数字的“IntegerType”中的“identifier”之一。

注：当引用已定义“identifier”的整数值时，用“IntegerValue”和“XMLIntegerValue”的“identifier”形式更好。

18.11 有“NamedNumberList”的整数类型的值记法实例内，取自“NamedNumberList”的“identifier”和引用名都是名称的任何事件应该解释为“identifier”。

19 枚举类型的记法

19.1 枚举类型（见 3.6.24）应该用记法“EnumeratedType”来引用：

```
EnumeratedType ::=
                ENUMERATED " {"Enumerations " }"
```

```
Enumerations ::=
        RootEnumeration                                    |
        RootEnumeration "," "..." ExceptionSpec            |
        RootEnumeration "," "..." ExceptionSpec "," AdditionalEnumeration
RootEnumeration ::= Enumeration
AdditionalEnumeration ::= Enumeration
Enumeration ::=
        EnumerationItem | EnumerationItem "," Enumeration
EnumerationItem ::= identifier | NamedNumber
```

注 1："EnumeratedType"的每个值有一个与不同整数相关的标识符。然而，不希望值本身有任何整数语义。规定"EnumerationItem"的"NamedNumber"替换项提供控制值的表达式以便有利于可兼容的扩展。

注 2：在"RootEnumeration"中的"NamedNumber"内的数值不必有序或者相邻，而"AdditionalEnumeration"中的"NamedNumber"内的数值是有序的但不必相邻。

19.2 对于每个"NamedNumber"，"identifier"和"SignedNumber"应与"Enumeration"中的所有其他"identifier"和"SignedNumber"不同。18.2 和 18.3 条也适用于每个"NamedNumber"。

19.3 （在"EnumeratedType"中）每个是"identifier"的"EnumerationItem"连续地被赋给不同的非负整数。对于"RootEnumeration"，赋值从零开始、但不包括任何是"NamedNumber"的"EnumerationItem"中采用的任何整数的连续整数。

注：整数值与"EnumerationItem"相关，以帮助定义编码规则。否则，它没有在 ASN.1 规则中采用。

19.4 每个新"EnumerationItem"的值应该比类型中所有以前定义的"AdditionalEnumeration"要大。

19.5 当"NamedNumber"用于定义"AdditionalEnumeration"中的"EnumerationItem"时，与之相关的值应该与（本类型中）所有前面定义"EnumerationItem"的值不同，无论前面定义的"EnumerationItem"在枚举根中存在与否。例如：

```
A ::= ENUMERATED {a, b, ... , c(0)}   -- 无效，因为 a 和 c 都为 0
B ::= ENUMERATED {a, b, ... , c, d(2)} -- 无效，因为 c 和 d 都为 2
C ::= ENUMERATED {a, b(3), ... , c(1)} -- 有效，因为 c=1
D ::= ENUMERATED {a, b, ... , c(2)} -- 有效，因为 c=2
```

19.6 对于"EnumerationItem"不是在"RootEnumeration"中定义时，与作为"identifier"（而不是"NamedNumber"）的"AdditionalEnumeration"替换项中的第一个"EnumerationItem"相关的值应该是最小值，而且"AdditionalEnumeration"（若有的话）中所有前面的"EnumerationItem"较小。例如，下面都是有效的：

```
A ::= ENUMERATED {a, b, ... , c}          -- c=2
B ::= ENUMERATED {a, b, c(0), ... , d}    -- d=3
C ::= ENUMERATED {a, b, ... , c(3), d}    -- d=4
D ::= ENUMERATED {a, z(25), ... , d}      -- d=1
```

19.7 枚举类型的标记是通用类别，编号为 10。

19.8 枚举类型的值应该由记法"EnumeratedValue"定义，或者当用作"XMLValue"时，用记法"XMLEnumeratedValue"。这些产生式是：

```
EnumeratedValue ::= identifier
XMLEnumeratedValue ::= "<" & identifier "/>"
```

19.9 在"EnumeratedValue"和"XMLEnumeratedValue"中的"identifier"应该等于与值相关"EnumeratedType"序列中的"identifier"。

19.10 枚举类型值记法的实例内，取自"Enumeration"的"identifier"和引用名都是名字的任何事件应

该解释为“identifier”。

20 实数类型的记法

20.1 实数类型(见3.6.54)应该用记法“RealType”来引用:

RealType ::= REAL

20.2 实数类型的标记是通用类别,编号为9。

20.3 实数类型的值是值PLUS-INFINITY and MINUS-INFINITY及能用包含三个整数M、B和E的下列公式规定的实数:

$$M\times B^{E}$$

这里,M称作尾数,B称作底数,E称作指数。

20.4 实数类型有一个相关的类型,该类型用来精确定义实数类型的抽象值,并也用来支持实数类型的值和子类型记法。

注:编码规则可定义不同的、用来规定编码的类型,或者不引用相关类型可规定编码。特别是,如果“base”是10,PER和BER中的编码提供一个二进制代码的十进制(BCD)编码,和如果“base”是2,提供允许来自于硬件浮点表达式的有效转换的编码。

20.5 值定义和分成子类型的相关类型是(有标准注解):

```
SEQUENCE{
        mantissa INTEGER,
        mantissa INTEGER(2|10),
        exponent INTEGER
        -- 相关的数学实数是“mantissa”乘以“base”为底的“exponent”次幂
}
```

注1:尽管“base”2和“base”10表示的非零值确定同样的实数值,它们还是被认为是不同的抽象值,而且可携带不同的应用层语义。

注2:记法“REAL(WITH COMPONENT{..., base (10)})”能用来限制值集为底数是10的抽象值(对底数是2的抽象值也类似)。

注3:本类型能携带一个能以典型浮点硬件储存的任何数字的准确有限表达式及任何有限的十进制字符表达式数字。

20.6 实数类型的值应该用记法“RealValue”来定义,或者用于“XMLValue”时,用记法“XML RealValue”定义:

```
RealValue ::=
        NumericRealValue        |
        SpecialRealValue
NumericRealValue::=
        realnumber              |
        "-" realnumber          |
        SequenceValue -- 相关序列类型的值
SpecialRealValue ::=
        PLUS-INFINITY           |
        MINUS-INFINITY
```

“NumericRealValue”的第二和第三个替换项不应用于零值。

```
XMLRealValue::=
        XMLNumericRealValue |XMLSpecialRealValue
XMLNumericRealValue::=
```

realnumber |
" - "realnumber

"XMLNumericRealValue"的第二个替换项不应该用作零值：

XMLSpecialRealValue ::=
" < " &PLUS-INFINITY " /> " | " < " MINUS-INFINITY " /> "

20.7 当使用"realnumber"记法时，它标识出对应以 10 为"base"的抽象值。如果"RealType"被约束为以 2 为"base"，"realnumber" 以 2 为"base"的抽象值标识对应"realnumber"规定的十进制值，或如果不能准确表示时对应本机系统定义的精确度。

21 位串类型的记法

21.1 位串类型(见 3.6.6)应该用记法"BitStringType"来引用：

BitStringType ::=
BIT STRING
BIT STRING " { "NamedBitList " } "
NamedBitList ::=
NamedBit |
NamedBitList " , " NamedBit
NamedBit ::=
identifier " ("number ")" |
identifier " ("DefinedValue ") " |

21.2 位串中的第一位叫做起始位，位串中的最后一位叫做结尾位。

注：这一术语用于规定值记法及定义编码规则。

21.3 "DefinedValue"应该引用类型整数的非负值。

21.4 出现在"NamedBitList"中的每个"number"或"DefinedValue"的值应该不同，而且是位串值中的不同位的数字。位串的起始位用"number"零标识，连续的位上有连续的值。

21.5 出现在"NamedBitList"中的每个"identifier"应该不同。

注 1：在"NamedBitList"中的"NamedBit"产生式序列的顺序没有意义。

注 2：由于出现在"NamedBitList"内的"identifier"不能用来规定与"NamedBit"相关的值，"DefinedValue"从不可能被误以为是"IntegerValue"。因此在下列情况下：

a INTEGER ::= 1
T1 ::= INTEGER {a(2)}
T2 ::= BIT STRING {a(3),b(a)}

其中 a 的最后事件指明值 1，因为它既不能引用 a 的第二个也不能引用第三个事件。

21.6 "NamedBitList"的出现不会影响本类型的抽象值的集。允许是包含 1 位而非已命名位的值。

21.7 当"NamedBitList"用来定义位串类型时，ASN.1 编码规则随意地在被编码或解码的值上加上(或移去)任意多个结尾 0 位。因此，应用设计者应保证不同的语义与这些仅在结尾 0 位不同的值不相关。

21.8 该类型的标记是通用类别，编号为 3。

21.9 位串类型的值应该用记法"BitStringValue"来定义，或者当用作"XMLValue"时，用记法"XML-BitStringValue"。这些产生式是：

BitStringValue ::=
bstring |
hstring |
"{ "IdentifierList "}" |

```
        "{" "}"                         |
        CONTAINING Value
IdentifierList ::=
        identifier                      |
        IdentifierList "," identifier
XMLBitStringValue ::=
        XMLTypedValue                   |
        Xmlbstring                      |
        XMLIdentifierList               |
        empty
XMLIdentifierList::=
        "<" &identifier "/>"            |
        XMLIdentifierList " < " &identifier  " /> "
```

21.10 除非位串有包含 ASN.1 类型但不包括 ENCODED BY 的内容约束，否则，不应该使用“XMLTypedValue”替换项。如果使用这一替换项，则“XMLTypedValue”应该是内容约束中的 ASN.1 类型的值。

21.11 除非位串有“NamedBitList”，否则不应使用“XMLIdentifierList”替换项。

21.12 在“BitStringValue”或“XMLBitStringValue”中的每个“identifier”应该和与该值相关的“BitStringType”产生式序列中的“identifier”相同。

21.13 “empty”替换项指明带空位的位串。

21.14 如果位串有已命名位，则“BitStringValue”或“XMLBitStringValue”记法用对应“identifier”的数字规定的位的位置中的值指明位串值，而且，所有其他位是零。

注：对有“NamedBitList”的“BitStringType”，“BitStringValue”中的“{" "}”产生式序列和“XMLBitStringValue”中的“empty”用来指明没有包含任何位的位串。

21.15 当使用“bstring”或“xmlbstring”记法时，位串值的起始位在左边，而位串值的结束位在右边。

21.16 当使用“hstring”记法时，每个十六进制数字的最高有效位对应位串中最左边的位。

注：这一记法在任何情况下都不约束解码规则将位串置成用来传送 8 位位组的方式。

21.17 除非位串值由 4 位的倍数组成，否则不应该使用“hstring”记法。

例如：

'A98A'H

和

'1010100110001010'B

是同一位串值的替换记法。如果用“NamedBitList”定义类型，则(单个)结尾零不能形成长度是 15 位的值的一部分。如果没有用“NamedBitList”定义类型，则结尾零形成长度是 16 位的值的一部分。

21.18 如果具有包括 CONTAINING 的位串类型内容约束，则只能使用 CONTAINING 替换项。然后，“Value”应该是“ContentsConstraint”中“Type”的值的值记法(GB/T 16262.3—2006 第 11 章)。

注：由于 GB/T 16262.3—2006 的 11.3 条禁止“ContentsConstraint”之后的进一步约束，因此，该值记法从不可能在子类型约束中出现，而且，除非支配者有“ContentsConstraint”，否则，上面的文本禁止其使用。

21.19 如果有不包含 ENCODED BY 的位串类型的内容约束，应该使用 CONTAINING 替换项。

22 八位位组串类型的记法

22.1 八位位组串类型(见 3.6.49)应由记法“八位位组串类型”引用：

```
OctetStringType ::= OCTET STRING
```

22.2 该类型的标记是通用类别，编号为 4。

22.3 八位位组串类型的值应该用记法“OctetStringValue”定义，或者当用作“XMLValue”时，用记法“XML OctetStringValue”，这些产生式是：

```
OctetStringValue ::=
        bstring                 |
        hstring                 |
        CONTAINING Value
XMLOctetStringValue ::=
        XMLTypedValue           |
        xmlhstring
```

22.4 除非八位位组串有包含 ASN.1 类型但不包括 ENCODED BY 的内容约束，否则，不应该使用“XMLTypedValue”替换项。如果使用这一替换项，则“XMLTypedValue”应该是内容约束中的 ASN.1 类型的值。

22.5 在规定八位位组串的编码规则时，用术语首位八位位组和结尾位八位位组引用八位位组，而对八位位组中的位用术语最高有效位和最低有效位引用。

22.6 当使用“bstring”记法时，“bstring”记法最左位应该是八位位组串值的首位八位位组的最高有效位。若“bstring”不是八位的整数倍时，应理解为好像它含有附加零结束位，以形成下一个八的整数倍。

22.7 当使用“hstring”或“xmlhstring”记法时，最左的十六进制数字应该是第一个八位位组的最高有效半八位位组。

22.8 如果“hstring”是十六进制数字奇数，应理解为好像它含有单个附加结束位零十六进制数字。“xmlhstring”不应该是十六进制数字奇数。

22.9 如果有包括 CONTAINING 的八位位组串类型内容约束，就只能使用 CONTAINING 替换项。然后，“Value”应该是“ContentsConstraint”中“Type”的值的值记法（见 GB/T 16262.3—2006 第 11 章）。

注：由于 GB/T 16262.3—2006 的 11.3 条禁止“ContentsConstraint”之后的进一步约束，因此，这一值记法从不可能在子类型约束中出现，而且，除非支配者有“ContentsConstraint”，否则，上面的文本禁止其使用。

22.10 如果有不包含 ENCODED BY 的八位位组串类型的内容约束，应该使用 CONTAINING 替换项。

23 空类型记法

23.1 空类型（见 3.6.44）应该用记法“NullType”来引用：

```
NullType ::=  NULL
```

23.2 该类型的标记是通用类别，编号为 5。

23.3 空类型的值由记法“NullValue”引用，或者当用作“XMLValue”时，用记法“XMLNullValue”。这些产生式是：

```
NullValue ::= NULL
XMLNullValue ::= empty
```

24 序列类型的记法

24.1 定义序列类型（见 3.6.60）的记法应该是“SequenceType”：

```
SequenceType::=
        SEQUENCE  "{"  "}"                                  |
        SEQUENCE "{" ExtensionAndException OptionalExtensionMarker "}"  |
        SEQUENCE "{" ComponentTypeLists "}"
```

```
ExtensionAndException ::=  "..." | "..." ExceptionSpec
OptionalExtensionMarker ::= "," "..." | empty
ComponentTypeLists::=
        RootComponentTypeList                    |
        RootComponentTypeList "," ExtensionAndException ExtensionAdditions
            OptionalExtensionMarker              |
        RootComponentTypeList "," ExtensionAndException ExtensionAdditions
            ExtensionEndMarker " , " RootComponentTypeList   |
        ExtensionAndException ExtensionAdditions ExtensionEndMarker " , "
            RootComponentTypeList                |
        ExtensionAndException ExtensionAdditions OptionalExtensionMarker
RootComponentTypeList::= ComponentTypeList
ExtensionEndMarker ::= "," "..."
ExtensionAdditions ::=
        " , " ExtensionAdditionList              |
        empty
ExtensionAdditionList ::=
        ExtensionAddition                        |
        ExtensionAdditionList "," ExtensionAddition
ExtensionAddition ::=
        ComponentType                            |
        ExtensionAdditionGroup
ExtensionAdditionGroup::= " [["VersionNumber ComponentTypeList "]]"
VersionNumber::= empty|number " : "
ComponentTypeList::=
        ComponentType                            |
        ComponentTypeList "," ComponentType
ComponentType::=
        NamedType                                |
        NamedType OPTIONAL                       |
        NamedType DEFAULT Value                  |
        COMPONENTS OF Type
```

24.2 当选择自动加标记的模块的定义内存在“ComponentTypeList”产生式(见 12.3),而且,“ComponentType”的前三个替换项中的任何一个的“NamedType”事件不包含“TaggedType”,那么为整个“ComponentTypeList”选择自动标记转换,否则不能这样。

注 1:为序列类型采用成分列表定义内的“TaggedType”记法把标记控制交给了规定者,正好与通过自动标记机制的自动赋值相反。因此,在下面的情况下:

```
T ::= SEQUENCE {a INTERGER, b [1] BOOLEAN, c OCTET STRING}
```

即使选择了自动标记的模块内有序列类型 T 的定义,自动标记也不适用于成分 a、b、c 列表。

注 2:只有那些选择了自动标记的模块内出现的“ComponentTypeList”产生式事件是通过自动标记转换的候选者。

24.3 单独对每个“ComponentTypeLists”事件采用自动标记转换并且是在 24.4 规定的 COMPONENTS OF 转换之前。然而,正如 24.7 至 24.9 的规定,自动标记转换(如果适用的话)应用于 COMPONENTS OF 转换之后。

注：它的作用是通过显式出现在“ComponentTypeList”中的标记抑制自动标记的应用，但是不能通过“COMPONENTS OF”之后“Type”中出现的标记来抑制。

24.4 在“COMPONENT OF Type”记法中的“Type”应该是序列类型。“COMPONENT OF Type”记法应该用来定义(在成分列表的这点上)引用类型的所有成分类型的内容，可在“Type”中出现的任何扩展标记和扩展附加除外。(只包含“COMPONENT OF Type”中“Type”的“RootComponentTypeList”；“COMPONENT OF Type”记法忽略了扩展标志和扩展附加(如果有的话))。该转换忽略了作用于被引用类型的任何子类型约束。

注：这一转换逻辑上在满足下面各条的要求之前完成。

24.5 下面各条每个都标识根或扩展附加或两者中的“ComponentType”序列事件。24.5.1 的规则应该适于所有这些序列。

24.5.1 当有一个或多个都标有 OPTIONAL 或 DEFAULT 的连续的“ComponentType”事件时，这些“ComponentType”的标记及系列中任何紧接的成分类型的标记应该不同(见第 30 章)。如果选择自动标记，标记不同的要求仅适用于自动标记完成之后，而且，将总能满足。

24.5.2 第 24.5.1 条应该适用于根中的“ComponentType”序列。

24.5.3 第 24.5.1 条以类型定义中它们事件的文本顺序(忽略所有的版本括号和省略记法)，应该适用于根或扩展附加中“ComponentType”的整个序列。(也见 48.7)

24.6 当使用“ComponentTypeLists”的第三个或第四个替换项时，扩展附加中的所有“ComponentType”应该有标记，该标记与文本上紧接“ComponentType”的标记直到并包括结尾“RootComponentTypeList”中没有标志 OPTIONAL 或 DEFAULT(如果有的话)的第一个这种“ComponentType”不同。(也见 48.7)

24.7 “ComponentTypeLists”的自动置标记转换逻辑上在 24.4 规定的转换之后完成，但是只有 24.2 确定它应该适用于哪个“ComponentTypeLists”事件。自动标记转换通过用 24.9 中规定的替代项“TaggedType”事件代替“NamedType”产生式中原有的“Type”来影响“ComponentTypeLists”的每个“ComponentType”。

24.8 如果自动置标记有效而且扩展根中的“ComponentType”没有标记，那么，“ExtensionAdditionList”内的“ComponentType”不应该是“TaggedType”。

24.9 如果自动置标记有效，“TaggedType”的替代项规定如下：

a) 替代“TaggedType”记法使用“Tag Type”替换项；

b) 替代“TaggedType”的“Class”是空(即：置标记是上下文规定)；

c) 替代“TaggedType”中的“ClassNumber”对于“RootComponentTypeList”中的第一个“ComponentType”是标记值零，第二个为 1，以此类推，按递增标记数字继续下去；

d) 如果“RootComponentTypeList”丢失，“ExtensionAdditionList”中的第一个“ComponentType”的替代“TaggedType”中的“ClassNumber”是零，否则，就是比“RootComponentTypeList”中的最大“ClassNumber”大 1，而“ExtensionAdditionList”中的下一个“ComponentType”的“ClassNumber”比第一个大 1，以此类推，按递增标记数字继续进行。

e) 替代“TaggedType”中的“Type”是被替代的原有“Type”。

注 1：由 30.6 提供了支配替代“TaggedType”的隐式置标记或显式式置标记的规范的规则。自动置标记总是隐式置标记，除非“Type”是选择类型或者开放类型记法或者是“DummyReference”(见 GB/T 16262.4 的 8.3)，这时，它是显式置标记。

注 2：一旦 24.7 满足，成分的标记就完全确定了，而且不能修改。即使序列类型在应用自动置标记转换的另一个“ComponentTypeList”内成分的定义中被引用。因此，在下面的情况下：

```
T ::= SEQUENCE { a  Ta,  b  Tb,  c  Tc }
E ::= SEQUENCE { f1  E1,  f2  T,  f3  E3 }
```

作用于 E 的成分的自动置标记决不会影响附加在 T 的成分 a、b 和 c 上的标记，而不管 T 的置标记环境如何。如

果 T 是自动置标记环境中定义的，而 E 不是在自动置标记环境中，则自动置标记仍然适用于 T 的成分 a、b 和 c。

注 3：当序列类型作为“Type”出现在“COMPONENT OF Type”中时，其中的每个“ComponentType”事件由采用 24.4 在可能将自动置标记应用于引用序列类型之前复制。因此，在下面的情况下：

```
T ::= SEQUENCE { a  Ta,  b  SEQUENCE { b1  T1,  b2  T2,  b3  T3 },  c  Tc }
W ::= SEQUENCE { x  Wx,  COMPONENTS OF  T,  y  Wy }
```

如果 W 是在自动置标记的环境中定义的，T 中的 a、b 和 c 的标记不必与 W 中的 a、b 和 c 的标记相同，但是 b1、b2 和 b3 的标记在 T 和 W 中都一样。换句话说，自动置标记转换只能在给出的“ComponentTypeLists”上应用一次。

注 4：分成子类型不影响自动置标记。

注 5：当自动置标记在某一位置时，在扩展插入点(见 3.6.29)以外的任意位置插入新成分，由于改变标记的副作用而引起与规范旧版本互工作的问题，可能导致其他成分变化。

24.10 如果出现“OPTIONAL”或者“DEFAULT”，对应的值可以从新类型的值中忽略。

24.11 如果“DEFAULT”存在，省略那个类型的值准确地相当于插入由“Value”定义的值，该“Value”应该是“NamedType”产生式序列中的由“Type”定义的类型值的值记法。

24.12 对应于“ExtensionAdditionGroup”(全部成分一起)的值是可选的。然而，如果出现这样的值，那么，应该出现对应于“ComponentTypeList”内没有标有 OPTIONAL 或 DEFAULT 的成分的值。

24.13 在“ComponentTypeList”(与那些通过 COMPONENT OF 扩展获得的一起)的所有“NamedType”产生式序列中的“identifier”应该都是不同的。

24.14 除非具有为逻辑上位于扩展附加类型与扩展根之间的没有标志的 OPTIONAL 或 DEFAULT 的所有扩展附加类型所规定的值，否则，不应规定给定扩展附加类型的值。

注 1：当类型通过附加扩展附加从扩展根(第一版)经第二版发展到第三版时，第三版任何附加的编码中的出现要求没有标志 OPTIONAL 或 DEFAULT 的第二版中所有附加的编码的出现。

注 2：如果没有标志 OPTIONAL 或 DEFAULT，本身是扩展类型但不包含在“ExtensionAdditionGroup”内的“ComponentType”应该总是要被编码，抽象值正从用没有定义“ComponentType”的抽象语法早期版本的发送者转送除外。

注 3：由于下面的原因，推荐使用“ExtensionAdditionGroup”产生式：

a) 它能根据编码规则产生更简洁的编码(如，PER)。

b) 语法更精确，因为它清楚地指明如果本身被定义的扩展附加组被编码(和注 1 比较)，“ExtensionAdditionGroup”中定义的和没有标志 OPTIONAL 或 DEFAULT 的类型的值总出现在编码中。

c) 语法使“ExtensionAdditionGroup”中哪个类型必须作为组被应用层支持更清楚。

24.15 只有模块内的所有“ExtensionAddition”和“ExtensionAdditionAlternatives”是带“VersionNumber”的“ExtensionAdditionGroup”或“ExtensionAdditionAlternativesGroup”，才应该使用“VersionNumber”。在“ExtensionAdditionGroup”的每个“VersionNumber”中的“number”应该大于或等于 2，而且应该比插入点内任何前面的“ExtensionAdditionGroup”中的“number”大。

注 1：这里使用的惯例是没有扩展附加组的规范是版本 1，因此，第一个附加的扩展附加组将有大于或等于 2 的数字。为“ExtensionAdditions”需要单个“ExtensionAddition”时，“ExtensionAdditionGroup”可以和“ExtensionAddition”一起使用。

注 2：使用“VersionNumber”的约束仅适用于单个模块内而且不强加约束给引入的类型。

24.16 所有序列类型的标记都是通用类，编号为 16。

注：单一序列类型有和序列类型(见 25.2)一样的标记。

24.17 定义一个序列类型的值的记法应该是“SequenceValue”，或者当用作“XMLValue”时是“XMLSequenceValue”。这些产生式是：

```
SequenceValue ::=
        "{" ComponentValueList"}"        |
        "{"  "}"
```

```
ComponentValueList ::=
        NamedValue|
        ComponentValueList "," NamedValue
XMLSequenceValue ::=
        XMLComponentValueList          |
        empty
XMLComponentValueList ::=
        XMLNamedValue|
        XMLComponentValueList XMLNamedValue
```

24.18 "{" "}"或"empty"记法只用于下列条件下：

a) 在"SequenceType"中的所有"ComponentType"序列被标记为 DEFAULT 或 OPTIONAL，而且忽略所有的值；或者

b) 类型记法是 SEQUENCE{}。

24.19 没有标志为 OPTIONAL 或 DEFAULT 的"SequenceType"中的每个"NamedType"应该有一个"NamedValue"或"XML NamedValue"，而且，值应该与对应"NamedType"序列的顺序一样。

25 单一序列类型的记法

25.1 由其他类型定义单一序列类型(见 3.6.61)的记法应该是"SequenceOfType"。

```
SequenceOfType ::= SEQUENCE OF Type | SEQUENCE OF NamedType
```

注：如果用于"SequenceOfType"XML 值记法的 XML 标记名称需要大写的起始字母，那么应该采用第一个替换项。(然后，XML 标记名称从"Type"的名称形成。)

25.2 所有单一序列类型的标记都是通用类别，编号为 16。

注：序列类型有与单一序列类型(见 24.15)一样的标记。

25.3 用于定义单一序列类型值的记法是"SequenceOfValue"，或者当用作"XMLValue"时，是"XML SequenceOfValue"。这些产生式是：

```
SequenceOfValue ::=
        " {"ValueList " }"             |
        " {" NamedValueList "}"        |
        "{" "}"
ValueList ::=
        Value                          |
        ValueList "," Value
NamedValueList::=
        NamedValue                     |
        NamedValueList "," NamedValue
XMLSequenceOfValue::=
        XMLValueList                   |
        XMLDelimitedItemList           |
        XMLSpaceSeparatedList          |
        empty
XMLValueList::=
        XMLValueOrEmpty                |
        XMLValueOrEmpty XMLValueList
```

```
XMLValueOrEmpty::=
        XMLValue                                    |
        " <"& NonParameterizedTypeName " />"
XMLSpaceSeparatedList::=
        XMLValueOrEmpty                             |
        XMLValueOrEmpty " " XMLSpaceSeparatedList
XMLDelimitedItemList::=
        XMLDelimitedItem                            |
        XMLDelimitedItem XMLDelimitedItemList
XMLDelimitedItem::=
        " <"& NonParameterizedTypeName " >"XMLValue
            " </" & NonParameterizedTypeName " >"   |
        " <" & identifier" >"XMLValue " </" & identifier " >"
```

"{" "}"或"Empty"记法用于"SequenceOfValu"或"XMLSequenceOfValu"是空列表时。

注 1：语法意义可能按这些值的顺序放置。

注 2："XMLSpaceSeparatedList"产生式不用于本部分，而且不用于 XML 值记法。提供它是为了允许"IntegerType"、"RealType"、"ObjectIdentifierType"、"RelativeOIDType"和 GeneralizedTime 及 UTCTime 有用类型的编码中采用"XMLSpaceSeparatedList"的规范。也可能为某些 SEQUENCE OF SEQUENCE 和 SEQUENCE OF SET 实例规定使用"XMLValueList"代替"XMLDelimitedItemList"。

25.4 如果成分的"XMLValue"是"empty"，那么，应该选择"XMLValueOrEmpty"的第二个替换项代表成分的那个值。

25.5 应该按照表 5 的第 2 列使用"XMLValueList"或"XMLDelimitedItemList"产生式，其中，成分的"Type"在第 1 列中列出。

表 5 ASN.1 类型的"XMLSequenceOfValue"和"XMLSetOfValue"记法

ASN.1 类型	XML 值记法
BitStringType	XMLDelimitedItemList
BooleanType	XMLValueList
CharaterStringType	XMLDelimitedItemList
ChoiceType	XMLValueList
ConstrainedType	见 25.7
DefinedType	见 25.9
EmbeddedPDVType	XMLDelimitedItemList
EnumeratedType	XMLValueList
ExternalType	XMLDelimitedItemList
InstanceOfType	见 GB/T 16262.2 的 C.9
IntegerType	XMLDelimitedItemList
NullType	XMLValueList
ObjectClassFieldType	见 GB/T 16262.2 的 14.10 和 14.11
ObjectIdentifierType	XMLDelimitedItemList
OctetStringType	XMLDelimitedItemList

表 5(续)

ASN.1 类型	XML 值记法
RealType	XMLDelimitedItemList
RelativeOIDType	XMLDelimitedItemList
SelectionType	见 25.8
SequenceType	XMLDelimitedItemList
SequenceOfType	XMLDelimitedItemList
SetType	XMLDelimitedItemList
SetOfType	XMLDelimitedItemList
TaggedType	见 25.6
UsefulType(GeneralizedTime)	XMLDelimitedItemList
UsefulType(UTCTime)	XMLDelimitedItemList
UsefuType(ObjectDescriptor)	XMLDelimitedItemList
TypeFromObject	见 GB/T 16262.2 的 15.6
ValueSetFromObjects	见 GB/T 16262.2 的 15.6

25.6 如果成分的“Type”是“TaggedType”,那么确定“XMLSequenceOfValue”记法的类型应该是“TaggedType”中的“Type”(见 30.1)。如果它本身是一个“TaggedType”,那么本条应该递归地应用。

25.7 如果成分的“Type”是“ConstrainedType”,那么确定“XMLSequenceOfValue”记法的类型应该是“ConstrainedType”中的“Type”(见 45.1)。如果它本身是一个“ConstrainedType”,那么本条应该递归地应用。

25.8 如果成分的“Type”是“SelectionType”,那么确定“XMLSequenceOfValue”记法的类型应该是“SelectionType”引用的“Type”(见第 29 章)。

25.9 如果成分的“Type”是“DefinedType”,那么确定“XMLSequenceOfValue”记法的类型应该是“DefinedType”引用的“Type”(见 13.1)。

25.10 如果且只有“SequenceOfType”包含“identifier”,应该使用“XMLDelimitedItem”的第二个替换项,而且,“XMLDelimitedItem”中的“identifier”应该是那个“identifier”。

25.11 如果使用“XMLDelimitedItem”的第一个替换项,那么,如果单一序列类型的成分(忽略任何标记之后)是“typereference”或“ExternalTypeReference”,那么,“NonParameterizedTypeName”应该是那个“typereference”或“ExternalTypeReference”,否则,它应该是对应成分固有类型的表 4 中规定的“xmlasn1typename”。

25.12 如果使用“SequenceOfType”的第一个替换项,那么,应该使用“SequenceOfValue”的第一个替换项。“SequenceOfValue”的“ValueList”中的每个“Value”,以及“XMLSequenceOfValue”替换项中的每个“XMLValue”应该是“SequenceOfType”中规定的类型的。

25.13 如果使用“SequenceOfType”的第二个替换项,那么,应该使用“SequenceOfValue”的第二个替换项,而且“NamedValueList”中的每个“NamedValue”应该包含“SequenceOfType”的“NamedType”规定的类型“Value”。“NamedValue”中的“identifier”应该是“SequenceOfType”的“NamedType”中的“identifier”。

26 集合类型的记法

26.1 由其他类型定义集合类型(见 3.6.64)的记法应该是“SetType”。

SetType ::=

```
        SET " {" " }"                    |
        SET " {"ExtensionAndException OptioanalExtensionMarker " }"   |
        SET " {"ComponentTypeLists " }"
```

"ComponentTypeLists"、"ExtensionAndException"及"OptioanalExtensionMarke"在 24.1 中规定。

26.2 在"COMPONENT OF Type"记法中的"Type"应该是集合类型。"COMPONENTS OF Type"记法应该用来定义成分列表的这一点上引用类型的所有成分类型的内容,可能出现在"Type"中的任何扩展标志和扩展附加除外。(只包含"COMPONENTS OF Type"中"Type"的"RootComponentTypeList","COMPONENTS OF Type"记法忽略了扩展标志和扩展附加(如果有的话)),该转换忽略了应用于引用类型的任何子类型约束。

注:该转换逻辑上在满足下面各条的要求之前完成。

26.3 在集合类型中"ComponentType"类型应该都有不同的标记(见第 30 章)。加在"ExtensionAdditions"的每个新"ComponentType"的标记应该正则地比"ExtensionAdditions"中的其他成分的要大(见 8.6)。

注:出现该记法的模块中的"TagDefault"是 AUTOMATIC TAGS 时,不管实际"ComponentType"怎样,作为应用 24.7 的结果,实现了这一要求。(也见 48.7.)

26.4 第 24.2 和 24.7 至 24.13 条也适用于集合类型。

26.5 所有集合类型的标记都是通用类别,编号为 17。

注:单一集合类型有和集合类型(见 27.2)一样的标记。

26.6 语义与集合类型中值的顺序无关。

26.7 定义集合类型值的记法应该是"SetValue",或者当用作"XMLValue"时是"XML SetValue"。这些产生式是:

```
SetValue ::= "{" ComponentValueList " }"      |
        " {" " }"
XMLSetValue ::=
        XMLComponentValueList                 |
        empty
```

在 24.17 中规定了"ComponentValueList"和"XMLComponentValueList"。

26.8 "SetValue"和"XMLSetValue"只应该分别是"{" "}"和"empty",如果:

a) 在"SetType"中的所有"ComponentType"序列标为"DEFAULT"或"OPTIONAL",且所有的值都被省略;或者

b) 类型记法是 SET {}。

26.9 形码未标有 DEFAULT 或 OPTIONAL 的"SetType"中的每个"NamedType"应该有一个"NamedValue"或"XMLNamedValue"。

注:这些"NamedValue"或"XMLNamedValue"可以任意顺序出现。

27 单一集合类型的记法

27.1 由其他类型定义单一集合类型(见 3.6.65)的记法应该是"SetOfType"。

```
SetOfType ::=
        SET OF Type          |
        SET OF NamedType
```

注:如果用于"SetOfType"的 XML 值记法的 XML 标记名称需要大写的起始字母,那么应该采用第一个替换项。(然后,XML 标记名称从"Type"的名称形成。)

27.2 所有单一集合类型的标记都是通用类别,编号为 17。

注:集合类型有与单一集合类型(见 26.5)一样的标记。

27.3 用于定义单一集合类型值的记法应该是“SetOfValue”，或者用作“XMLValue”时是“XML SetOfValue”。这些产生式是：

```
SetOfValue ::=
        " {"ValueList " }"          |
        " {" NamedValueList " }"    |
        " {" " }"
XMLSetOfValue ::=
        XMLValueList                |
        XMLDelimitedItemList        |
        XMLSpaceSeparatedList       |
        empty
```

在25.3中规定了“ValueList”、“NamedValueList”和“XMLSetOfValue”的替换项。“{" "}”或“empty”记法用在“SetOfValue”或“XMLSetOfValue”是空列表时。

注1：语义重要性不应该按照这些值的次序给出。

注2：不要求编码规则来保持这些值的顺序。

注3：单一集合类型不是值的数学集合，因此，作为示例，对于SET OF INTEGER，值{1}和{1 1}是不同的。

27.4 如果使用“SetOfType”的第一个替换项，那么，应该使用“SetOfValue”的第一个替换项。“SetOfValue”的“ValueList”中的每个“Value”，以及“XMLSetOfValue”替换项中的每个“XMLValue”应该是“SetOfType”中规定的类型的。

27.5 如果使用“SetOfType”的第二个替换项，那么，应该使用“SetOfValue”的第二个替换项，而且，“NamedValueList”中的每个“NamedValue”序列应该包含“SetOfType”的“NamedType”中规定的类型“Value”。“NamedValue”中的“identifier”应该是“SetOfType”的“NamedType”中的“identifier”。

28 选择类型的记法

28.1 由其他类型定义选择类型(见3.6.13)的记法应该是“ChoiceType”。

```
ChoiceType ::= CHOICE"{"AlternativeTypeList " }"
AlternativeTypeList ::=
        RootAlternativeTypeList     |
        RootAlternativeTypeList ","
            ExtensionAndException ExtensionAdditionAlternatives
            OptionalExtensionMarker
RootAlternativeTypeList ::= AlternativeTypeList
ExtensionAdditionAlternatives ::=
        "," ExtensionAdditionAlternativeList   |
        empty
ExtensionAdditionAlternativesList ::=
        ExtensionAdditionAlternative|
        ExtensionAdditionAlternativesList "," ExtensionAdditionAlternative
ExtensionAdditionAlternative ::=
        ExtensionAdditionAlternativesGroup |
        NamedType
ExtensionAdditionAlternativeGroup ::= " [[" VersionNumber AlternativeTypeList " ]] "
AlternativeTypeList ::=
```

NamedType |

AlternativeTypeList " , " NamedType

注："T::= CHOICE{a A}"，但 A 不是同一个类型，而且根据编码规则可能被不同地编码。

28.2 当选择自动置标记的模块的定义内存在"AlternativeTypeList"产生式时(见 12.3)，且任何"AlternativeTypeList"中的"NamedType"都不包含"TaggedType"，则为整个"AlternativeTypeLists"选择自动置标记转换，否则不能这样。

28.3 在"AlternativeTypeLists"内"AlternativeTypeList"产生式中定义的类型应该有不同的标记(见第 30 章和 48.7)。如果选择自动置标记，标记不同的要求仅适用于自动置标记完成之后，而且应总是满足的。

28.4 如果自动置标记有效而且在扩展根中的"NamedType"没有标记，那么，"ExtensionAdditionAlternativesList"内的"NamedType"都不应该是有已标记类型。

28.5 自动置标记转换用替代"TaggedType"代替"NamedType"产生式中的原有"Type"影响"AlternativeTypeLists"的每个"NamedType"。替代"TaggedType"规定如下：

a) 替代"TaggedType"记法使用"Tag Type"替换项；

b) 替代"TaggedType"的"Class"是空(如置标记是上下文规定的)；

c) 替代"TaggedType"中的"ClassNumber"对"RootAlternativeTypeList"中的第一个"NamedType"是标记值零，第二个是 1，以此类推，以递增标记数字继续；

d) 在"ExtensionAdditionAlternativesList"中的第一个"NamedType"的替代"TaggedType"中的"ClassNumber"比"RootAlternativeTypeList"中最大的"ClassNumber"大 1，"ExtensionAdditionAlternativesList"中的下一个"NamedType"的"ClassNumber"比第一个大 1，以此类推，以递增标记数字继续；

e) 替代"TaggedType"中的"Type"是被代替的原有"Type"。

注 1：由 30.6 提供了支配为替代"TaggedType"的隐式置标记或显式置标记规范的规则。自动置标记总是隐式置标记，除非"Type"是无标记的值择类型或无标记的开放类型记法，或无标记的"DummyReference"(见 GB/T 16262.4—2006 的 8.3)，这种情况下它是显式置标记。

注 2：一旦采用自动置标记，成分的标记完全确定了，而且即使在使用自动置标记转换的另一个"AlternativeTypeLists"内的替换项的定义中引用选择类型时也不能修改。因此，在下面的情况下：

T::=CHOICE { a Ta, b Tb, c Tc}

E::=CHOICE {f1 E1, f2 T, f3 E3}

应用于 E 成分的自动置标记决不会影响附加在 T 的成分 a、b 和 c 的标记，不管 T 的置标记环境如何。如果 T 在自动置标记环境中定义，而 E 不是在自动置标记环境中，自动置标记仍然适用于 T 的成分 a、b 和 c。

注 3：分成子类型不影响自动置标记。

注 4：当自动动置标记在某一位置时，在非扩展插入点(见 3.6.29)的任意位置插入新替换项，由于改变标记的副作用而引起与规范旧版本互工作的问题可能导致其他替换项变化。

28.6 在 24.1 中定义了"VersionNumber"，而且，在 24.15 中规定的整个模块中限制一致使用"VersionNumber"应该应用于本产生式中的"number"。

28.7 加在"ExtensionAdditionAlternativesList"上的每个新"NamedType"的标记应该正则地比"ExtensionAdditionAlternativesList"中其他替换项的那些大(见 8.6)，而且，应该是"ExtensionAdditionAlternativesList"中的最后一个"NamedType"。

28.8 选择类型包含标记不都相同的值。(标记依赖于将值贡献给了选择类型的替换项。)

28.9 当这一类型没有扩展标志且用在本部分要求使用带不同标记的类型的地方时(见 28.3)，选择类型的值的所有可能标记应该认为在这个要求内。假设"TagDefault"不是 AUTOMATIC TAGS 的下面示例说明了这一要求。

示例：

```
1    A ::= CHOICE{
            b          B,
            c          NULL}
     B ::= CHOICE{
            d          [0]NULL,
            e          [1]NULL}
2    A ::=CHOICE{
            b          B,
            c          C}
     B ::=CHOICE{
            d          [0]NULL,
            e          [1]NULL}
     C ::=CHOICE{
            f          [2]NULL,
            g          [3]NULL}
3 (INCORRECT)
     A ::=CHOICE{
            b          B,
            c          C}
     B ::=CHOICE{
            d          [0]NULL,
            e          [1]NULL}
     C ::=CHOICE{
            f          [0]NULL,
            g          [1]NULL}
```

例 1 和例 2 是记法的正确用法。例 3 没有用自动置标记是不正确的，其中类型 d 和 f 的标记是相同的，同时类型 e 和 g 的标记也是相同的。

28.10 在“AlternativeTypeLists”中所有“NamedType”的“identifier”应该与那个表中其他“NamedType”的不同。

28.11 定义选择类型值的记法应该是“ChoiceValue”，或者当用作“XMLValue”时是“XMLChoiceValue”。这些产生式是：

```
ChoiceValue ::= identifier ":" Value
XMLChoiceValue::= "<"& identifier ">"XMLValue "</"& identifier">"
```

28.12 “Value”或“XMLValue”应该是由“identifier”命名的“AlternativeTypeLists”中类型的值的记法。

29 精选类型的记法

29.1 定义精选类型(见 3.6.59)的记法应该是“SelectionType”。

```
SelectionType ::= identifier "<" Type
```

这里，“Type”指明选择类型，而“identifier”是出现在那个选择类型定义的“AlternativeTypeLists”中的某些“NamedType”的。

29.2 当“Type”指明受约束类型时，精选在双亲类型上完成，忽略双亲类型上的任何子类型约束。

29.3 在“SelectionType”用作“NamedType”时，出现“NamedType”的“identifier”，以及“SelectionT-

ype"的"identifier"。

29.4 在"SelectionType"用作"Type"时,保留"identifier",而且指明的类型是被选择替换项的类型。

29.5 精选类型的值的记法应该是"SelectionType"引用的类型的值的记法。

30 已标记类型的记法

已标记类型(见3.6.70)是和旧类型同形的新类型,但是具有不同的标记。已标记类型主要用在本部分要求使用有不同标记的类型(见24.5至24.6、26.3和28.2)之处。使用模块中AUTOMATIC TAGS的"TagDefault"允许这一切不用那个模块中已标记类型记法的显式出现完成。

注:协议确定取自几个数据类型的值可随时及时传播时,可能需要不同的标记使接收者能正确地对这些值解码。

30.1 已标记类型的记法应该是"NamedType":

```
NamedType ::=
        Tag Type              |
        Tag IMPLICIT Type     |
        Tag EXPLICIT Type
Tag ::= " [" Class ClassNumber "]"
ClassNumber::=
        Number                |
        DefiniedValue
Class::=
        UNIVERSAL             |
        APPLICATION           |
        PRIVATE               |
        empty
```

30.2 在"DefiniedValue"中的"valuereference"应该是类型整数的,并且指派了非负值。

30.3 新类型与旧类型同形,但是带有以"Class"为类别和以"ClassNumber"为序号的标记,除非"Class"是"empty",这时,标记是上下文规定的类别,而序号是"ClassNumber"。

30.4 除本部分定义的类型外,"Class"不应该是"UNIVERSAL"。

注1:对通用类别标记的使用,ITU-T和ISO会经常协商。

注2:E.2.22条包含有关使用标记类别格式的指导和提示。

30.5 标记的所有应用是隐式置标记或显式置标记。对于那些提供选择的编码规则,隐式置标记指明:在传送期间不需要显式标识"TaggedType"中的"Type"的原始标记。

注:当是通用类别时,保留旧标记可能有用,因此,不用新类型的ASN.1定义的知识无歧义地标识旧类型。然而,通常用IMPLICIT实现最小传送八位位组。在GB/T 16263.1中给出了用IMPLICIT编码的示例。

30.6 如果下述任何一项成立,置标记构造规定显式置标记:

a) 使用"Tag EXPLICIT Type"替换项;

b) 使用"Tag Type"替换项,而且模块的"TagDefault"值是EXPLICIT TAGS或者是空;

c) 使用"Tag Type"替换项,而且模块的"TagDefault"值是IMPLICIT TAGS或者AUTOMATIC TAGS,但是,"Type"定义的类型是无已标记的选择类型、无已标记的开放类型或者无已标记的"DummyReference"(见GB/T 16262.4—2006的8.3)。

否则,置标记构造规定隐式置标记。

30.7 如果"Class"是"empty",使用"Tag"没有限制,而不是24.5至24.6、26.3和28.2中不同标记要求暗示的。

30.8 如果"Type"定义的类型是无标记选择类型或无标记开放类型或无标记"DummyReference"(见

GB/T 16262.4—2006 的 8.3),则不应该使用 IMPLICIT 替换项。

30.9 "TaggedType"的值的记法应该是"TaggedValue",或者当用作"XMLValue"时是"XMLTaggedValue"。这些产生式是:

```
TaggedValue ::= Value
XMLTaggedValue::=XMLValue
```

这里"Value"或"XMLValue"是"TaggedValue"中"Type"的值的记法。

注:"Tag"在这个记法中不出现。

31 客体标识符类型的记法

31.1 客体标识符类型(见 3.6.48)应该用记法"ObjectIdentifierType"引用:

```
ObjectIdentifierType ::=
        OBJECT IDENTIFIER
```

31.2 该类型的标记是通用类别,编号是 6。

31.3 客体标识符的值记法应该是"ObjectIdentifierValue",或者当用作"XMLValue"时是"XMLObjectIdentifierValue"。这些产生式是:

```
ObjectIdentifierValue ::=
        " {" ObjIdComponentsList " }"     |
        " {" DefinedValue ObjIdComponentsList "}"
ObjIdComponentsList::=
        ObjIdComponents                   |
        ObjIdComponents ObjIdComponentsList
ObjIdComponents::=
        NameForm                          |
        NumberForm                        |
        NameAndNumberForm                 |
        DefinedValue
NameForm::=identifier
NumberForm::=number | DefinedValue
NameAndNumberForm::=
        identifier "(" NumberForm ")"
XMLObjectIdentifierValue::=
        XMLObjIdComponentList
XMLObjIdComponentList::=
        XMLObjIdComponent                 |
        XMLObjIdComponent& " . "&XMLObjIdComponentList
XMLObjIdComponent::=
        NameForm                          |
        XML NumberForm                    |
        XML NameAnd NumberForm
XML NumberForm::=number
XML NameAnd NumberForm::=
        identifier& "(" &XML NumberForm & " )"
```

31.4 在"NumberForm"的"DefinedValue"中的"valuereference"应该是类型整数的,并且指派非负值。

31.5 在“ObjectIdentifierValue”的“DefinedValue”中的“valuereference”应该是类型客体标识符的。

31.6 “ObjIdComponents”的“DefinedValue”应该是类型相对客体标识符的，而且应该标识从客体标识符树中的某些开始节点到客体标识符树中的某些后来节点的弧的有序集合。开始节点用较早的“ObjIdComponents”标识，而后面的“ObjIdComponents”(如果有的话)标识取自后面节点的弧。要求开始节点既不能是根、也不能是紧接根下面的节点。

注：相对客体标识符值应该与规定的客体标识符值相关以便无歧义地标识客体。要求客体标识符值(见31.10)至少有两个成分。这就是为什么要限制开始节点。

31.7 “NameForm”应该仅用于那些数字值和标识符在GB/T 17969.1的附录A至附录C(也见本部分的附录D)规定的那些客体标识符成分，而且应该是GB/T 17969.1的附录A至附录C规定的标识符之一。当GB/T 17969.1规定同义标识符时，任何同义字可以以相同的语义使用。当GB/T 17969.1中规定的标识符和包含“NameForm”的模块内的ASN.1值引用都是相同的名字时，客体标识符值内的名字应该视为GB/T 17969.1标识符。

31.8 在“NumberForm”和“XML NumberForm”中的“number”是指派给客体标识符成分的数值。

31.9 当数值赋给客体标识符成分时，应该规定“NameAndNumberForm”和“XMLNameAndNumberForm”中的“identifier”。

注：将数值分配给客体标识符成分的管理机构在GB/T 17969.1中标识。

31.10 GB/T 17969.1中规定了与客体标识符值相关的语义。

注：GB/T 17969.1要求客体标识符值应包含至少两条弧。

31.11 客体标识符成分的重要部分是简化为“NameForm”或“NumberForm”或者“XMLNumberForm”，并为客体标识符成分提供数值。除了GB/T 17969.1的附录A至附录C中(也见本部分的附录D)规定的弧之外，客体标识符成分的数值总是出现在客体标识符值记法实例中。

31.12 在“ObjectIdentifierValue”包含客体标识符值的“DefinedValue”时，它引用的客体标识符成分列表作为前缀加在值中显式出现的成分上。

注：GB/T 17969.1建议只要指派客体标识符值来标识客体，也指派了客体描述符值。

例如，

按GB/T 17969.1规定赋值的标识符，值：

```
{iso  standard  8571  pci(1)}
```

和

```
{1  0  8571  1}
```

将各标识一个客体，ISO 8571中定义的“pci”。正如“XML ObjectIdentifierValue”中的

```
iso. standard. 8571. pci(1)
```

和

```
1.0.8571.1
```

一样。

随着下面的附加定义：

```
ftam OBJECT IDENTIFIER ::={iso standard 8571}
```

下面的值相当于上面的那些值：

```
{ftam  pci(1)}
```

32 相对客体标识符类型记法

32.1 相对客体标识符类型(见3.6.57)应该用记法“RelativeOIDType”来引用：

```
RelativeOIDType::=
                    RELATIVE-OID
```

32.2 该类型的标记是通用类别，编号为 13。

32.3 相对客体标识符的值记法应该是“RelativeOIDValue”，或者当用作“XMLValue”时是“XMLRelativeOIDValue”。这些产生式是：

```
RelativeOIDValue::=
        "{"RelativeOIDComponentsList "}"
RelativeOIDComponentsList::=
        RelativeOIDComponents                    |
        RelativeOIDComponents RelativeOIDComponentsList
RelativeOIDComponents::=
        NumberForm                               |
        NameAndNumberForm
        DefinedValue
XMLRelativeOIDValue::=
        XMLRelativeOIDComponentList
XMLRelativeOIDComponentList::=
        XMLRelativeOIDComponent                  |
        XMLRelativeOIDComponent& " . " &XMLRelativeOIDComponentList
XMLRelativeOIDComponent::=
        XMLNumberForm                            |
        XMLNameAndNumberForm
```

32.4 产生式“NumberForm”、“NameAndNumberForm”、“XMLNumberForm”、“XMLNameAndNumberForm”及它们的语义在 31.3 至 31.11 条中定义。

32.5 “RelativeOIDComponents”的“DefinedValue”应该是类型相对客体标识符的，而且应该标识从客体标识符树中的某些开始节点到客体标识符树中的某些后来节点的弧的有序集合。开始节点用较早的“RelativeOIDComponents”(如果有的话)标识，而后面的“RelativeOIDComponents”(如果有的话)标识取自后面节点的弧。

32.6 第一个“RelativeOIDComponents”或者“XMLRelativeOIDComponent”标识从客体标识符树中的某些开始节点到客体标识符树中的某些后来节点的一个或多条弧。开始节点能用与类型定义相关的注解定义。如果与类型定义相关的注解内没有开始节点的定义，那么它需要作为通信实例中的客体标识符值传送(见 E.2.19)。要求开始节点既不能是根、也不能是紧接根下面的节点。

注：相对客体标识符值应该与特定的客体标识符值相关以便无歧义地标识客体。要求客体标识符值(见 31.10)至少有两个成分。这就是为什么要限制开始节点。

示例

用下面的定义：

```
thisUniversity   OBJECT IDENTIFIER ::=
                    {iso member-body country (29) universities (56) thisuni (32)}
firstgroup   RELATIVE-OID ::= {science-fac (4) maths-dept (3)}
```

或在 XML 值记法中：

```
thisUniversity::=<OBJECT_IDENTIFIER>1.2.29.56.32</ OBJECT_IDENTIFIER >
firstgroup::=<RELATIVE_OID>4.3</RELATIVE_OID>
```

相对客体标识符：

```
relOID   RELATIVE-OID ::= { firstgroup room (4) socket (6)}
```

或在 XML 值记法中：

relOID ::= <RELATIVE_OID>4.3.4.6</RELATIVE_OID>

可用来替代 OBJECT IDENTIFIER 值{1 2 29 56 32 4 3 4 6},只要当前根是 thisUniversity(通过应用知道或应用传送)。

33 嵌入式 pdv 类型的记法

33.1 嵌入式 pdv 类型(见 3.8.21)应该用记法"EmbeddedPDVType"来引用。

EmbeddedPDVType ::= EMBEDDED PDV

注:术语"Embedded PDV"是指取自嵌入到报文中的可能不同抽象语法(实质上,在隔开和标识协议中定义的报文编码和值)的抽象值。以前,它是指取自其在 OSI 表示层使用的"嵌入式表示层数据值(Embedded Presentation Data Value)",不过,现在已不再使用这个展开式,而且,它应该理解为"嵌入式值(embedded value)"。

33.2 该类型的标记是通用类别,编号为 11。

33.3 类型由表示下面的值组成:

a) 可能,但不是必须,是 ASN.1 类型的值的单个数据值编码;和

b) 以下的标识(单独或一起):

1) 抽象语法;和

2) 传送语法。

注 1:数据值可能是 ASN.1 类型的值,或者,例如:可能是静止图像或移动图片的编码。标识由一个或两个客体标识符组成,或者(在 OSI 环境中)引用规定抽象和传送语法的 OSI 表示上下文标识符。

注 2:标识抽象语法和/或编码也可能由应用层设计者确定为一个固定值,这时,它在通信实例中没有被编码。

33.4 嵌入式 pdv 类型有一个相关类型,该相关类型用来支持嵌入式 pdv 类型的值和子类型记法。

33.5 值定义和分成子类型的相关类型(假设是自动标记环境)是(有标准注解):

```
SEQUENCE {
    identification              CHOICE{
        syntaxes                SEQUENCE {
            abstract            OBJECT IDENTIFIER,
            transfer            OBJECT IDENTIFIER}
        --抽象和传送语法客体标识符--,
        syntax                  OBJECT IDENTIFIER
        --标识抽象和传送语法的单个客体标识符--,
        presentation-context-id INTEGER
        --(仅适用于 OSI 环境)会话的
        --OSI 表示上下文标识抽象和传送语法 --
        context-negotiation     SEQUENCE{
            presentation-context-id INTEGER,
            transfer-syntax     OBJECT IDENTIFIER}
        --(仅适用于 OSI 环境)进程中的上下文协商、
        --表示层上下文标识符只标识抽象语法,
        -- 因此,应该规定传送语法 --
        transfer-syntax         OBJECT IDENTIFIER
        --值的类型(例如,是 ASN.1 类型的值的规范)由应用
        --设计者固定(因此,发送者和接受者都了解),提供这一情况
        --主要是支持 ASN.1 类型的选择-范围-加密(或其他编码转换)--
        fixed                   NULL
```

```
        -- 数据值是固定 ASN.1 类型的值(因此,发送者和接受者都了解)--}
    data-value-descriptor          ObjectDescriptor OPTIONAL
        --这样提供值的类别的人可读标识--
    data-value                     OCTET STRING}
(WITH COMPONENTS{
        ...,
        data-value-descriptor  ABSENT } )
```

注:嵌入式 pdv 类型不允许包含 data-value-descriptor 值。然而,这里提供的相关类型的定义强调存在于嵌入式 pdv 类型、外部类型与未限制的字符串类型之间的共性。

33.6 当整数值应该是 OSI 定义的上下文集中的 OSI 表示上下文标识符时,则 presentation-context-id 替换项只适用于 OSI 环境。这一替换项不能用于 OSI 上下文协商期间。

33.7 context-negotiation 替换项应该仅用于 OSI 环境,而且,仅用于 OSI 上下文协商期间。整数值应该是提议附加在 OSI 定义的上下文集上的 OSI 表示上下文标识符。客体标识符 transfer-syntax 应标识为那个用来对值编码的 OSI 表示上下文提议的传送语法。

33.8 嵌入式 pdv 类型值的记法应该是 33.5 中定义的相关类型的值记法,其中,类型 OCTET STRING 的 data-value 成分的值表示采用 identification 中规定的传送语法的编码。

```
EmbeddedPdvValue ::= SequenceValue              -- 在 33.5 中定义的相关类型的值
XML EmbeddedPDVValue ::= XMLSequenceValue  -- 在 33.5 中定义的相关类型的值
```

例如,如果强加了单个选项,像使用语法,那么可通过书写下面的内容完成:

```
EMBEDDED PDV (WITH COMPONENT {
        ...,
        identification (WITH COMPONENT {
            syntaxes  PRESENT})})
```

34 外部类型的记法

34.1 外部类型(见 3.6.37)应该用记法"ExternalType"来引用:

```
ExternalType ::= EXTERNAL
```

34.2 该类型的标记是通用类别,编号为 8。

34.3 类型由表示下列内容的值组成:

a) 可能,但不是必须,是 ASN.1 类型值的单个数据值编码;和

b) 下面的标识(单独或一起):

1) 抽象语法;和

2) 传送输语法;和

c) (可选择)提供数据值范畴的人们可读描述的客体描述符。除非与使用"ExternalType"记法相关的注解明显地允许,否则不应该出现可选择的客体描述符。

注:33.3 的注 1 也适用于外部类型。

34.4 外部类型有相关类型,这个类型用来精确定义外部类型抽象值,也用来支持外部类型的值和子类型记法。

注:编码规则可定义用来衍生编码的不同类型,或不引用任何相关类型可规定编码。例如,由于历史的原因 BER 编码采用不同的序列类型。

34.5 值定义和分成子类型的相关类型(假设是自动标记环境)是(有标准注解):

```
SEQUENCE {
        identification                  CHOICE{
```

```
        syntaxes                    SEQUENCE {
            abstract                  OBJECT IDENTIFIER,
            transfer                  OBJECT IDENTIFIER}
        --抽象和传送语法客体标识符--,
        syntax                      OBJECT IDENTIFIER
        --标识抽象和传送语法的单个客体标识符--,
        presentation-context-id     INTEGER
        -(仅适用于 OSI 环境)会话的
        -- OSI 表示上下文标识抽象和传送语法 --
        context -negotiation        SEQUENCE{
            presentation-context-id   INTEGER,
            transfer-syntax           OBJECT IDENTIFIER}
        --(仅适用于 OSI 环境)进程中的上下文会话、
        --表示上下文标识符只标识抽象语法,
        -- 因此,应该规定传送语法 --
        transfer-syntax             OBJECT IDENTIFIER
        --值的类型(例如,是 ASN.1 类型的值的规范)由应用
        --设计者固定(因此,发送者和接受者都了解),提供这一情况
        --主要是支持 ASN.1 类型的选择-范围-加密(或其他编码转换)--
        fixed                       NULL
        -- 数据值是固定 ASN.1 类型的值(因此,发送者和接受者都了解)--}
    data-value-descriptor           ObjectDescriptor OPTIONAL
        --这样提供值的类别的人可读标识--
    data-value                      OCTET STRING}
        (WITH COMPONENTS{
          ...,
            identification (WITH COMPONENT {
                ...,
                syntaxes            ABSENT,
                transfer-syntax     ABSENT,
                fixed               ABSENT })})
```

注：由于历史原因，外部类型不允许有 identification 的替换项 syntaxes、transfer-syntax 或 fixed。要求这些选项的应用设计者应该使用嵌入式 pdv 类型。这里提供的相关类型的定义强调存在于外部类型、未限制的字符串类型与嵌入式 pdv 类型之间的共性。

34.6 33.6 和 33.7 的内容也适用于外部类型。

34.7 外部类型值的记法应该是 34.5 中定义的相关类型的值记法，其中，类型 OCTET STRING 的 data-value 成分的值表示使用 identification 中定义的传送语法的编码。

```
ExternalValue ::= SequenceValue          -- 在 34.5 中定义的相关类型的值
XMLExternalValue ::= XMLSequenceValue    -- 在 34.5 中定义的相关类型的值
```

注：由于历史的原因，编码规则能传输编码不是八位精确倍数的 EXTERNAL 中的嵌入式值。这样的值不能在用上面相关类型的值记法中表示。

35 字符串类型

这些类型由取自某些规定的字符汇中的字符串组成。通常用一个或多个表中的字符元来定义一个

字符汇及其编码，每个字符元对应于字汇中的一个字符。通常也给每个字符元赋予图形符号和字符名称，虽然在有些字汇中，字符元为空或者有名称但没有字型(有名称但没有形状的字符元的示例包括如GB/T 1988 中 EOF 的控制字符和如 GB/T 13000.1 中 THIN-SPACE 和 EN-SPACE 的空格字符)。

总之，与字符元相关的信息指明字汇中不同的抽象字符，即使信息是空的(那个字符元没有被赋予图形符号或名称)。

字符串类型的 ASN.1 基本值记法有三个变量(能组合而成)，形式规定如下：

a) 串中采用赋值图形符号(可能包含空格字符)的字符表达式；这是“cstring”记法；

注 1：当字汇中不只一个字符使用相同的图形符号时，这样的表达式在打印的表达式中可能有歧义。

注 2：当字汇中出现不同宽度的空格字符或规范用按比例-空格字体打印时，这样的表达式在打印的表达式中可能有歧义。

b) 通过给出已赋予字符的系列 ASN.1 值引用的字符串值中的字符列表；第 38 章中的模块 ASN1-CHARACTER-MODULE 中为 GB/T 13000.1 字符汇和 IA5String 字符汇定义了这类值引用的集合。除非用户使用上述 a)或下述 c)中描述的值记法指派这类值引用，否则，这一形式不会在其他字符汇中出现；

c) 通过用其字符元在字符汇表中的位置标识每个抽象字符的字符串值中的字符列表。这一形式只在 IA5String、UniversalString、UTF8String 和 BMPString 中出现。

字符串类型的 ASN.1 XML 值记法使用“xmlcstring”记法，它包括对某些特殊字符、和对采用十进制或十六进制(见 11.15)的字符的规范使用转义序列的能力。

36 字符串类型的记法

36.1 引用字符串类型(见 3.6.11)的记法应该是：

```
CharacterStringType ::=
        RestrictedCharacterStringType   |
        UnrestrictedCharacterStringType
```

“RestrictedCharacterStringType”是受限制的字符串类型的记法并且在第 37 章中定义。

“UnrestrictedCharacterStringType”是无限制的字符串类型的记法并且在 40.1 中定义。

36.2 每个受限制的字符串类型的标记在 37.1 中规定。无限制的字符串类型的标记在 40.2 中规定。

36.3 字符串值的记法应该是：

```
CharacterStringValue ::=
        RestrictedCharacterStringVlaue   |
        UnrestrictedCharacterStringValue
XMLCharacterStringValue ::=
        XMLRestrictedCharacterStringValue   |
        XMLUnrestrictedCharacterStringValue
```

“RestrictedCharacterStringValue”和“XMLRestrictedCharacterStringValue”分别在 37.8 和 37.9 中定义。“UnrestrictedCharacterStringValue”和“XMLUnrestrictedCharacterStringValue”是无限制的字符串值的记法并且在 40.7 中定义。

37 受限制字符串类型的定义

本章定义其值被限制为取自字符的某些规定集的零个、一个或多个字符的序列。引用受限制字符串类型的记法应该是“RestrictedCharacterStringType”：

```
RestrictedCharacterStringType ::=
        BMPString                |
```

GeneralString |
GraphicString |
IA5String |
ISO646String |
NumericString |
PrintableString |
TeletexString |
T61String |
UniversalString |
UTF8String |
VideotexString |
VisibleString

每个“RestrictedCharacterStringType”替换项通过规定下面的内容定义：

a) 赋给类型的标记；和

b) 所被类型引用的名称(例如，数字串)；和

c) 通过引用列出图形符号的表或引用编码字符集 ISO 国际登记(见所使用的具有转义序列的编码字符集的 ISO 国际登记)中的登记号或引用 GB/T 13000.1，用于定义类型的字符集中的字符。

37.1 表 6 列出了所引用的每个受限制字符串类型的名称，赋给类型的通用类别标记的序号，定义登记号或表，或者定义文本章号，以及在必要时，标识与表中登录有关的注解。记法中定义了同义名称时，要用圆括号列出。

37.2 表 7 列出会在数字串类型和数字串字符抽象语法中出现的字符。

表 6 受限制字符串列表

引用类型的名称	通用类别号	定义登记号[a]、表号、或 GB/T 16262.1 章	注
TF8String	12	37.16 条	
NumericString	18	表 7	注 1
PrintableString	19	表 8	注 1
TeletexString(T61String)	20	6、87、102、103、106、107、126、144、150、153、156、164、165、168+SPACE+DELETE	注 2
VideotexString	21	1、13、72、73、87、89、102、108、126、128、129、144、150、153、164、165、168+ SPACE+DELETE	注 3
IA5String	22	1、6+ SPACE+DELETE	
GraphicString	25	所有 G 集合+SPACE	
VisibleString(ISO646String)	26	6+SPACE	
GeneralString	27	所有 G 和所有集合−SPACE+DELETE	
UniversalString	28	见 37.6	
BMPString	30	见 37.15	

注 1：类型风格、大小、颜色、强度或者其他显示特性没有意义。

注 2：登记号 6 和 156 能代替 102 和 103 使用。

注 3：对应这些登记号的登录提供 CCITT Rec. T.100 和 ITU-T Rec. T.101 的功能。

a) 所使用的具有转义序列的编码字符集的 ISO 国际登记中列出了定义登记号。

表 7 Numer icString

名称	图形
Digitals	0、1、...9
Space	(space)

37.3 赋予下面的客体标识符和客体描述符值来标识和描述数字串字符抽象语法：

{joint-iso-itu-t asn1(1) specification(0) characterStrings(1) numericString(0)}

和

" NumericString character abstract syntax "

注 1：这个客体标识符值能用于 CHARACTER STRING 值和其他需要携带与这些值分开的字符串类型的标识的情况中。

注 2：NumericString 字符抽象语法的值可用下面来编码：

a) GB/T 13000.1 给出的编码抽象字符的规则之一。这时，用与 GB/T 13000.1 附录 N 中的那些规则相关的客体标识符来标识字符传送语法。

b) 固有类型 NumericString 的 ASN.1 编码规则。这时，字符传送语法由客体标识符值{joint-iso-itu-t asn1(1) basic-encoding(1)}标识。

37.4 表 8 列出会在 PrintableString 类型和 PrintableString 字符抽象语法中出现的字符。

表 8 PrintableString

名 字		图 形
拉丁大写字母	Latin capital letters	A、B、...Z
拉丁小写字母	Latin small letters	a、b、...z
数字	Digits	0、1、...9
间隔	SPACE	(空)
撇号	APOSTROPHE	'
左圆括号	LEFT PARENTHESIS	(
右圆括号	RIGHT PARENTHESIS	)
正号	PLUS SIGN	+
逗号	COMMA	,
负号	HYPHEN-MINUS	-
句号	FULL STOP	.
斜线	SOLIDUS	/
冒号	COLON	:
等于记号	EQUALS SIGN	=
问号	QUESTION MARK	?

37.5 赋予下面的客体标识符和客体描述符值来标识和描述 PrintableString 串字符抽象语法：

{joint-iso-itu-t asn1(1) specification (0) CharacterStrings(1) printableString(1)}

和

" PrintableString character abstract syntax "

注 1：这个客体标识符值能用于 CHARACTER STRING 值和需要携带与这些值分开的字符串类型的标识的其他情况中。

注 2：PrintableString 字符抽象语法的值可用下面方法编码：

a) 在 GB/T 13000.1 中给出的编码抽象字符的规则之一。这时，用与 GB/T 13000.1 附录 N 中的那些规则相关的客体标识符来标识字符传送语法。

b) 固有类型 PrintableString 的 ASN.1 编码规则。这时，字符传送语法由客体标识符{joint-iso-itu-t asn1(1) basic-encoding(1)}标识。

37.6 会在 UniversalString 类型中出现的字符是 GB/T 13000.1 允许的任何字符。

37.7 使用这种类型要用到 GB/T 13000.1 规定的一致性要求。

注：第 38 章为 GB/T 13000.1 附录 A 中定义的“子集图形字符集”定义了包含这种类型许多子类型的 ASN.1 模块。

37.8 受限制字符串类型的“RestrictedCharacterStringValue”记法应该是“cstring”(见 11.14),“CharacterStringList”、“Quadruple”或“Tuple”。“Quadruple”仅能定义长度为 1 的字符串,而且仅用在 UniversalString、UTF8String 或 BMPString 类型的值记法中。“Tuple”仅能定义长度为 1 的字符串,且仅用在 IA5String 类型的值记法中。

```
RestrictedCharacterStringValue ::=
        cstring                 |
        CharacterStringList     |
        Quadruple               |
        Tuple
CharacterStringList ::=  " {"  CharSyms  " }"
CharSyms ::=
        CharsDefn               |
        CharSyms  " , "  CharsDefn
CharsDefn ::=
        cstring                 |
        Quadruple               |
        Tuple                   |
        DefinedValue
Quadruple ::= " {" Group " , " Plane " , " Row " , " Cell "}"
Group  ::= number
Plane  ::= number
Row    ::= number
Cell   ::= number
Tuple  ::= " {" TableColumn " , " TableRow " }"
TableColumn ::= number
TableRow ::= number
```

注 1：“cstring”记法仅可无歧义地用在能显示值中出现的字符的图形符号的媒体上。相反,如果媒介没有这种能力,无歧义地规定采用这种图形符号的字符串值的唯一手段是用“CharacterStringList”记法的方法,而且,只有类型是 UniversalString、UTF8String、BMPString 或 IA5String 和使用“CharsDefn”的“DefinedValue”替换项(见 7.1.2)。

注 2：第 38 章定义了指明类型 BMPString(和 UniveralString 及 UTF8String)和 IA5String 的单个字符(大小为 1 的串)的许多“valuereference”。

例如,假设当字符“Σ”不能在现有媒体中表示时,我们希望为 UniversalString 规定“abcΣdef”的值,该值也能表示如下：

```
IMPORTS BasicLatin, greekCapitalLetterSigma FROM ASN1-CHARACTER-MODULE
        {joint-iso-itu-t asn1(1)  specification(0)  modules(0) iso10646(0) };
MyAlphabet::= UniversalString (FROM (BasicLatin | greekCapitalLetterSigma))
mystring MyAlphabet::= {" abc", greekCapitalLetterSigma, " def"}
```

注 3：当规定 UniversalString、UTF8String 或 BMPString 类型的值时,除非能解决有类似形状的不同图形字符出现的歧义性,否则不应采用“cstring”记法。

例如：不应使用下面的“cstring”记法,因为在 BASIC LATIN、CYRILLIC 和 BASIC GREEK 字母表中出现图

形符号'H'、'O'、'P'和'E'并且是有歧义的。

```
IMPORTS BasicLatin, Cyrillic, BasicGreek FROM ASN1-CHARACTER-MODULE
        {joint-iso-itu-t asn1(1)  specification(0)  modules(0) iso10646(0)};
MyAlphabet::= UniversalString (FROM (BasicLatin | Cyrillic | BasicGreek ) )
mystring MyAlphabet::= "HOPE"
```

无歧义定义 mystring 的替换项将是：

```
mystring MyAlphabet (BasicLatin)::= " HOPE"
```

形式上，mystring 是 MyAlphabet 的子集的值的值引用，但是，它能通过附录 B 的值映射规则用于 MyAlphabet 内这个值需要值引用的任何地方。

37.9 "XMLRestrictedCharacterStringValue"记法是：

```
XMLRestrictedCharacterStringValue::=xmlcstring
```

37.10 有些字符不能直接用"xmlcstring"表示。他们应该采用 11.15 中规定的转义序列表示。

注：如果受限制字符串值包含 11.15.1 中规定的非 GB/T 13000.1 字符的字符，他们不能用"xmlcstring"表示，而且这些值不能用 XML 编码规则传送(见 ISO/IEC 8825-4)。

37.11 在"CharsDefn"中的"DefinedValue"应该是引用那个类型的值。

37.12 在"Plane"、"Row"、"Cell"产生式中的"number"应该小于 256，而在"Group"产生式中，它应该小于 128。

37.13 "Group"规定 UCS 编码范围中的一个组，"Plane"规定组内的平面，"Row"规定平面内的行，"Cell"规定行内的字符元。这个记法标识的抽象字符是"Group"、"Plane"、"Row"和"Cell"值规定的字符元的抽象字符。在所有情形中，允许的字符集可用分成子类型限制。

注：当使用像 GeneralString、GraphicString 和 UniversalString 等没有应用约束的开放-结尾字符串类型时，应用设计者应该仔细考虑一致性暗示。像 TeletexString 等有界但大的字符串类型，也需要有关一致性的详细文本。

37.14 在"TableColumn"产生式中的"number"应该是 0 到 7 的范围，而"TableRow"产生式中的"number"应该是 0 到 15 的范围。"TableColumn"规定符合 GB/T 2311 图 1 的字符代码表的列，"TableRow"规定其中的行。当代码表包含第 0 列和第 1 列中的登记登录 1，以及第 2 列至第 7 列中的登记登录 6 时，这一记法仅用于 IA5String(见与转义序列使用的编码字符集的 ISO 国际登记)。

37.15 BMPString 是有自身唯一标记和只包含 GB/T 13000.1 的基本多文种平面(那些对应最先 64K-2 字符元的，其编码用来表示基本多文种平面之外的字符的少量字符元)中的字符的 UniversalString 的子类型。它有如下定义的相关类型：

```
UniversalString(Bmp)
```

其中 Bmp 在 ASN.1 模块 ASN1-CHARACTER-MODULE(见第 38 章)中定义为对应 GB/T 13000.1 附录 A 中定义的"BMP"集名称的 UniversalString 的子类型。

注 1：由于 BMPString 是固有类型，它没有在 ASN1-CHARACTER-MODULE 中定义。

注 2：定义 BMPString 为固有类型是为了使没有约束的编码规则(如 BER)能用 16 位而不是 32 位编码。

注 3：在值记法中，所有的 BMPString 值是有效的 UniversalString 和 UTF8String 值。

37.16 UTF8String 在抽象层是和 UniversalString 同义的，而且能用于使用 UniversalString 的地方(符合要求不同标记的规则)，但是它有不同的标记，并且是不同的类型。

注：BER 和 PER 使用的 UTF8String 的编码与 UniversalString 的不同，而且，对于大部分文本将少有多余的词。

38 GB/T 13000.1 中定义的命名字符和集

本章规定包含取自 GB/T 13000.1 的每个字符的值引用名定义的 ASN.1 固有模块，其中，每个名称引用编号为 1 的 UniversalString 值。这个模块也包含为取自 GB/T 13000.1 的每个字符集的类型引用名称的定义，其中每个名称引用 UniversalString 类型的子集。

注：这些值可用于 UniversalString 类型及其衍生的类型的值记法。在 38.1 规定的模块中已定义的所有值和类型引用都输出，而且必须通过采用他们的任意模块引入。

38.1 ASN.1 模块"ASN1-CHARACTER-MODULE"的规范

该模块没有在这里全部打印，而是规定定义它的方法。

38.1.1 模块以下列为开始：

```
ASN1-CHARACTER-MODULE {joint-iso-itu-t asn1(1) specification(0) modules(0)
    iso10646(0)}
    DEFINITIONS ::=BEGIN
    -- 本模块内定义的所有值引用和类型引用隐式输出，
    -- 而且用任何模块引入。
    -- GB/T 1988 控制字符：
    nul IA5String ::= {0, 0}
    soh IA5String ::= {0, 1}
    stx IA5String ::= {0, 2}
    etx IA5String ::= {0, 3}
    eot IA5String ::= {0, 4}
    enq IA5String ::= {0, 5}
    ack IA5String ::= {0, 6}
    bel IA5String ::= {0, 7}
    bs IA5String ::= {0, 8}
    ht IA5String ::= {0, 9}
    lf IA5String ::= {0, 10}
    vt IA5String ::= {0, 11}
    ff IA5String ::= {0, 12}
    cr IA5String ::= {0, 13}
    so IA5String ::= {0, 14}
    si IA5String ::= {0, 15}
    dle IA5String ::= {1, 0}
    dc1 IA5String ::= {1, 1}
    dc2 IA5String ::= {1, 2}
    dc3 IA5String ::= {1, 3}
    dc4 IA5String ::= {1, 4}
    nak IA5String ::= {1, 5}
    syn IA5String ::= {1, 6}
    etb IA5String ::= {1, 7}
    can IA5String ::= {1, 8}
    em IA5String ::= {1, 9}
    sub IA5String ::= {1, 10}
    esc IA5String ::= {1, 11}
    is4 IA5String ::= {1, 12}
    is3 IA5String ::= {1, 13}
    is2 IA5String ::= {1, 14}
    is1 IA5String ::= {1, 15}
    del IA5String ::= {7, 15}
```

38.1.2 对于 GB/T 13000.1 中第 24 章和第 25 章中给出的图形字符(glyphs)的字符名称的每个列表

中的每个登录，模块包含形式陈述：

<namedcharacter>BMPString ::= <tablecell>

-- 表示字符<iso10646name>，见 GB/T 13000.1

其中：

a) <iso10646name>是从 GB/T 13000.1 中列出的一个表衍生出来的字符名称；

b) < namedcharacter >通过应用 38.2 中规定的程序到<iso10646 name >获得的串；

c) < tablecell >是对应列表登录的 GB/T 13000.1 中表字符元中的图形字符。

例如：

```
latinCapitalLetterA BMPString ::= {0, 0, 0, 65}
-- 表示字符 LATIN CAPITAL LETTER A，见 GB/T 13000.1
greekCapitalLetterSigma BMPString ::= {0, 0, 3, 163}
-- 表示字符 GREEK CAPITAL LETTER SIGMA，见 GB/T 13000.1
```

38.1.3 对 GB/T 13000.1 附录 A 中规定的图形字符的集的每个名称，形式模块中包含陈述：

```
<namedcollectionstring> ::= BMPString
        (FROM(<alternativelist>))
    -- 表示字符集<collectionstring>，
    -- 见 GB/T 13000.1。
```

其中

a) <collectionstring>是在 GB/T 13000.1 中指派的字符集的名称；

b) < namedcollectionstring >通过应用 38.3 的程序到< collectionstring >形成的；

c) < alternativelist >是像为 GB/T 13000.1 规定的每个字符在 38.2 中产生的一样，通过使用< namedcharacter >形成的。

产生的类型引用，< namedcollectionstring >，形成了有限的子集。（见附录 F 中的指南）

注：有限子集是规定子集中的字符表。它与选择的子集相反，是 GB/T 13000.1 附录 A 中列出的字符集加 BASIC LATIN 集。

例如(部分)：

```
space BMPString                 ::= {0, 0, 0,3 2}
exclamationMark BMPString       ::= {0, 0, 0,3 3}
quotationMark BMPString         ::= {0, 0, 0, 34}
...                             -- 等等
tilde BMPString                 ::= {0, 0, 0, 126}
BasicLatin::= BMPString
        (FROM (space
        | exclamationMark
        | quotationMark
        |...      -- 等等
        | tilde ))
        )
-- 表示字符 BASIC LATIN 集，见 GB/T 13000.1。
-- 本示例中的圆括号用来简化并且是“等等”的意思；
-- 不能在实际的 ASN.1 模块中这样使用。
```

38.1.4 GB/T 13000.1 定义实现的三个级。默认情况下，由于这些类型没有对使用组合字符的限制，故在 ASN1-CHARACTER-MODULE 中定义的所有类型，除“Level1”和“Level 2”外，都符合 3 级实

现。“Level 1”指明要求 1 级实现,“Level 2”指明 2 级实现,而“Level 3”指明 3 级实现。因此,ASN1-CHARACTER-MODULE 中定义了下列内容:

Level1 ::= BMPString (FROM (ALL EXCEPT CombiningCharacters))

Level2 ::= BMPString (FROM (ALL EXCEPT CombiningCharactersType-2))

Level3 ::= BMPString

注 1:“CombiningCharacters”和“CombiningCharactersType -2”是分别对应于 GB/T 13000.1 附录 A 中定义的“COMBINING CHARACTERS”和“COMBINING CHARACTERS B-2”的< namedcollectionstring >。

注 2:“Level1”和“Level2”将用于“IntersectionMark”(见第 46 章)之后或作为“ConstraintSpec”中唯一的约束。(示例见附录 E.2.7.1。)

注 3:有关本内容的更多信息见附录 F.2.5。

38.1.5 模块以下面的语句结束:

END

38.1.6 用户定义的,和 38.1.3 中的示例相当的是:

BasicLatin ::= BMPString (FROM (space..tidle))

-- 表示字符 BASIC LATIN 集,见 GB/T 13000.1。

38.2 <namedcharacter>是采用<iso10646name>(见 38.1.2)和应用下列算法获得的串:

a) <iso10646name>的每个大写字母转换成对应的小写字母,除非大写字母是以 SPACE 开头,此时,大写字母保持不变;

b) 每个数字和每个 HYPHEN-MINUS 保持不变;

c) 删除每个 SPACE。

注:上面的算法,和 GB/T 13000.1 附录 K 中的字符命名指南一起将总是为 GB/T 13000.1 中列出的各个字符名产生无歧义的值记法。

例如:取自 GB/T 13000.1 的 0 行 60 字符元,被命名为“LESS-THAN SIGN”并且有图形表达式“<”的字符能用 less-thanSign 的“DefinedValue”引用。

38.3 < namedcollectionstring >是采用< collectionstring >和应用下列算法获得的串:

a) GB/T 13000.1 集名的每个大写字母转换成对应的小写字母,除非大写字母是以 SPACE 开头或它是名字的第一个字母,此时,大写字母保持不变;

b) 每个数字和每个 HYPHEN-MINUS 保持不变;

c) 删除每个 SPACE。

例如:

1 在 GB/T 13000.1 附录 A 中定义为 BASIC LATIN 的集有 ASN.1 类型引用

BasicLatin

2 由 BASIC LATIN 集中的字符以及 BASIC ARABIC 集构成的字符串类型能被定义为:

my-Character-String::=BMPString (FROM (BasicLatin | BasicArabic))

注:上面的结构是必需的,因为明显的更简单结构

my-Character-String::=BMPString (FROM (BasicLatin | BasicArabic))

将只允许是整个 BASIC LATIN 或 BASIC ARABIC,而不是二者混合的串。

39 字符的正则顺序

39.1 为了“ValueRange”分成子类型并为编码规则可使用,为 UniversalString、UTF8String、BMP-String、NumericString、PrintableString、VisibleString 和 IA5String 规定了字符的正则顺序。

39.2 仅仅因为本章,字符和代码表中的字符元一一对应,不管那个字符元被赋予字符名称或者形状,也不管它是控制字符还是印刷字符、组合还是非组合字符。

39.3 抽象字符的正则顺序由 GB/T 13000.1 的 32 位表达式中其值的正则顺序定义,正则顺序中小数

字首先出现和大数字最后出现。

39.4 在“PermittedAlphabet”记法(或个别字符)内“ValueRanges”的结束点能用模块 ASN1-CHARACTER-MODULE 中定义的 ASN.1 值引用或(这时,图形符号在规范上下文和用来表示他的媒体中是无歧义的)通过给出“cstring”中的图形符号(38.1 中定义了 ASN1-CHARACTER-MODULE),或通过采用 37.8 的“Quadruple”或“Tuple”记法规定。

39.5 对于 BMPString,字符元的正则顺序定义(见 GB/T 13000.1)为:

256 * (行号)+(字符元号)

整个字符集准确地包含 256 * 256 个字符。在“PermittedAlphabet”记法内(或单个字符)的“ValueRange”结束点能用模块 ASN1-CHARACTER-MODULE 中定义的 ASN.1 值引用或(规范的上下文中图形符号明确时)在“cstring”中给出图形符号来规定。不能将一个字符元规定为范围的结束点或者没有名字或图形符号赋给那个字符元时标识单个字符。

39.6 对 NumericString,从左到右逐步增加的正则顺序定义(见 37.2 的表 7)为:

(间隔) 0 1 2 3 4 5 6 7 8 9

整个字符集精确地包含 11 个字符。“ValueRange”(或单个字符)的结束点能用“cstring”中的图形符号规定。

注:这个顺序与 GB/T 13000.1 中 BASIC LATIN 集中对应字符的顺序一致。

39.7 对于 PrintableString,从左到右和从顶到底逐步增加的正则顺序定义(见 37.4 的表 8)为:

(SPACE) (APOSTROPHE) (LEFT PARENTHESIS) (RIGHT PARENTHESIS) (PLUS SIGN) (COMMA) (HYPHEN-MINUS) (FULL STOP) (SOLIDUS) 0123456789 (COLON) (EQUAL SIGN) (QUESTION MARK) ABCDEFGHIJKLMNOPQRSTUVWXYZ-abcdefghijklmnopqrstuvwxyz

整个字符集精确地包含 74 个字符。“ValueRange”(或单个字符)的结束点能用“cstring”中的图形符号规定。

注:这个顺序与 GB/T 13000.1 中 BASIC LATIN 集中对应字符的顺序一致。

39.8 对于 VisibleString,字符元的正则顺序根据 GB/T 1988 编码(称为 GB/T 1988 ENCODING)定义为:

(GB/T 1988 ENCODING)- 32

注:即,正则顺序与 GB/T 1988 代码表的字符元 2/0 至 7/14 中的字符一样。

整个字符集精确地包含 95 个字符。“ValueRange”(或单个字符)的结束点能用“cstring”中的图形符号规定。

39.9 对于 IA5String,字符元的正则顺序根据 GB/T 1988 编码定义为:

(GB/T 1988 ENCODING)

整个字符集精确地包含 128 个字符。“ValueRange”(或单个字符)的结束点能用“cstring”中的图形符号或 38.1.1 中定义的 GB/T 1988 控制字符值引用规定。

40 无限制字符串类型的定义

本章定义类型值是任何字符抽象语法值的类型。在 OSI 环境中,该抽象语法可能是 OSI 定义的上下文集的一部分。否则,可为使用无限制字符串类型的每个实例直接引用。

注 1:抽象字符语法(及一个或多个对应的字符传送语法)可由能分配 ASN.1 OBJECT IDENTIFIERS 的任何组织定义。

注 2:利益相同产生的外形通常将确定对 CHARACTER STRING 规定实例或实例组所支持的字符抽象语法和字符传送语法。在 OSI 应用中,它通常应包括 OSI 协议实现一致性声明(PICS)中引用所支持的语法。

40.1 无限制的字符串类型(见 3.6.76)应该用记法“UnrestrictedCharacterStringType”引用:

UnrestrictedCharacterStringType ::= CHARACTER STRING

40.2 该类型的标记是通用类别，编号为 29。

40.3 类型由表示下列内容的值组成：

a) 可能但不必是 ASN.1 字符串类型的值的字符串值；和

b) 以下的标识(单独或一起)：

1) 字符抽象语法；和

2) 字符传送语法。

40.4 无限制字符串类型有相关类型。用该相关类型来支持其值和子类型记法。

40.5 为值定义和分成子类型的相关类型(假设是自动标记环境)是(有标准注解)：

```
SEQUENCE {
    identification              CHOICE{
        syntaxes                  SEQUENCE {
            abstract                OBJECT IDENTIFIER,
            transfer                OBJECT IDENTIFIER}
            --抽象和传送语法客体标识符--,
        syntax                    OBJECT IDENTIFIER
            --标识抽象和传送语法的单个客体标识符--,
        presentation-context-id   INTEGER
        -(仅适用于 OSI 环境)
        -- 协商的 OSI 表示上下文标识抽象和传送语法 --
        context -negotiation      SEQUENCE{
            presentation-context-id   INTEGER,
            transfer-syntax           OBJECT IDENTIFIER}
        --(仅适用于 OSI 环境)
        -- 进程中的上下文协商、表示上下文标识符只标识抽象语法，
        -- 因此，应该规定传送语法 --
        transfer-syntax           OBJECT IDENTIFIER
        --值的类型(例如，是 ASN.1 类型的值的规范)由应用
        --设计者固定(因此，发送者和接受者都了解)，--
        -- 提供这一情况主要是支持 ASN.1 类型的选择-范围-加密(或其他编码转换)--
        fixed                     NULL
        -- 数据值是固定 ASN.1 类型的值(因此，发送者和接受者都了解)--}
    data-value-descriptor       ObjectDescriptor OPTIONAL
        --这样提供值的类别的人可读标识 --
    string-value                OCTET STRING }
    ( WITH COMPONENTS {
        ... ,
        data-value-descriptor   ABSENT } )
```

注：无限制字符串类型不允许包含与“identification”一起的“data-value-descriptor”值。然而，这里提供的相关类型的定义强调嵌入式 pdv 类型、外部类型与无限制字符串类型之间存在的共性。

40.6 正文的 33.6 和 33.7 也适用于无限制字符串类型。

40.7 值记法应该是相关类型的值记法，其中类型 OCTET STRING 的 string-value 成分的值表示使用“identification”中规定的传送语法的编码。

UnrestrictedCharacterStringValue ::= SequenceValue -- 在 40.5 中定义的相关类型的值

XMLUnrestrictedCharacterStringValue ::=

XMLSequenceValue -- 在 40.5 中定义的相关类型的值

40.8 在附录 E.2.8 中给出了无限制字符串类型的示例。

41 第 42 至 44 章中定义的类型的记法

41.1 引用 42 至 44 章中定义的类型的记法应该是：

UsefulType ::= typereference

其中，“typereference”是 42 至 44 章中用 ASN.1 记法定义的之一。

41.2 每个“UsefulType”的标记在 42 至 44 章中规定。

42 通用时间

42.1 这一类型应该用下面名称引用：

GeneralizedTime

42.2 该类型由表示下列内容的值组成：

a) 如 GB/T 7408 定义的日历日期；和

b) 一天的时间，除值 24 不能用于小时外，精确到 GB/T 7408 中定义的任何程度；和

c) 如 GB/T 7408 中定义的当地时差因素。

42.3 该类型可用 ASN.1 定义如下：

GeneralizedTime ::= [UNIVERSAL 24] IMPLICIT VisibleString

带有“VisibleString”的值限制在下面字符串之一：

a) 如 GB/T 7408 规定的、表示日历日期的串，用四位数字表示年，二位数字表示月和二位数字表示日，不使用分隔号，后随 GB/T 7408 中规定的表示一天时间的串，没有分隔号，而有十进制逗号或日期(如 GB/T 7408 中提供的)，也没有终止符 Z(如 GB/T 7408 中提供的)；或

b) 上面 a)中的字符后面紧接大写字母 Z；或

c) 上面 a)中的字符后面紧接表示当地时间差的字符串，如 GB/T 7408 中规定的，没有分隔符。

在 a)的情况下，时间应表示当地时间。在 b)的情况下，时间应表示国际协调时。在 c)的情况下，如 a)情况形成的串部分表示当地时间(t_1)，而时差(t_2)能按下列方法确定国际协调时：

国际协调时是 $t_1 - t_2$

例如：

情况 a)：

"19851106210627.3" 表示当地时间 1985 年 11 月 6 日下午 9 时 6 分 27.3 秒。

情况 b)：

"19851106210627.3Z" 是上述的国际协调时。

情况 c)：

"19851106210627.3-0500" 表示例 a)中的当地时间，当地时间比国际协调时晚 5 个 h。

42.4 标记应该如 41.3 中定义。

42.5 值记法应该是 42.3 中定义的“VisibleString”的值记法。

43 世界时间

43.1 该类型应该用下列名称引用：

UTCTime

43.2 该类型由表示下列内容的值组成：

a) 日历日期;和

b) 精确到一分或一秒的时间;和

c) (可选择)与国际协调时不同的当地时间。

43.3 该类型用 ASN.1 定义如下:

UTCTime ::= [UNIVERSAL 23] IMPLICIT VisibleString

带有"VisibleString"的值限制为下面平行放置的字符串:

a) 六位数字 YYMMDD,其中,YY 是公元年号的后两位数,MM 是月份(一月记为 01),DD 是月中的日(01 至 31);和

b) 下面二者之一:

1) 四位数字 hhmm,其中 hh 是小时(00 至 23),mm 是分(00 至 59);或

2) 六位数字 hhmmss,其中,hh 和 mm 如上面 1)中的时间,ss 是秒(00 至 59);和

c) 下面二者之一:

1) 字符 Z;或

2) 字符+或-之一,后随 hhmm,其中 hh 是小时,mm 是分。

上面 b)中的替换项允许改变时间规范内的精确度。

在替换项 c)1)中,时间是国际协调时。在替换项 c)2)中,上面 a)和 b)规定的时间(t_1)是当地时间;上面 c)2)规定的时差(t_2)能使国际协调时按下列来确定:

等同的世界时间是 t_1-t_2

例 1:如果当地时间是 1982 年 1 月 2 日上午 7 时,而国际协调时是 1982 年 1 月 2 日中午 12 时,UTCTime 值是二者之一:

——"8201021200Z";或

——"8201020700-0500"。

例 2:如果当地时间是 2001 年 1 月 2 日上午 7 时,而国际协调时是 2001 年 1 月 2 日中午 12 时,UTCTime 值是二者之一:

——"0101021200Z";或

——"0101020700-0500"。

43.4 标记应该如 43.3 中定义。

43.5 值记法应该是 43.3 中定义的"VisibleString"的值记法。

44 客体描述符类型

44.1 该类型应该用下列名称引用:

ObjectDescriptor

44.2 该类型由用来描述一个客体的人们可读的文本组成。文本不是客体的无歧义标识,但是不同客体的相同文本并不常见。

注:建议将类型 OBJECT INDENTIFIER 值赋给客体的权威机构也应该将类型 ObjectDescriptor 的值赋给那个客体。

44.3 该类型用 ASN.1 定义如下:

ObjectDescriptor ::= [UNIVERSAL 7] IMPLICIT GraphicString

"GraphicString"含有描述客体的文本。

44.4 标记应该如 44.3 中定义。

44.5 值记法应该是 44.3 中定义的"GraphicString"的值记法。

45 受约束类型

45.1 "ConstrainedType"记法允许对(双亲)类型加以约束,约束其值集为双亲的某些子类型或(在集

合或序列类型内)规定成分关系适用于双亲类型的值,以及同一集合或序列值中某些其他成分的值。它也允许有与约束有关的例外标识符。

```
ConstrainedType ::=
        Type Constraint      |
        TypeWithConstraint
```

在第一个替换项中,双亲类型是"Type",而且该约束由 45.6 中定义的"Constraint"规定。第二个替换项在 45.5 中定义。

45.2 当"Constraint"记法紧接单一集合或单一序列类型记法之后时,它适用于(最里面的)单一集合或单一序列记法中的"Type",而不适合单一集合或单一序列类型。

注:例如,下面的约束"(SIZE(1..64))"适用于 VisibleString,但不适用于 SEQUENCE OF:

```
NamesOfMemberNations ::= SEQUECNCE OF VisibleString (SIZE (1..64))
```

45.3 当"Constraint"记法紧接精选类型记法之后时,它也适用于选择类型,但不适用于所选替换项的类型。忽略这样的约束(见 29.2)。

注:在下面的示例中,约束(WITH COMPONENTS {..., a ABSENT})适用于 CHOICE 类型 T,但不适用于所选的 SEQUENCE 类型,而且不影响 V 的值。

```
T::=CHOICE{
    a  SEQUENCE {
        a  INTEGER OPTIONAL,
        b  BOOLEAN
    },
    b  NULL
}
V::= a < T (WITH COMPONENTS { ..., a  ABSENT})
```

45.4 当"Constraint"记法紧接"TaggedType"记法之后时,全部记法的解释一样,无论"TaggedType"或"Type"是否认为是双亲类型。

45.5 作为 45.2 中规定的解释的结果,提供特定的记法来允许约束适用于单一集合或单一序列类型。这就是"TypeWithConstraint":

```
TypeWithConstraint ::=
        SET Constraint OF Type                |
        SET SizeConstraint OF Type            |
        SEQUENCE Constraint OF Type           |
        SEQUENCE SizeConstraint OF Type       |
        SET Constraint OF NamedType           |
        SET SizeConstraint OF NamedType       |
        SEQUENCE Constraint OF NamedType      |
        SEQUENCE SizeConstraint OF NamedType
```

在第一和第二个替换项中,双亲类型是"SET OF Type",而在第三个和第四个中,它是"SEQUENCE OF Type"。在第五和第六个替换项中,双亲类型是"SET OF NamedType",而在第七个和第八个中,它是"SEQUENCE OF NamedType"。在第一、第三、第五和七个替换项中,约束是"Constraint"(见 45.6),而在第二、第四、第六和第八个中,它是"SizeConstraint"(见 47.5)。

注:尽管"Constraint"替换项包括了对应的"SizeConstraint"替换项,由于历史的原因提供了"SizeConstraint"替换项。

45.6 由记法"Constraint"规定的约束:

```
Constraint ::= " (" ConstraintSpec ExceptionalSpec " )"
ConstraintSpec ::=
```

SubtypeConstraint |
GeneralConstraint

"ExceptionalSpec"在 49 中定义。除非与"extension marker"(见第 48 章)一起使用,否则,它仅表示"ConstraintSpec"是否包括"DummyReference"(见 GB/T 16262.4 的 8.3)或者是"UserDefinedConstraint"(见 GB/T 16262.3 的第 9 章)的事件。在 GB/T 16262.3 的 8.1 中定义了"GeneralConstraint"。

45.7 记法"SubtypeConstraint"是通用"ElementSetSpecs"记法(见第 46 章):

SubtypeConstraint ::= ElementSetSpecs

在这里的上下文中,元素是双亲类型的值(元素集的支配者是双亲类型)。在集中至少应该有一个元素。

46 元素集规范

46.1 在某些记法中,可以规定某些标识类型或信息客体类别(支配者)的元素集。这时,使用记法"ElementSetSpec":

```
ElementSetSpecs ::=
        RootElementSetSpec                          |
        RootElementSetSpec "," "..."                |
        RootElementSetSpec "," "..." "," AdditonalElementSetSpec
RootElementSetSpec ::= ElementSetSpec
AdditonalElementSetSpec ::= ElementSetSpec
ElementSetSpec ::= Unions   |
        ALL Exclusions
Unions ::= Intersections   |
        UElems UnionMark Intersections
UElems ::= Unions
Intersections ::= IntersectionElement   |
        IElems IntersectionMark IntersectionElements
IElems ::= Intersections
IntersectionElements ::= Elements | Elems Exclusions
Elems ::= Elements
Exclusions ::= EXCEPT Elements
UnionMark ::= "|"   |   UNION
IntersectionMark ::= "^" | INTERSECTION
```

注 1:脱字字符"^"和单词 INTERSECTION 同义。字符"|"和单词 UNION 同义。建议,作为格式问题,字符或单词能用于整个用户规范。EXCEPT 能以任一种形式使用。

注 2:从最高到最低的优先顺序是:EXCEPT、"^"、"|"。注意,规定了"ALL EXCEPT"以便它不能用没有把"ALL EXCEPT xxx"圆括号括起来的其他约束打断。

注 3:任何有"Elements"的地方,会出现没有圆括号的约束[例如:INTEGER (1..4)]或圆括号括起来的子类型约束[例如:INTEGER((1..4 | 9))]。

注 4:注意,两个"EXCEPT"运算符必须用"|"、"^"、"("或")"分开,因此,(A EXCEPT B EXCEPT C)是不允许的。它必须改成((A EXCEPT B) EXCEPT C)或(A EXCEPT (B EXCEPT C))。

注 5:注意,((A EXCEPT B) EXCEPT C) 与 (A EXCEPT (B | C)) 是相同的。

注 6:"ElementSetSpecs"引用的元素是"RootElementSetSpec"和"AdditonalElementSetSpec"(出现时)引用的元素的联合。

注7：当元素是信息客体时(即，支配者是信息客体类别)，使用如GB/T 16262.2的12.3中定义的记法"ObjectSetElements"。

46.2 形成集的元素是：

a) 如果选择"ElementSetSpecs"的第一个替换项，"Unions"中规定的那些内容[见b)]，除了"Exclusions"的"Elements"记法中规定的那些内容外，支配者的所有其他元素；

b) 如果选择"Unions"的第一个替代项，那么是"Intersections"[见c)]中规定的那些内容，在"UElems"或"Intersections"中至少规定一次；

c) 如果选择"Intersections"的第一个替代项，那么是"IntersectionElements"[见d)]中规定的那些内容，否则是也由"IntersectionElements"规定的"IElems"规定的那些内容；

d) 如果选择"IntersectionElements"的第一个替代项，则为"Elements"中规定的那些内容，除了在"Exclusions"中规定外，应在"Elems"中规定。

46.3 如果下面的条件有效，定义值集为可扩展的：

a) 对于"Elements"：在较外面的层有扩展标志；

注：即使双亲的所有值包含在新受限类型的根中这也适用。

b) 对于"Unions"：至少一个"UElems"是可扩展的；

c) 对于"Intersections"：至少一个"IElems"是可扩展的；

d) 对于"Exclusions"：EXCEPT之前的元素集是可扩展的。

否则，值的集不是可扩展的(也见G.4)。

46.4 如果值的集是可扩展的，如46.2中规定的，通过只用集算法中包含的值集的根值形成集算法能够确定根值。通过采用由扩展附加增加的根值形成值集能够确定扩展附加，对集算法中包含的每个值集，随后排除确定为根值的值。

46.5 "Elements"记法定义为：

```
Elements ::=
        SubtypeElements    |
        ObjectSetElements  |
        "("ElementSetSpec")"
```

本记法规定的元素是：

a) 如果使用"SubtypeElements"替换项，则在下面第47章描述。该记法应仅用于支配者是类型，而且包括的实际类型应进一步约束记法的可能性时。在这一上下文中，支配者称为双亲类型；

b) 如果采用"ObjectSetElements"记法，则在GB/T 16262.2的12.10中描述。该记法仅用于当支配者是信息客体类别时；

c) 如果采用第三种替换项，是"ElementSetSpec"规定的那些内容。

46.6 当支配类型是可扩展的，形成子类型约束或值集内的集算法时，集算法中只采用支配类型的扩展根中的抽象值。这时，集算法中所用的所有值记法(包括值引用)实例要求引用支配类型的扩展根的抽象值。要求范围约束的结束点引用在支配类型的扩展根中出现的值，而整个范围规范引用全部(且只有)是支配类型扩展根内范围中的那些值。

46.7 当形成含有信息客体集的集算法时，所有信息客体都用于集算法中。如果贡献给集算法的任何信息客体集是可扩展的，或如果在"ElementSetSpec"最外层有扩展标志，则集算法的结果是可扩展的。

46.8 如果子类型约束通过可扩展约束的应用连续地应用到可扩展的双亲类型上，其中用的值记法不应引用不在双亲类型的扩展根中的值。只要约束应用于没有扩展标志和可能扩展附加的双亲类型，第二个(连续应用的)约束的结果定义为是一样的。

例如：

```
Foo::= INTEGER (1..6,...,73,..80)
```

Bar::= Foo (73) -- 非法的

foo Foo ::= 73 -- 合法,因为它是 Foo 的值记法,不是约束的一部分。

由于 73 不在 Foo 的扩展根中,Bar 是非法的。如果 73 已经在 Foo 的扩展根中,示例将是合法的,而且 Bar 将约束 73 的单个值。

47 子类型元素

47.1 概述

对“SubtypeElements”提供了许多不同形式的记法。他们标识如下,而且将他们的语法和语义在下面各条中定义。表 9 概括了哪种记法适用于哪种双亲类型。

```
SubtypeElements::=
        SingleValue                |
        ContainedSubtype           |
        ValueRange                 |
        PermittedAlphabet          |
        SizeConstraint             |
        TypeConstraint             |
        InnerTypeConstraint        |
        PatternConstraint
```

表 9 子类型值集的适用性

类型(或通过置标记或分成子类型从该类型派生的)	Single Value	Contained Subtype	Value Range	Size Constraint	Permitted Alphabet	Type Constraint	InnerType Constraint	Pattern Constraint
Bit String	是	是	否	是	否	否	否	否
Boolean	是	是	否	否	否	否	否	否
Choice	是	是	否	否	否	否	是	否
Embedded-pdv	是	否	否	否	否	否	是	否
Enumerated	是	是	否	否	否	否	否	否
External	是	否	否	否	否	否	是	否
Instance-of	是	是	否	否	否	否	是	否
Integer	是	是	是	否	否	否	否	否
Null	是	是	否	否	否	否	否	否
Object class field type	是	是	否	否	否	否	否	否
Object Descriptor	是	是	否	是	是	否	否	否
Object Identifier	是	是	否	否	否	否	否	否
Octet String	是	是	否	是	否	否	否	否
Open type	否	否	否	否	否	是	否	否
Real	是	是	是	否	否	否	是	否
Relative Object Identifier	是[b]	是[b]	否	否	否	否	否	否

表 9(续)

类型(或通过置标记或分成子类型从该类型派生的)	Single Value	Contained Subtype	Value Range	Size Constraint	Permitted Alphabet	Type Constraint	InnerType Constraint	Pattern Constraint
Restricted Character String Type	是	是	是[a]	是	是	否	否	是
Sequence	是	是	否	否	否	否	是	否
Sequence-of	是	是	否	是	否	否	是	否
Set	是	是	否	否	否	否	是	否
Set-of	是	是	否	是	否	否	是	否
Time Types	是	是	否	否	否	否	否	否
Unrestricted Character StringType	是	否	否	是	否	否	是	否

[a] 只允许在 BMPString、IA5String、NumericString、PrintableString、VisibleString、UTF8String 和 UniversalString 的"PermittedAlphabet"内。

[b] 约束或值集中所有相对客体标识符类型或值的开始节点应该与支配者的开始节点相同。

47.2 单值

47.2.1 "SingleValue"记法应该是:

SingleValue::=Value

其中,"Value"是双亲类型的值记法。

47.2.2 "SingleValue"规定由"Value"规定的双亲类型的单值。

47.3 包含子类型

47.3.1 "ContainedSubtype"记法应该是:

ContainedSubtype ::= Includes Type

Includes ::= INCLUDES | empty

当"ContainedSubtype"中的"Type"是空类型的记法时,不应该使用"Includes"产生式的"empty"替换项。

47.3.2 "ContainedSubtype"规定也在"Type"根中的双亲类型的根中的所有值。要求"Type"从作为双亲类型的相同固有类型派生而来。

47.3.3 包含的子类型约束中使用的可扩展"Type"引用的值集不从"Type"中继承扩展标志。忽略不在那一类型扩展根中的"Type"中的任何值,而且,不贡献给受约束类型的值。

注:使用可扩展"Type"本身不会使受约束类型扩展。

47.4 值范围

47.4.1 "ValueRange"记法应该是

ValueRange::=LowerEndpoint ".." UpperEndpoint

47.4.2 "ValueRange"规定通过规定范围结束点的值指定的值范围内的值。这一记法仅适用于整数类型、某些受限制字符串类型(只有 IA5String、NumericString、PrintableString、VisibleString、BMPString、UniversalString 和 UTF8String)的"PermittedAlphabet"和实数类型。要求"ValueRange"中规定的所有值是在双亲类型根中。

注:为了分成子类型,PLUS-INFINITY 超过所有的实值和 MINUS-INFINITY 小于所有的实值。

47.4.3 范围的每个结束点是封闭(这时规定了那个结束点)或者是开放(这时没有规定结束点)的。在开放时,结束点规范包含一个小于符号("<"):

LowerEndpoint ::= LowerEndValue | LowerEndValue "<"

UpperEndpoint ::= UpperEndValue | "<" UpperEndValue

47.4.4 也可以不规定结束点，这时在那个方向延伸到双亲类型允许的范围：

LowerEndValue ::= Value | MIN

UpperEndValue ::= Value | MAX

注：当 ValueRange 用作"PermittedAlphabet"时，"LowerEndpoint"和"UpperEndpoint"的大小应为 1。

47.5 大小约束

47.5.1 "SizeConstraint"的记法应该是：

SizeConstraint ::= SIZE Constraint

47.5.2 "SizeConstraint"只适用于位串类型、八位位组串类型、字符串类型、单一集合类型或单一序列类型。

47.5.3 "Constraint"为被规定值的长度规定允许的整数值，并采用适用于下列双亲类型的任何约束形式：

INTEGER (0..MAX)

"Constraint"应该用"SubtypeConstraint"替代"ConstraintSpec"。

47.5.4 测量单位依赖于双亲类型，如下：

类型	测量单位
位串	位
八位位组串	八位位组
字符串	字符
单一集合	成分值
单一序列	成分值

注：本条中为确定字符串值大小规定的字符数计数应该清楚地与八位位组计数区别开来。字符计数应该根据用于类型的，特别是与引用标准、表或能在这种定义中出现的登记中的登记数有关系的字符集的定义来解释。

47.6 类型约束

47.6.1 "TypeConstraint"记法应该是：

TypeConstraint ::= Type

47.6.2 这一记法仅适用于开放类型记法，而且约束开放类型为"Type"的值。

47.7 允许字母表

47.7.1 "PermittedAlphabet"记法应该是：

PermittedAlphabet ::= FROM Constraint

47.7.2 "PermittedAlphabet"规定能用双亲串的子字母表构成的全部值。这一记法仅能用于受限制字符串类型。

47.7.3 除了那些应该用"SubtypeConstraint"替代"ConstraintSpec"外，"Constraint"是所有能适用于双亲类型(见表 9)的约束。子字母表精确地包括那些出现在"Constraint"允许的双亲串类型的一个或多个值中的字符。

47.7.4 如果"Constraint"是可扩展的，那么允许字母表约束选择的值集是可扩展的。根中的值集是"Constraint"的根允许的那些，而且扩展附加是根与"Constraint"扩展附加一起允许的那些，但不包括已经在根中的那些值。

47.8 内部子类型

47.8.1 "InnerTypeConstraint"记法应该是：

InnerTypeConstraint ::=

WITH COMPONENT SingleTypeConstraint |

WITH COMPONENTS MultiTypeConstraints

47.8.2 "InnerTypeConstraint"只规定那些满足约束出现的集合的值和/或双亲类型成分的值。除非满足明显或隐含的所有约束(见 47.8.6),否则,不规定双亲类型的值。这一记法可适用于单一集合、单一序列、集合、序列和选择类型。

注:通过 COMPONENT OF 转换来忽略适用于集合或序列类型的"InnerTypeConstraint"(见 24.4 和 26.2)。

47.8.3 对于按照单个其他(内部)类型(单一集合和单一序列)定义的类型,提供了采用子类型值规范形式的约束。它的记法是"SingleTypeConstraint":

SingleTypeConstraint ::= Constraint

"Constraint"定义单个其他(内部)类型的子类型。如果且只有每个内部值属于将"Constraint"用于内部类型获得的子类型,才规定双亲类型的值。

47.8.4 对于按照多个其他(内部)类型(选择、集和序列)定义的类型,提供了这些内部类型的许多约束。它的记法是"MultipleTypeConstraints":

```
MultipleTypeConstraints ::=
        FullSpecification |
        PartialSpecification
FullSpecification ::= "{" TypeConstraints "}"
PartialSpecification ::= "{" "..." " , " TypeConstraints "}"
TypeConstraints ::=
        NamedConstraint |
        NamedConstraint " , " TypeConstraints
NamedConstraint ::=
        Identifier ComponentConstraint
```

47.8.5 "TypeConstraints"包含对双亲类型的成分类型的约束列表。对于序列类型,约束必须按顺序出现。约束作用的内部类型用其标识符标识。对于给出的成分,最多应该有一个"NamedConstraint"。

47.8.6 "MultipleTypeConstraints"由"FullSpecification"或者"PartialSpecification"组成。当使用"FullSpecification"时,应该在能约束为缺失(见 47.8.9)且没有明显列出的所有内部类型中有"ABSENCE"的隐式出现约束。采用"PartialSpecification"时,没有隐式约束且能从列表中忽略任何内部类型。

47.8.7 特定的内部类型可以按照它的出现(在双亲类型的值中)、它的值或两者约束。记法是"ComponentConstraint":

ComponentConstraint ::= ValueConstraint PresenceConstraint

47.8.8 内部类型值的约束用记法"ValueConstraint"表示:

ValueConstraint ::= Constraint | empty

如果且只有内部值属于作用于内部类型的"Constraint"规定的子类型时,双亲类型的值满足这一约束。

47.8.9 内部类型出现的约束应该用记法"PresenceConstraint"表示:

PresenceConstraint::=PRESENT | ABSENT | OPTIONAL | empty

这些候选项的意义以及允许他们的情形在 47.8.9.1 至 47.8.9.3 中定义。

47.8.9.1 如果双亲类型是序列或集合,标记为 OPTIONAL 的成分类型可约束为 PRESENT(这时,如果也只有对应的成分值出现才满足约束)或者为 ABSENT(这时,如果也只有对应的成分值缺失才满足约束)或者为"OPTIONAL"(这时,对应的成分值出现上没有加以约束)。

47.8.9.2 如果双亲类型是选择,成份类型可约束为 ABSENT(此时,如果也只有对应的成分类型没有用在值中才满足约束),或 PRESENT(此时,如果也只有对应的成分类型用在值中才满足约束);在

"MultiTypeConstraints"中最多应该有一个 PRESENT 关键字。

注：解释示例见附录 E 的 E.4.6。

47.8.9.3 空"PresenceConstraint"的意义依赖于是采用"FullSpecification"还是"PartialSpecification"：

a) 在"FullSpecification"中，它相当于对标记 OPTIONAL 的集合或序列成分的 PRESENT 约束，且不强加进一步的约束；

b) 在"PartialSpecification"中，不强加约束。

47.9 模式约束

47.9.1 "PatternConstraint"记法应该是：

PatternConstraint::=PATTERN Value

47.9.2 "Value"应该是包含附录 A 中定义的 ASN.1 常规表达式的类型 UniversalString(或引用这种字符串)的"cstring"。"PatternConstraint"选择满足 ASN.1 常规表达式的双亲类型的那些值。整个值应该满足整个 ASN.1 常规表达式，即，"PatternConstraint"不选择其开头字符与(整个)ASN.1 常规表达式相配但包括进一步结尾字符的值。

注："Value"形式上定义为类型 UniversalString 的值，但是类型 UniversalString 和 UTF8String 的值集是相同的(见 37.16)。因此，全部相当的定义可能已经说明"Value"是类型 UTF8String 的值。

48 扩展标志

注：像一般的约束记法一样，扩展标志不影响 ASN.1 的某些编码规则，例如，基本编码规则，但是对某些有影响，例如，紧缩编码规则。它对用 ECN 定义的编码的影响由 ECN 规范确定。

48.1 扩展标志，省略号，是希望扩展附加的指示。它不说明怎样处理这种附加而是它们在解码期间不应该当错误处理。

48.2 扩展标志和例外标识符一起使用(见第 49 章)是希望的扩展附加的指示并且如果约束违规时提供标识由应用采取的行动的方法。建议该记法用在那些使用储存和向前或者任何其他中继形式的情形，以指明(例如)任何未识别的扩展附加回到可重新编码和中继的应用。

48.3 包括可扩展子类型约束、值集或信息客体集的集算法的结果在第 46 章中规定。

48.4 如果用可扩展约束定义的类型在"ContainedSubtype"中引用，则新定义的类型不继承扩展标志或任何他的扩展附加(见 47.3.3)。可通过在其"ElementsSetSpecs"的最外层包含扩展标志使新定义的类型可扩展(也见 46.3)。例如：

```
A ::= INTEGER (0..10,...,12)   -- A 是可扩展的。
B ::= INTEGER (A)              -- B 是不可扩展的且约束为 0-10。
C ::= INTEGER (A,...)          -- C 是可扩展的且约束为 0-10。
```

48.5 如果用"ElementsSetSpecs"进一步约束以可扩展约束来定义类型，则产生的类型既不继承扩展标志也不继承可能在前面的约束中出现的任何扩展附加(见 46.8)。例如：

```
A ::=INTEGER(0..10,...)   --A 是可扩展的。
B ::=A  (2..5)            --B 是不可扩展的。
C ::= A                   --C 是可扩展的。
```

48.6 不应该出现约束为缺失的集、序列或选择类型的成分，无论集、序列或选择类型是否是可扩展类型。

注：内部类型约束不影响可扩展性。

例如：

```
A::=SEQUENCE{
        a  INTEGER
        b  BOOLEAN OPTIONAL,
        ...
```

```
}
B::=A ( WITH COMPONENTS {b ABSENT}) --B是可扩展的,但是b不应该以它的任何值出现。
```

48.7 当本部分要求不同标记(见24.5至24.6、26.3和28.3)时,进行标记唯一性检查前应概念性应用下面的转换:

48.7.1 下列情况下,在扩展插入点概念性增加新元素或替换项(称作概念增加元素,见48.7.2):

a) 没有扩展标志但在模块开头隐含可扩展性,然后,增加扩展标志而且在扩展标志之后增加新元素作为第一个附加;或者

b) 在CHOICE或SEQUENCE或SET中有单个扩展标志,然后,在紧贴结束括号之前的CHOICE或SEQUENCE或SET后面增加新元素;或者

c) 在CHOICE或SEQUENCE或SET中有两个扩展标志,那么,新元素加在紧贴第二个扩展标志之前。

48.7.2 这一概念增加元素只是通过规则的应用来检查要求不同标记(见24.5至24.6、26.3和28.3)的合法性。它是在应用自动标记(如果适用的话)和COMPONENTS OF扩大之后概念增加的。

48.7.3 这一概念增加元素定义具有与所有正常ASN.1类型的标记不同的标记,但是,与所有这样的概念增加元素的标记匹配而且如GB/T 16262.2的14.2注2中规定的,与开放类型的不确定标记匹配。

注:必须有与概念增加元素和开放类型相关的标记唯一性规则以及要求不同标记的规则(见24.5至24.6、26.3和28.3)并足以保证:

a) 当BER编码被解码时,任何未知的扩展附加能无歧义地归因于单个插入点;和

b) 未知的扩展附加绝不可能与OPTIONAL元素混淆。

在PER中,上面的规则足以保证这些特性,但不是必需的。尽管如此,他们仍然用作ASN.1的规则以保证记法与编码规则独立。

48.7.4 如果要求不同类型的规则由于这些概念增加元素而违规,那么规范非法使用了可扩展性记法。

注:上面规则的目的是精确约束使用插入点(特别是不在各个SEQUENCE或SET或CHOICE尾部的那些)。设计这些约束来保证在BER、DER和CER中版本1系统收到的未知元素可能无歧义地归因于规定的插入点。如果例外处理这种附加元素对不同插入点是不同的,这样做将很重要。

48.8 示例

48.8.1 例1

```
A ::= SET{
        a    A,
        b    CHOICE{
             c    C,
             d    D,
             ...,
        }
}
```

是合法的,因为,像任何附加的材料必须是"b"的部分一样无歧义。

48.8.2 例2

```
A ::= SET{
        a    A,
        b    CHOICE{
             c    C,
             d    D,
             ...,
        },
```

```
            ...,
            d    D
    }
```

是非法的，因为，附加的材料可能是“b”的部分，或可能在“A”的较外面层，而版本 1 系统不能告诉是哪一个。

48.8.3 例 3

```
A ::= SET{
        a    A,
        b    CHOICE{
             c    C,
             ...
        },
        d    CHOICE{
             e    E,
             ...
        }
}
```

也是非法的，因为附加的材料可能是 b 或 d 的部分。

48.8.4 用可扩展选择内的可扩展选择或标志为 OPTIONAL 或 DEFAULT 的序列元素内的可扩展选择能够构成更复杂的示例，但是上面的规则是必需的而且足以保证版本 1 中不出现的元素能由版本 1 系统无歧义地归因于精确的一个插入点。

49 例外标识符

49.1 在复杂的 ASN.1 规范中，在许多地方需要特别指出解码器应处理其中没有完全规定的材料。这些情况特别出现在使用用抽象语法参数定义的约束中(见 GB/T 16262.4—2006 第 10 章)。

49.2 在这些情况下，应用设计者需要标识干扰某些依赖实现约束时采取的行动。提供例外的标识符作为引用 ASN.1 规范部分的无歧义手段以便指出采取的行动。标识符由“!”字符构成，紧接着是可选的 ASN.1 类型和那个类型的值。在没有类型时，则假设 INTEGER 是值的类型。

49.3 如果出现“ExceptionSpec”，它指明标准正文的文本中有说明怎样处理与“!”字符有关的约束干扰。如果没有，那么，实现者要么需要标识描述他们采取的行动的文本，要么当出现约束干扰时，将采取依赖实现的行动。

49.4 “ExceptionSpec”的记法定义如下：

```
ExceptionSpec::= "!" ExceptionIdentification | empty
ExceptionIdentification::=
        SignedNumber        |
        DefinedValue        |
        Type " : " Value
```

最初两个替换项指明类型整数的例外标识符。第三个替换项指明任意类型(“Type”)的例外标识符(“Value”)。

49.5 用多重约束来约束类型时，不止一个有例外标识符，在最外约束中的例外标识符应该认为是那个类型的例外标识符。

49.6 当用于集合算法的类型里出现例外标志时，应该忽略例外标识符而且不应该由被约束的类型作为集合算法的结果继承。

附 录 A
（规范性附录）
ASN.1 常规表达式

A.1 定义

A.1.1 ASN.1 常规表达式是描述其格式符合模式本身的串集的模式。常规表达式本身是串；它与算法表达式类似，通过使用各种运算符将较小的表达式组合。最小的表达式（通常）由一个或两个字符组成，是表示字符集的占位符。

这里出现的常规表达式与像 Perl 等脚本语言的那些表达式和 XML 模式的那些表达式非常相似，那里可以找到用法的某些其他示例。

A.1.2 大部分字符，包括全部字母和数字，是与它们本身匹配的常规表达式。

例如：

常规表达式“fred”只与串“fred”相匹配。

A.1.3 可以拼接两个常规表达式；产生的常规表达式与通过拼接两个分别与子表达式相匹配的子串形成的任意串相匹配。

A.2 元字符

A.2.1 元字符序列（或元字符）是一个或多个在常规表达式上下文中有特定意义的相邻字符的集合。下面的列表包含所有的元字符序列。它们的意义在下面各条中解释。

[ ]		与用“-”指明范围的集合中的任意字符匹配。 开括号之后的“^”补充紧跟其后的集合。
{g,p,r,c}		标识 GB/T 13000.1 字符的四元组（见 37.8）
\N{名称}		如 38.1 中定义，与已命名字符（或已命名字符集的任意字符）匹配
.		与任意字符匹配（除非它是 11.1.6 中定义的换行字符之一）
\d		与任何数字匹配（相当于“[0～9]”）
\w		与任何字母数字字符匹配（相当于“[a～zA～Z0～9]”）
\t		与 HORIZONTAL TABULATION(9)字符匹配（见 11.1.6）
\n		与 11.1.6 中定义的任意一个换行字符匹配
\r		与 CARRIAGE RETURN(13)字符匹配（见 11.1.6）
\s		与任意一个空白字符匹配（见 11.1.6）
\b		与单词边界匹配
\	（前缀）	引用文本元字符并使它从字面上解释
\\		与 REVERSE SOLIDUS(92)字符“\”匹配
" "		与 QUOTATION MARK(34)字符(")匹配
|	（中缀）	两个表达式之间的替换项
()		封闭表达式的分组
*	（后缀）	与前面表达式匹配 0、1 或几次
+	（后缀）	与前面表达式匹配 1 或几次
?	（后缀）	与前面表达式匹配 1 或一次也没有
#n	（后缀）	与前面表达式准确地匹配 n 次（n 是单个数字）
#(n)	（后缀）	与前面表达式准确地匹配 n 次

#(n,)	(后缀)	与前面表达式匹配至少 n 次
#(n,m)	(后缀)	与前面表达式匹配至少 n 次但不超过 m 次
#(,m)	(后缀)	与前面表达式匹配不超过 m 次

注 1：字符 CIRCUMFLEX ACCENT(94)"^"和 HYPHEN-MINUS(45)"-"是 A.2.2 中定义的串中某些位置上的附加元字符。

注 2：本附录中字符名称后的圆括号内的值是 GB/T 13000.1 中字符的十进制值。

注 3：该记法不提供元字符"^"和"$"来分别与串的开始和结尾匹配。因此，串应该整体上与常规表达式相匹配，除如果后者在它的开头、结尾或者两端包括".*"之外。

注 4：除非空白出现在紧接换行之前或之后，否则，下面元字符序列不能包含空白(见 11.1.6)：

```
{g,p,r,c}
\N{名称}
#n
#(n)
#(n,)
#(n,m)
#(,m)
```

如果常规表达式包含换行、紧接换行之前或之后出现的任何间隔字符没有意义且与什么也不匹配(见 11.14.1)。

A.2.2 用"["和"]"封闭的字符列表与该列表中的任意单个字符相匹配。如果列表中的第一个字符是脱字符号"^"，那么它与不在列表中的任意字符匹配。通过给出第一个和最后一字符，用连字符隔开(根据 39.3 中定义的顺序关系)可以规定字符的范围。除"]"和"\"外，所有元字符序列失去它们在列表内的特殊意义。为了包括文字 CIRCUMFLEX ACCENT(94)"^"，将它放在除第一个位置外的任何位置或在它前放置反斜线。为了包括文字 HYPHEN-MINUS(45)"-"，将它放在列表的最前或最后，或在它前放置反斜杠。为了包括文字 CLOSING SQUARE BRACKET(93)"]"，将它放在最前面。如果列表中的第一个字符是脱字符号"^"，那么当它们紧接在那个脱字符号之后时，字符"-"和"]"也与它们自身相匹配。A.2.3、A.2.4、A.2.6 和 A.2.7 中定义的元字符序列能用于它们保持其意义的方括号之间。

例如：

常规表达式"[0123456789]"，或相当的"[0～9]"与任何单个数字相匹配。

常规表达式"[^0]"与除 0 之外的任何单个字符相匹配。

常规表达式"[\d^.-]"与任何单个数字、脱字符号、连字符或句号相匹配。

A.2.3 为了避免有相同图形字符的两个 GB/T 13000.1 字符之间的任何歧义，提供了两个记法。形式"{组、面、行、字符元}"的记法引用按照 37.8 中定义的"四元组"产生式的(单个)字符。

A.2.4 如果"valuereference"是引用当前模块中定义或引入的大小为 1(见第 37 章)的受限制字符串值，则形式"\N{ valuereference }"的记法与引用的字符相匹配。如果"typereference"是引用当前模块中定义的"RestrictedCharaterStringType"的子类型或第 37 章中定义的"RestrictedCharaterStringType"之一，则形式"\N{ typereference }"的记法与引用的字符集的任何字符相匹配。

注：在特定情况下，"valuereference"或"typereference"可能是模块 ASN1-CHARACTER-MODULE(见 38.1)中定义和引入到当前模块(见 37.8)的引用之一。

例如：

常规表达式"\N{greekCapitalLetterSigma}"与 GREEK CAPITAL LETTER SIGMA 相匹配。

常规表达式"\N{BasicLatin}"与 BASIC LATIN 字符集的任意(单个)字符相匹配。

"[\N{BasicLatin}\N{Cyrillic}\N{BasicGreek}]+"，或相当的"(\N{BasicLatin} | \N{Cyrillic} | \N{BasicGreek})+"是与任何取自规定的三个字符集字符的(非空)数字构成的串相匹配的常规表达式。

A.2.5 句号"."除非是 11.1.6 中定义的换行字符之一，否则，它与任意单个字符相匹配。

A.2.6 符号"\d"与"[0-9]"同义，即：它与任意单个数字相匹配。符号"\t"与字符 HORIZONTAL

TABULATION(9)相匹配。符号"\w" 与"[a～zA～Z0～9]"同义,即:它与任意单个(大写或小写)字符或任意单个数字相匹配。

例如:

常规表达式"\w+(\s\w+)*\."与至少一个(字母数字)词相匹配。该词用11.1.6中定义的一个空白字符分隔。结束句号前没有空白字符。

A.2.7 符号"\r"与CARRIAGE RETURN(13)相匹配。符号"\n"与11.1.6中定义的任意一个换行字符相匹配。符号"\s"与11.1.6中定义的任意一个空白字符相匹配。符号"\b"与词开始或结束的空串相匹配。

例如:

常规表达式".*\bfred\b.*"与包含词"fred"(该词不仅仅是一串四个字符;它是划界了的)的任意串相匹配。因此,它与"fred"或"I am fred the first"等类似的串相匹配,但不与"My name is freddy"或"I am afred I don't know how to spell 'afraid'!"等类似的串相匹配。

A.2.8 通常用作元字符的字符能通过用"\"作其前缀来字面说明。如果常规表达式包含QUOTATION MARK(34),该字符应该用一对QUOTATION MARK字符表示。

例如:

常规表达式"\."与(单个)串"."相匹配,但不与任何单个字符的任何串相匹配。

常规表达式""""与包含单个QUOTATION MARK的串相匹配。

常规表达式"\)"与串")"相匹配。

常规表达式"\a"与字符"a"相匹配。

注:第四个示例表明,允许反斜线在非元字符的字符前,但是不赞成这种用法(因为在本部分将来的版本中会允许其他元字符)。

A.2.9 通过中缀操作符"|"可以合并两个或多个常规表达式。产生的常规表达式与每个子表达式相匹配的任意串相匹配。

A.2.10 常规表达式可后接重复操作符。如果操作符是"?",则前面的项是可选的而且最多匹配一次。如果操作符是"*",则前面的项将匹配零或多次。如果操作符是"+",则前面的项将匹配一次或多次。如果操作符是"#(n)"的形式,则前面的项准确地匹配n次;在这一特定情况下,如果n由一位数字组成,则可以省略圆括号。如果它是"#(n,)"的形式,则项匹配n或更多次。如果它是"#(,m)"的形式,则项是可选的而且匹配最多m次。如果它是"#(n,m)"的形式,则项匹配至少n次,但不超过m次

注:将元字符"*"、"+"、"?"或"#"用作常规表达式的第一个字符是非法的。将元字符"#"或"|"用作常规表达式的最后一个字符也是非法的。

例如:,

"555-1212"这样的电话号码是通过常规表达式"\d#3-\d#4"来相匹配的,或者相当于"\d#(3)-\d#(4)"。

"$12345.90"这样的美元价格是通过常规表达式"$\d#(1,)(\.\d#(1,2))?"来相匹配的。注意,当"#"符号后接范围时要求有圆括号。

"123-45-5678"这样的社会保险号码是通过常规表达式"\d#3-? \d#2-? \d#4"来相匹配的。

A.2.11 重复(见A.2.10)优先于拼接(见A.1.3),而按顺序优先于交替(见A.2.9)。全部子表达式可包围在圆括号中以打破这些优先规则。

A.2.12 当常规表达式包含圆括号中的子表达式时,每个(非引用的)开圆括号从常规表达式的左边到右边连续地赋予不同的(严格地讲正)整数。然后每个子表达式能够被引用到有像使用相关整数的"\1"、"\2"等记法的注解中去。不允许空的子表达式"()"。

例如:

"((\d#2)(\d #2)(\d#4))" --\1是日期而\2是月、\3是星期几和\4是年。

注:对形式引用常规表达式的子表达式在许多情况下有要求。一个这样的实例是需要写出文本将ASN.1模块中的常规表达式归档。这是能用来提供这种引用的记法。该记法不在本部分中的其他地方使用。

附 录 B
（规范性附录）
类型和值兼容的规则

本附录希望主要为工具建造者使用以保证他们能相同地解释语言。本附录的目的是为了清楚地规定什么是合法的 ASN.1 和什么是不合法的 ASN.1，而且能够规定任意值引用名标识符的精确值，以及任意类型或值集引用名标识符的精确值集。不准备用它来提供任何上面描述以外目的的 ASN.1 记法的有效转换的定义。

B.1 需要值映射概念（辅导介绍）

B.1.1 考虑下面 ASN.1 定义：

```
A ::= INTEGER
B ::= [1] INTEGER
C ::= [2] INTEGER (0..6,...)
D ::= [2] INTEGER (0..6,...,7)
E ::= INTEGER (7..20)
F ::= INTEGER {red(0), white (1), blue(2), green(3), purple(4)}
a A ::= 3
b B ::= 4
c C ::= 5
d D ::= 6
e E ::= 7
f F ::= green
```

B.1.2 很清楚，值引用 a、b、c、d、e 和 f 能用在分别由 A、B、C、D、E 和 F 支配的值记法中。例如，

```
W ::= SEQUENCE { w1   A   DEFAULT a}
```

和

```
x A ::=a
```

和

```
Y ::= A(1..a)
```

都是 B.1.1 中给出定义的有效值。然而，如果上面的 A 用 B、或 C、或 D、或 E、或 F 替代，产生的语句将合法吗？同样，如果上面的值引用 a 在每一个这种情况下用 b、或 c、或 d、或 e、或 f 替代，产生的语句合法吗？

B.1.3 更复杂的问题将考虑在每种情况下用赋值右边的显式文本来代替类型引用。考虑下边的示例：

```
f INTEGER {red(0), white (1), blue(2), green(3), purple(4)} ::= green
W ::= SEQUENCE {
    w1 INTGER {red(0), white (1), blue(2), green(3), purple(4)} DEFAULT f}
x INTEGER {red(0), white (1), blue(2), green(3), purple(4)} ::= f
Y ::= INTEGER {red(0), white (1), blue(2), green(3), purple(4)} (1..f)
```

上面的是合法的 ASN.1 吗？

B.1.4 某些上面的示例即使是合法（大部分是——见后面的文本）的情况，将奉劝用户不要书写类似的文本，因为它们至少有些模糊和最容易混淆。然而，经常会把某些类型值（不必正好是 INTEGER 类

型)的值引用用作为带有应用于支配者中的置标记或分成子类型的那个类型的默认值。引入*值映射*概念是为了提供确定上面哪种构造是合法的清楚和精确手段。

B.1.5 再考虑:

C ::= [2] INTEGER (0..6,...)

E ::= INTEGER (7..20)

F ::= INTEGER {red(0), white (1), blue(2), green(3), purple(4)}

在每种情况下产生新的类型。对于F,我们能在它的值与通用类型INTEGER的值之间清楚地1-1对应标识。在C和E的情况下,我们能在它们的值与通用类型INTEGER的值的子集之间清楚地1-1对应标识。我们把这种关系称作两个类型中的值之间的*值映射*。而且,因为F、C和E中的值都与INTEGER值的(1-1)映射,我们能用这些映射提供F、C和E它们本身值之间的映射。对F和C,在图B.1中示出。

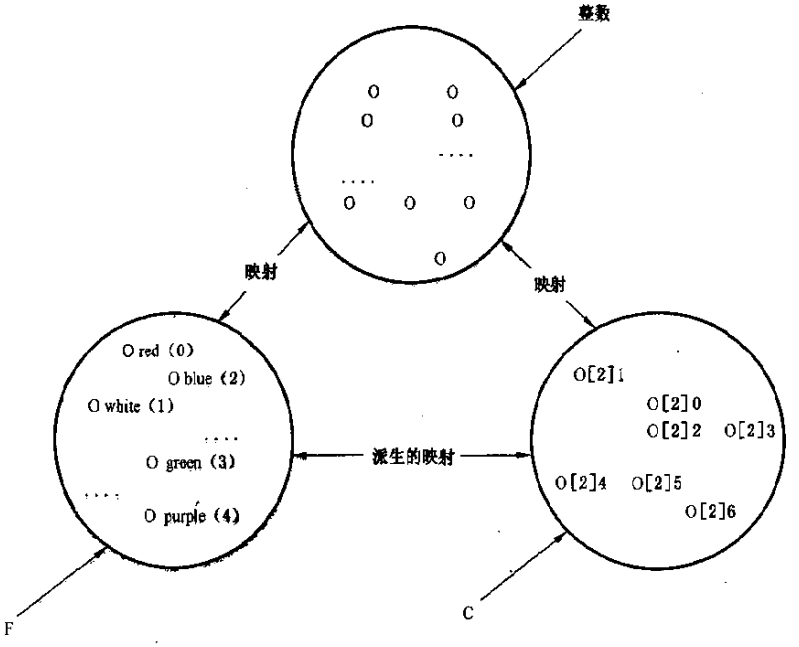

图 B.1

B.1.6 现在当我们有下列值引用:

c C ::= 5

对于在某些上下文中要求用C中的值来标识F中值,那么,只要C中的那个值和F中的(单个)值存在值映射,我们能(且确实)定义c是F中值的合法引用。这在图B.2中示出,其中值引用c用来标识F中的值,而且能用在我们应该另外必须定义下列直接引用f1的地方:

f1 F ::= 5

B.1.7 应该注意,在某些情况下,在一个类型中应有值(例如,B.1.1的A中的7至20)与另一个类型中的值(例如,B.1.1的E中的7至20)有值映射,但是其他值(A的21以上)没有这种映射。A中这类值的引用不会提供E中值的有效引用。(在本例中,整个E与A的子集有值映射。在一般情况下,在有值映射的两种类型中可能都有值的子集,其他值在没有映射的两种类型中。)

B.1.8 在ASN.1中标注,用正常的英语文本来规定上面和类似情况中的合法性。B.6章给出合法性的精确要求并且应该在对复杂结构有疑虑时引用。

注:"Type"结构的两个事件之间定义存在值映射的事实允许采用(使用一个"Type"结构建立的)值引用来标识另一

个充分相似的"Type"结构中的值。它允许用两个文本隔开、没有与虚拟和实际参数兼容性规则相冲突的"Type"结构来将虚拟和实际参数归类。也允许用一个"Type"结构来规定信息客体类别的范围和用充分相似的、不同"Type"结构规定信息客体中的对应值。(这些示例不是无遗漏的。)然而,建议利用这一自由的优点只是为了简单的情形,如,SEQUENCE OF INTEGER,或 CHOICE{int INTEGER, id OBJECT IDENTIFER},而不能用于更复杂的"Type"结构。

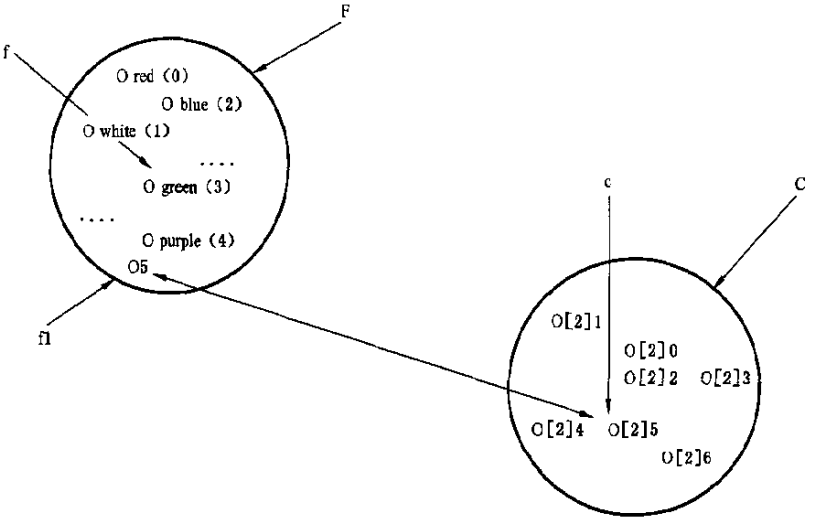

图 B.2

B.2 值映射

B.2.1 基础的模型是类型,像非重叠的容器,包含值,有定义不同新类型(见图 B.1 和 B.2)的每个 ASN.1"Type"结构事件。本附录规定当这些类型间存在*值映射*,以及在需要引用某些其他类型中的值时,可使用引用一个类型中的值。

例如:考虑:

X ::= INTEGER

Y ::= INTEGER

X 和 Y 是两个不同类型的类型引用名(指针),但是,这些类型间存在值映射,因此,当由 Y 支配时(例如,紧跟 DEFAULT 后面),可采用 X 值的任意值引用。

B.2.2 在所有可能的 ASN.1 值集中,值映射把一对值关联起来。值映射的整个集合是数学关系。这个关系具有下面的特性:它是反身的(每个 ASN.1 值与本身有关),对称的(如果从值 x1 到值 x2 定义存在值映射,那么从 x2 到 x1 自动存在值映射),而且,它是可传递的(如果从值 x1 到值 x2 有值映射,而且从值 x2 到值 x3 有值映射,那么,从 x1 到 x3 自动存在值映射)。

B.2.3 另外,给出任意两个类型 X1 和 X2,视为值集,从 X1 中的值到 X2 中的值的值映射集是一一对应关系,即,对于 X1 中所有的值 x1、X2 中的 x2,如果从 x1 到 x2 有值映射,那么:

a) 从 x1 到 X2 中另一个不同于 x2 的值没有值映射;和

b) 从 X1 中的任意值(非 x1)到 x2 没有值映射。

B.2.4 在值 x1 和值 x2 之间存在值映射时,如果某些支配类型这样要求的话,能自动地采用任一个的值引用。

注:某些"Type"结构中的两个值之间定义了存在值映射的事实仅仅为了提供使用 ASN.1 记法的灵活性。这些映射的存在不携带暗示两种类型携带相同的应用语义,但是,建议只有对应的类型确实携带相同的应用语义,才使用无值映射就是非法的 ASN.1 结构。注意,值映射将经常存在于同一 ASN.1 结构中的两个类型之间任意

的大规范中，但是，携带完全不同应用语义，而且这些值映射的存在从不用于确定整个规范的合法性。

B.3 相同类型定义

B.3.1 用相同类型定义的概念来使值映射能在相同或足够相似到将正常希望它们的使用可交换的“Type”的两个实例间定义。为了给出“足够相似”的精确意义，本条规定一系列应用于每个“Type”实例的转换为那些“Type”实例产生*正常形式*。如果(且只有)两个“Type”实例，当它们的正常形式是相同词项的相同有序列表时，则定义两个“Type”实例是相同类型定义(见第 11 章)。

B.3.2 在 ASN.1 规范中的每个“Type”事件是第 11 章中定义的词项的有序列表。通过以那样的顺序应用 B.3.2.1 至 B.3.2.6 中定义的转换可获得正常形式。

B.3.2.1 删去所有的注解(见 11.6)。

B.3.2.2 下面的转换不递归，因此，只需要以任意顺序应用一次：

a) 对于由“ValueSetTypeAssignment”定义的类型，其定义由“TypeAssignment”用如 15.6 中规定的“ValueSet”内容的、相同的“Type”和子类型约束代替；

b) 对于每个整数类型：“NamedNumberList”(见 18.1)，如果有的话，重新排序以便“identifier”按字母表顺序排列(“a”开头，“z”最后)；

c) 对于每个枚举类型：如 19.3 中的规定，数字增加到是“identifier”(没有数字)的任意“EnumerationItem”(见 19.1)；然后，“RootEnumeration”重新排序以便“identifier”按字母表顺序排列(“a”开头，“z”最后)；

d) 对于每个 bitstring 类型：“NamedBitList”(见 21.1)，如果有的话，重新排序以便“identifier”按字母表顺序排列(“a”开头，“z”最后)；

e) 对于每个客体标识符值：每个“ObjIdComponent”转换成它对应的、符合第 31 章语义的“NumberForm”(见 31.12 中的示例)；

f) 对于每个相对客体标识符值(见 32.3)：每个“RelativeOIDComponent”转换成它对应的、符合第 32 章语义的“NumberForm”；

g) 对于序列类型(见第 24 章)和集合类型(见第 26 章)：形式“ExtensionAndException”、“ExtensionAdditions”的任何扩展被切割并粘贴到“ComponentTypeLists”的尾部；删去“OptionalExtensionMarker”，如果出现的话。

如果“TagDefault”是 IMPLICIT TAGS，关键字 IMPLICIT 被加到“Tag”的所有实例上(见第 30 章)，除非有下面情况之一：

- 它已经出现；或
- 出现保留字 EXPLICIT；或
- 被标记的类型是 CHOICE 类型；或
- 它是开放类型。

如果“TagDefault”是 AUTOMATIC TAGS，根据 24.2 来决定是否应用自动标记(自动加标记将在后面进行)。

注：在 24.3 和 26.2 中规定，作为替换“Component of Type”的结果插入的“ComponentType”中出现的“Tag”本身不阻止自动加标记转换。

如果“ExtensionDefault”是 EXTENSIBILITY IMPLIED，若没有出现省略号(“...”)的话，则在“ComponentTypeLists”后加上；

h) 对于选择类型(见第 28 章)：“RootAlternativeTypeList”重新排序以便“NameType”的标识符按字母表顺序排列(“a”开头，“z”最后)。“OptionalExtensionMarker”，如果出现的话，被删去。如果“TagDefault”是 IMPLICIT TAGS，关键字 IMPLICIT 被加到“Tags”的所有实例上(见第 30 章)，除非有下面情况之一：

- 它已经出现;或
- 出现保留字 EXPLICIT;或
- 被标记的类型是 CHOICE 类型;或
- 它是开放类型。

如果"TagDefault"是 AUTOMATIC TAGS,根据 28.5 来决定是否应用自动标记(自动加标记将在后面进行)。如果"ExtensionDefault"是 EXTENSIBILITY IMPLIED,若没有出现省略号("...")的话,则在"ComponentTypeLists"后加上。

B.3.2.3 以规定的顺序递归地应用下面的转换直至达到固定点:

a) 对每个客体标识符值(见 31.3):如果值定义以"DefinedValue"开头,则"DefinedValue"用它的定义代替;

b) 对每个相对客体标识符值(见 32.3):如果值定义包含"DefinedValue",则"DefinedValue"用它们的定义代替;

c) 对于序列类型和集合类型:按照第 24 章和 26 章转换所有的"COMPONENT OF Type"实例(见第 24 章);

d) 对于序列、集合和选择类型:如果它较早就决定自动加标记(见 B.3.2.2 g)和 h)),根据第 24 章、26 章和 28 章应用自动加标记;

e) 对于精选类型:结构按照第 29 章用选择的替换项代替;

f) 按照下面的规则,所有类型引用由它们的定义代替:

- 如果代替的类型是引用正被转换的类型,类型引用用只与除某项本身外无其他项相匹配的特定项代替;
- 如果代替的类型是单一序列类型或单一集合类型,约束紧接在被替代类型的后面,如果有的话,移到关键字 OF 的前面;
- 如果被代替的类型是参数化类型或参数化值集合(见 GB/T 16262.4 的 8.2),则每个"DummyReference"用对应的"ActualParameter"替代。

g) 所有的值引用由它们的定义替代;如果被替代的值是参数化值(GB/T 16262.4 的 8.2),则每个"DummyReference"用对应的"ActualParameter"替代。

注:替代任意值引用之前,应该应用本附录的规程以保证值引用通过值映射或直接地标识其支配类型中的值。

B.3.2.4 对于集合类型:"RootComponentTypeList"重新排序以便"ComponentType" 按字母表顺序排列("a"开头,"z"最后)。

B.3.2.5 应该将下面的转换应用到值定义上:

a) 如果用标识符定义整数值,则用相关的数字替代那个标识符;

b) 如果用标识符定义位串值,则用删去所有结尾 0 位的对应"bstring"替代;

c) 删去"cstring"中每个紧连换行(包括换行)前后的所有空白;

d) 删去"bstring"和"hstring"中的所有空白;

e) 规格化用底数 2 定义的每个实数值以使尾数是奇数,规格化用底数 10 定义的每个实数值以使尾数最后的数字不是 0;

f) 每个 GeneralizedTime 和 UTCTime 值用符合 DER 和 CER 中编码时采用的规则串替代(见 GB/T 16263.1 的 11.7 和 11.8);

g) 应用 c)之后,每个 UTF8String、NumericString、PrintableString、IA5String、VisibleString (GB/T 1988String)、BMPString 和 UniveraslString 值用以"四元组"记法书写的 UniversalString 类型的相当值替代(见 37.8)。

B.3.2.6 任何"realnumber"事件应转换成以 10 为"base"的相关"SequenceValue"。任何与"Se-

quenceValue"相关的"RealValue"事件应转换成相同"base"的相关"SequenceValue",以使尾数的最后数字不是0。

B.3.3 如果两个"Type"实例,当转换成它们的正常形式时,是词项的相同列表(见第11章),那么,定义两个"Type"实例为相同类型定义,但下列除外:如果"objectclassreference"(见GB/T 16262.2的7.1),"objectreference"(见GB/T 16262.2的7.2)或"objectsetreference"(见GB/T 16262.2的7.3)出现在"Type"的规格化形式内,那么,这两个类型没有定义为相同类型定义,而且它们之间将不存在值映射(见B.4)。

注:插入该例外是避免需要为关于信息客体类别、信息客体和信息客体集记法的语法元素提供正常形式的转换规则。类似地,这时还不包括规格化所有值记法和集算法记法的规范。如果要证明是该规范的需求,可在本部分的将来版本中提供。引入相同类型定义和值映射的概念以保证可以通过采用引用名或通过复制文本来使用简单的ASN.1结构。不必为更复杂的包括信息客体类别等的"Type"实例提供这一功能。

B.4 值映射规范

B.4.1 如果"Type"的两个事件是B.3规则下的相同类型定义,那么,一个类型的每个值与另一个类型对应值之间存在值映射。

B.4.2 对于通过加标记(见第30章),从任意类型X2产生的类型X1,X1的所有成员与X2的对应成员之间定义为存在值映射。

注:当上面B.4.2中X1与X2的值之间,以及B.4.3中X3与X4的值之间定义存在值映射时,如果这种类型嵌入到其他方面相同但不同类型定义(如,SEQUENCE或CHOICE类型定义)中,产生的类型定义(SEQUENCE或CHOICE类型)将不是相同类型定义,而且它们之间没有值映射。

B.4.3 对于通过元素集结构或通过分成子类型、并通过从任意支配类型X4选择值产生的类型X3,新类型成员和由元素集或分成子类型结构选择的支配类型的那些成员之间被定义为存在值映射。有无扩展标志出现不影响这一规则。

B.4.4 在B.5中规定了某些字符串类型之间的附加值映射。

B.4.5 定义为已命名值的整数类型的任意类型的所有值与定义为无已命名值、或有不同已命名值、或已命名值不同名称、或两者都有的任意整数类型之间被定义为存在值映射。

注:值映射的存在不影响使用已命名值名称的任何范围规则要求。它们只能用于由在其中定义了它们的类型、或那个类型的类型引用名支配的范围中。

B.4.6 定义为已命名位的位串类型的任意类型的所有值与定义无已命名位、或有不同已命名位、或已命名位不同名称、或两者都有的任意位串类型之间被定义为存在值映射。

注:值映射的存在不影响使用已命名位名称的任何范围规则要求。它只能用于由在其中定义了它们的类型、或那个类型的类型引用名支配的范围中。

B.5 为字符串类型定义的附加值映射

B.5.1 有两组受约束的字符串类型,A组(见B.5.2)和B组(见B.5.3)。A组中所有类型之间被定义为存在值映射,而且当由其他类型之一支配时,可以使用这些类型值的值引用。对于B组中的类型,这些不同类型之间从不会存在值映射,A组中的任意类型和B组中的任意类型之间也不会存在值映射。

B.5.2 A组组成成分:

UTF8String

NumericString

PrintableString

IA5String

VisibleString(GB/T 1988String)

UniversalString

BMPString

B.5.3 B组组成成分：

TeletexString(T61String)

VideotexString

GraphicString

GeneralString

B.5.4 将每种类型的字符串值与 UniversalString 映射来规定 A 组中的值映射，然后采用值映射的传递性特性。为了将来自 A 组类型之一的值映射到 UniversalString，串用相同长度、每个字符按如下规定映射的 UniversalString 替代。

B.5.5 形式上，UTF8String 中的抽象值集是存在于 UniversalString 中但带不同标记的抽象值的相同集(见 37.16)，而且 UTF8String 中的每个抽象值被定义为与 UniversalString 中对应的抽象值映射。

B.5.6 用于形成类型 NumericString 和 PrintableString 的字符的图形字符(打印的字符形状)与分配给 GB/T 13000.1 的最初 128 个字符的图形字符的子集有可识别和无歧义的映射。使用图形字符的这种映射来定义这些类型的映射。

B.5.7 通过将每个字符映射成具有如 IA5String 和 VisibleString 的 BER 编码值(8 位)的 UniversalString 的 BER 编码中的相同(32 位)值的 UniversalString 字符来把 IA5String 和 VisibleString 映射成 UniversalString。

B.5.8 BMPString 形式上是 UniversalString 的子集，而且对应的抽象值有值映射。

B.6 特定类型和值兼容性要求

本章使用值映射概念来为某些 ASN.1 结构的合法性提供精确的文本。

B.6.1 具有支配类型 Y 的任何“Value”事件 x-记法，标识在支配类型 Y 中的值 y-val，该类型有一个到由 x-记法规定的值 x-val 的值映射。要求有这样的值存在。

例如，考虑下面最后一行中的 x 事件：

```
X ::=[0] INTEGER (0..30)
x X::=29
Y::=[1] INTEGER (25..35)
Z1::=Y(x | 30)
```

这些 ASN.1 结构是合法的，在最后的赋值中 x-记法 x 引用 X 中的 x-val 29，并且通过值映射标识 Y 中的 y-val 29。x-记法 30 引用 Y 中的 y-val 30，同时 Z1 是值 29 和 30 的集合。另一方面，赋值：

```
Z2 ::= Y (x | 20)
```

是非法的，因为没有 x-记法 20 能引用的 y-val。

B.6.2 任何“Type”事件，具有支配类型 V 的 t-记法，标识与“Type”类型 t-记法根中任意值有值映射的支配类型 V 根中值的完整集合。要求该集合至少含有一个值。

例如，考虑下面最后一行中的 W 事件：

```
V::=[0]INTEGER (0..30)
W::=[1]INTEGER (25..35)
Y::=[2]INTEGER (31..35)
Z1::=V (W | 24)
```

W 贡献值 25-30 给集算法导致 Z1 具有值 24-30。另一方面，赋值：

```
Z2::=V (Y | 24)
```

是非法的，因为 Y 中没有与 V 中值映射的值。

B.6.3 作为实际参数提供的任何值的类型要求从那个值到支配虚拟参数的类型中的值之一有值映

射，而且，它是那个被标识的支配类型的值。

B.6.4 如果“Type”作为是值集虚拟参数的虚拟参数的实际参数提供，那么那个“Type”的所有值要求与值集虚拟参数的支配者中的值具有值映射。实际参数选择与“Type”有映射的支配者中值的全部集。

B.6.5 在规定是值或值集参数的虚拟参数的类型 A 中，除非对 A 的所有值及对赋值右边使用 A 的每个实例，A 的那个值能合法地应用于虚拟参数的地方，否则，它是非法的规范。

B.7 示例

B.7.1 本章提供说明 B.3 和 B.4 的示例。

B.7.2 示例 1

```
X::= SEQUENCE          X1::=SEQUENCE
    {name VisibleString,        { name VisibleString,
    age INTEGER          -- 注解 -
                                  age INTEGER
X2::= [8] SEQUENCE     X3::=SEQUENCE
    { name VisibleString,       { name VisibleString,
    age INTEGER          age AgeType}
                      AgeType::= INTEGER
```

X、X1、X2 和 X3 都是相同类型定义。不能看到空白和注解的差别，X3 中使用 AgeType 类型引用也不影响类型定义。然而，注意，如果序列元素的任何标识符改变，则类型将不再是相同定义，而且它们之间将没有值映射。

B.7.3 示例 2

```
B::=SET                B1::=SET
    { name VisibleString,     {age INTEGER,
    age INTEGER}              name VisibleString }
```

它们是相同类型定义并且限定于不是在模块头中有 AUTOMATIC TAGS 的模块中提供，否则它们不是相同类型定义，而且它们之间将没有值映射。能用 CHOICE 和 ENUMERATED 写出类似的示例(用“EnumerationItem”的“identifier”形式)。

B.7.4 示例 3

```
C::=SET                C1::=SET
    { name [0] VisibleString,   { name VisibleString,
    age INTEGER}              age INTEGER (1..64)}
```

它们不是相同类型定义，它们中的任一个与 B 或 B1 中的任一个也不是相同类型定义，而且 C 和 C1 的任何值之间没有值映射，它们中的任一个和 B 或 B1 中的任一个之间也没有值映射。

B.7.5 示例 4

```
x  INTEGER { y (2)} ::= 3
z  INTEGER ::= x
```

是合法的，并且通过 B.4.5 中定义的值映射把值 3 赋给 z 。

B.7.6 示例 5

```
b1 BIT STRING ::= '101' B
b2 BIT STRING ::={version 1(0),version 2(1),version 3(2)}::=b1
```

它们是合法的，并且把值{ version 1,version 3}赋给 b2。

B.7.7 示例 6

根据 B.1.1 的定义，形式：

X DEFAULT y

的 SEQUENCE 元素是合法的，其中 X 是 A、B、C、D、E 或 F 中的任一个，或者是这些名称类型赋值右边的任意文本，而 y 是 a、b、c、d、e 或 f 中的任一个，下面例外：E DEFAULT y 对于所有的 a、b、c、d、f 是非法的，而且，C DEFAULT e 是非法的，因为在这些情况下，默认值引用到被默认的类型没有出现值映射。

附 录 C
（规范性附录）
指派的客体标识符值

本附录记录 ASN.1 系列标准中所赋的客体标识符和客体描述符值，并提供 ASN.1 模块以便在引用这些客体标识符值时使用。

C.1 本部分中指派的客体标识符

本部分中赋予下面的值：

37.3 条：

Object Identifer Value：

{joint-iso-itu-t asn1(1) specification(0) characterStrings(1) numerricString(0)}

Object Descriptor Value： " NumericString ASN.1Type"

37.5 条：

Object Identifer Value：

{joint-iso-itu-t asn1(1) specification (0) characterStrings (1) printableString(1)}

Object Descriptor Value： " PrintableString ASN.1Type"

38.1 条：

Object Identifer Value：

{joint-iso-itu-t asn1(1) specification(0) modules(0) iso10646(0)}

Object Descriptor Value：" ASN.1 Character Module"

第 C.2 章：

Object Identifer Value：

{joint-iso-itu-t asn1(1) specification(0) modules(0) object-identifier(1)}

Object Descriptor Value： " ASN.1 Object Identifer Module"

C.2 ASN.1 中的客体标识符及其编码规则标准

本章规定对 ASN.1 标准(GB/T 16262.1 至 GB/T 16262.4、GB/T 16263.1 至 GB/T 16263.2，以及 ISO/IEC 8825-3 至 ISO/IEC 8825-4)中定义的每个客体标识符值包含的值引用名定义的 ASN.1 模块。

注：这些值可用于 OBJECT IDENTIFIER 类型和其衍生的类型的值记法中。本章规定的模块中定义的全部值引用可输出和必须通过希望使用它们的任何模块引入。

```
ASN1-Object-Identifier-Module { joint-iso-itu-t asn1(1) specification(0) modules(0)object-
idenfitier(1)}
    DEFINITIONS::= BEGIN
    -- NumericString ASN.1 类型(见 37.3)--
    numericString OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1) specification(0) characterStrings(1)numericString(0)}
    -- PrintableString ASN.1 类型(见 37.5)--
    printableString OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1) specification(0)characterStrings(1)printableString(1)}
    -- ASN.1Character Module(见 38.1)--
```

```
    asn1CharacterModule OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1)  specification(0)modules(0)iso10646(0)}
    -- ASN.1Object Identifer Module (本模块)--
    asn1ObjectIdentiferModule OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1 (1)  specification(0)modules(0)object-identifers(1)}
    -- 单个 ASN.1 类型的 BER 编码--
    ber OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1)  basic-encoding(1)}
    -- 单个 ASN.1 类型的 CER 编码--
    cer OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1)  ber-derived(2)canonical-encoding(0)}
    -- 单个 ASN.1 类型的 DER 编码--
    der OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1)  ber-derived(2)distinguished-encoding(1)}
    -- 单个 ASN.1 类型的 PER 编码(基本对齐)--
    perBasicAligned OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1)  packed-encoding(3)basic(0)aligned(0)}
    -- 单个 ASN.1 类型的 PER 编码(基本非对齐)--
    perBasicUnaligned OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1)  packed-encoding(3)basic(0)unaligned(1)}
    -- 单个 ASN.1 类型的 PER 编码(正则对齐)--
    perCanonicalAligned OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1)  packed-encoding(3)canonical(1)aligned(0)}
    -- 单个 ASN.1 类型的 PER 编码(正则非对齐)--
    perCanonicalUnaligned OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1)  packed-encoding(3)canonical(1)unaligned(1)}
    -- 单个 ASN.1 类型的 XER 编码(基本)--
    xerBasic OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1)  xml-encoding(5)basic(0)}
    -- 单个 ASN.1 类型的 XER 编码(正则)--
    xerCanonical OBJECT IDENTIFIER ::=
        {joint-iso-itu-t asn1(1)  xml-encoding(5)正则(1)}
END -- ASN1-Object-Identifier-Module --
```

附 录 D
（资料性附录）
给客体标识符成分赋值

本附录描述客体描述符注册树的顶级弧。没有解释怎样增加新弧，也不解释注册权威机构应该遵循的规则。它们在 GB/T 17969.1 中规定。

D.1 客体标识符成分值的根赋值

D.1.1 从根节点开始规定了 3 条弧。值和标识符的赋值，以及随后成分值赋值的权威机构如下：

值	标识符	随后赋值权威机构
0	itu-t	ITU-T(见 D.2)
1	iso	ISO(见 D.3)
2	joint-iso-itu-t	见 D.4

D.1.2 按上面赋值的标识符 itu-t、iso 和 joint-iso-itu-t，每个可用作为“NameForm”(见 31.3)。

D.1.3 标识符 ccitt 和 joint-iso-ccitt 分别与 itu-t 和 joint-iso-itu-t 同义，因此可以在客体标识符值中出现。

D.2 客体标识符成分值的 ITU-T 赋值

D.2.1 从 itu-t 标识的节点开始规定了 5 条弧。值和标识符的赋值是：

值	标识符	子序列赋值权威机构
0	recommendation	见 D.2.2
1	question	见 D.2.3
2	administration	见 D.2.4
3	network-operator	见 D.2.5
4	identified-organization	见 D.2.6

这些标识符可用作为“NameForm”(见 31.3)。

D.2.2 在 recommendation 下的弧具有带赋予的标识符 a 至 z 的值 1 至 26。它们下的弧具有由字母标识的序列中 ITU-T(和 CCITT)标准的数字。其下的弧根据需要由 ITU-T(和 CCITT)标准确定。标识符 a 至 z 可用作为“NameForm”。

D.2.3 在 question 下的弧有研究期间批准的、对应 ITU-T 研究组的值。该值由公式计算：

study group number(研究组号) +(period * 32)

其中“period”对于 1984—1988 其值为 0，对于 1988—1992 其值为 1，等等，它与十进制数 32 相乘。

在每个研究组下的弧具有对应赋予那个研究组的问题的值。它下面的弧根据需要由指派研究问题的组确定(例如，工作方或特殊报告起草人组)。

D.2.4 在 administration 下的弧有 X.121 DCCs 的值。它下面的弧根据需要由 X.121 DCC 标识的国家的管理部门确定。

D.2.5 在 network-operator 下的弧具有 X.121 DNICs 的值。它下面的弧根据需要由 X.121 DNIC 标识的 ROA 或管理部门确定。

D.2.6 在 identified-organization 下的弧是 ITU 通信标准局(TSB)赋予的值。它下面的弧根据需要由 ITU 值标识的组织确定。

注：可能发现这条弧有用的组织包括：

——不运营公共数据网络的认可的运营机构；

——科学和工业组织；
——地区标准组织；和
——多国组织。

D.3 客体标识符成分值的 ISO 赋值

D.3.1 从“iso”标识的节点开始规定了三条弧。值和标识符的赋值是：

值	标识符	子序列赋值权威机构
0	standard	见 D.3.2
2	member-body	见 D.3.3
3	identified-organization	见 D.3.4

这些标识符可用作为“NameForm”(见 31.3)。

注：已经撤销使用弧 registration-authority(1)。

D.3.2 在每个 standard 下的弧应该有国际标准号的值。在国际标准是多部分时，部分号后面应该有附加弧，除非特殊情况下它不包括在国际标准中。进一步的弧应该具有如那个国际标准中定义的值。

D.3.3 紧接 member-body 下的弧应该有三位数字的国家代码的值，正如 GB/T 2659 中规定的，标识那个国家的 ISO 国家委员会。不允许在这些标识符上有客体标识符成分的“NameForm”。

D.3.4 紧接 identified-organization 下面的弧应该具有注册权威机构为 SJ/Z 9090—1987 分配的、标识那个权威机构像分配客体标识符成分一样特别注册的发布组织的国际代码指定者(ICD)值。紧接 ICD 下面的弧应该有符合 SJ/Z 9090—1987 的发布组织分配的“organization code”值。

D.4 客体标识符成分值的联合赋值

D.4.1 根据 GB/T 17969.3，在 joint-iso-itu-t 下的弧有时由 ISO/IEC 和 ITU-T 建立登记权威机构赋予和同意的值来标识联合 ISO/IEC|ITU-T 标准机构的领域。

附 录 E
（资料性附录）
举例和提示

本附录包含描述（假定）数据结构中采用 ASN.1 的示例。它也包含有使用 ASN.1 各种特性的提示或指南。除非另有说明，否则假设是 AUTOMATIC TAGS 环境。

E.1 人事记录的示例

用一个简单、假设的人事记录来说明 ASN.1 的用法。

E.1.1 人事记录的非形式描述

人事记录的结构和某个特定个人的值如下所示：

Name:	John P Smith
Title:	Director
Employee Number:	51
Date of Hire:	1971 年 9 月 17 日
Name of Spouse:	Mary T Smith
Number of Children:	2
Child Information	
Name:	Ralph T Smith
Date of Birth	1957 年 11 月 11 日
Child Information	
Name:	Susan B Jones
Date of Birth	1959 年 7 月 17 日

E.1.2 记录结构的 ASN.1 描述

下面使用数据类型的标准记法形式地描述每个人事记录的结构。

```
PersonnelRecord ::=[APPLICATION 0] SET
{       name            Name,
        title           VisibleString,
        number          EmployeeNumber
        dateOfHire      Date,
        nameOfSpouse    Name,
        children        SEQUENCE OF ChildInformation DEFAULT {}
}
ChildInformation::= SET
{name   Name,
        dateOfBirth     Date
}
Name::=[APPLICATION 1] SEQUENCE
{       givenName       VisibleString,
        initial         VisibleString,
        familyName      VisibleString
}
```

```
EmployeeNumber::=[APPLICATION 2] INTEGER
Date::=[APPLICATION 3] VisibleString    -- YYYY MMDD
```

本例说明了 ASN.1 语法分析的一个方面。语法结构 DEFAULT 只能适用于 SEQUENCE 或 SET 成分，它不能适用于 SEQUENCE OF 的元素。因此，在 PersonnelRecord 中的 DEFAULT { }适用于 children，而不适用于 ChildInformation。

E.1.3 记录值的 ASN.1 描述

下面使用数据值的标准记法形式地描述 John Smith 的个人记录值。

```
{  name   {givenName "John",initial " P", familyName "Smith"},
   title                  " Director",
   number                 51,
   dateOfHire             " 19710917"
   nameOfSpouse           {givenName " Mary", initial " T" familyName " Smith"},
   children
   {  {name   { givenName "Ralph", initial "T" , familyName " Smith"},
         dateOfBirth " 19571111"},
      {name{ givenName " Susan", initial " B" familyName " Jones"},
         dateOfBirth " 19590717"}
   }
}
```

或者以 XML 值记法：

```
person::=
    <PersonnelRecord>
        <name>
        <givenName>John</givenName>
            <initial>P</initial>
            <familyName>Smith</familyName>
            </name>
        <title>Director</title>
        <number>51</number>
        <dateOfHire>19710917</dateOfHire>
        <nameOfSpouse>
            <givenName>Mary</givenName>
            <initial>T</initial>
            <familyName>Smith</familyName>
        </nameOfSpouse>
        <children>
            <ChildInformation>
                <name>
                    <givenName>Ralph</givenName>
                    <initial>T</initial>
                    <familyName>Smith</familyName>
                </name>
                <dateOfBirth>19571111</dateOfBirth>
```

```
        </ChildInformation>
        <ChildInformation>
            < name >
                < givenName >Susan</ givenName >
                < initial >B</ initial >
                < familyName >Jones</ familyName >
            </ name >
            < dateOfBirth >19590717</ dateOfBirth >
        </ChildInformation>
    </children>
</PersonnelRecord>
```

E.2 使用记法指南

本部分定义的数据类型及形式记法是灵活的，可广泛地用于各种协议的描述。然而，这种灵活性有时会带来混淆，尤其是在第一次用这个记法时。本附录通过给出使用记法的指南和示例，意于将混淆减到最少。对于每个固有数据类型，提供了一个或多个使用指南。这里不考虑字符串类型（如：VisibleString）和第 41 至 43 章定义的类型。

E.2.1 布尔

E.2.1.1 用布尔类型为逻辑（即：双态）变量值建模，例如，是否问题的答案。

例：

```
Employed ::= BOOLEAN
```

E.2.1.2 当赋给布尔类型引用名称时，选择描述 true 状态的那个。

例：

```
Married ::= BOOLEAN
```

而不是

```
MaritalStatus ::= BOOLEAN
```

E.2.2 整数

E.2.2.1 用整数类型为坐标或整数变量的值（为满足通用性而不限界）建模。

例：

```
CheckingAccountBalance ::= INTEGER          -- 精确到分；负为超支。
balance CheckingAccountBalance ::= 0
```

或者用 XML 值记法：

```
balance::=<CheckingAccountBalance >0</CheckingAccountBalance >
```

E.2.2.2 定义整数类型有名数字的最小和最大允许值。

例：

```
DayOfTheMonth ::= INTEGER {first(1), last(31)}
today DayOfTheMonth ::= first
unknown DayOfTheMonth ::= 0
```

或者用 XML 值记法：

```
today::=< DayOfTheMonth ><first/></ DayOfTheMonth >
unknown::=< DayOfTheMonth ></ DayOfTheMonth >
```

注意，选择有名数字 first 和 last 是因为他们对于读者语法的重要性，并且不排除有其他可能小于 1、大于 31 或在 1 与 31 之间的 DayOfTheMonth 的可能性。

要限制 DayOfTheMonth 的值刚好是 first 和 last，它应该写作：

DayOfTheMonth ::= INTEGER {first(1), last(31) } (first | last)

以及要限制 DayOfTheMonth 的值在 1 与 31(含)之间的所有值，它应该写作：

DayOfTheMonth ::= INTEGER {first(1), last(31) } (first .. last)

dayOfTheMonth DayOfTheMonth ::= 4

或者用 XML 值记法：

dayOfTheMonth ::= < DayOfTheMonth >4</ DayOfTheMonth >

E.2.3 枚举

E.2.3.1 用枚举类型为有不少于三个状态的变量值建模。如果它们仅有的约束是不同的，则从零开始赋值。

例：

DayOfTheWeek ::= ENUMERATED {sunday(0), monday(1), tuesday(2), wednesday(3), thursday(4), friday(5), saturday(6),}

firstDay DayOfTheWeek ::= sunday

或者用 XML 值记法：

firstDay ::=< DayOfTheWeek ><sunday /></ DayOfTheWeek >

注意，由于他们对读者语义的重要性，在选择枚举 sunday、monday 等等时，DayOfTheWeek 限制为假设这些值之一且没有其他值。另外，只有名称 sunday、monday 等等能赋给值；不允许有相当的整数值。

E.2.3.2 用可扩展的枚举类型为变量值建模，变量值在目前只有两个状态、但是在协议的今后版本中可能有附加状态。

例：

MaritalStatus ::= ENUMERATED {single,married} -- MaritalStatus 第一版

在下列预料中

MaritalStatus ::= ENUMERATED { single,married,...,widowed} -- MaritalStatus 第二版

今后还有：

MaritalStatus ::= ENUMERATED { single,married,...,widowed,divorced} -- MaritalStatus 第三版

E.2.4 实数

E.2.4.1 用实数类型为大约数目建模。

例"

AngleInRadian ::= REAL

pi REAL ::= {mantissa 3141592653589793238462643383279,base10,exponent-30}

或者用实数的替换值记法：

pi REAL ::= 3.14159265358979323846264338327

或者用 XML 值记法：

pi::=

<REAL>

3.14159265358979323846264338327

</REAL>

E.2.4.2 应用设计者可能希望保证与实数值全部互工作，尽管在浮点硬件中不同，而且在实现决定中为应用使用(例如)单个或双倍长度浮点。它能通过下面实现：

App-X-Real::=REAL (WITH COMPONENTS{

```
                    mantissa (-16777215..16777215),
                    base  (2),
                    exponent (-125..128))
```

/*
发送者不应该传输这些范围之外的值，而且合格的接受者应能够接受和处理这些范围内的值。
*/

girth App-X-Real::={mantissa 16,base 2,exponent 1}

或者用 XML 值记法：

```
girth ::=
    <App-X-Real>
    32
    </App-X-Real>
```

E.2.5 位串

E.2.5.1 用位串类型为没有规定格式和长度，或者在其他地方规定，而且其位的长度不必是 8 的整数倍的二进制数据建模。

例：

```
G3FacsimilePage ::= BIT STRING
-- 符合 ITU-T T.4 建议的位序列。
image G3FacsimilePage::= '100110100100001110110'B
trailer BIT STRING::= '0123456789ABCDEF'H
body1 G3FacsimilePage::= '1101'B
body2 G3FacsimilePage::= '1101000'B
```

或者用 XML 值记法：

```
image::= <G3Facsimile>100110100100001110110</G3 Facsimile>
trailer ::=
        <BIT_ STRING>
            0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011
            1100 1101 1110 1111
        </BIT_ STRING>
body1::=<G3Facsimile >1101</G3 Facsimile >
body2::=<G3Facsimile >1101000</G3Facsimile >
```

注意，因为结尾 0 位很重要(由于 G3FacsimilePage 的定义中没有“NamedBitList”)，因此，“body1”和“body2”是不同的抽象值。

E.2.5.2 用有大小约束的位串类型为固定的依大小排列的位字段的值建模。

例：

```
BitField::=BIT STRING (SIZE (12))
    map1 BitField::= '100110100100'B
    map2 BitField::= '9A4'H
    map3 BitField::= '1001101001'B  -- 非法的——与大小约束冲突。
```

或者用 XML 值记法：

```
    map1::=<BitField>100110100100</Bit Field>
```

注意，“map1”和“map2”是相同的抽象值，因为“map2”的四个结尾位没有意义。

E.2.5.3 用位串类型为 bit map 值建模，bit map 是指明是否为每个对应的客体有序集合保持特定条

件的逻辑变量的有序集合。

```
DaysOfTheWeek ::= BIT STRING {
        sunday(0), monday(1), tuesday(2), wednesday(3),
        thursday(4), friday(5), saturday(6)} (SIZE (0..7))
sunnyDaysLastWeek1 DaysOfTheWeek ::= {sunday,monday,wednesday}
  sunnyDaysLastWeek2 DaysOfTheWeek::= '1101'B
  sunnyDaysLastWeek3 DaysOfTheWeek::= '1101000'B
sunnyDaysLastWeek4 DaysOfTheWeek::= '11010000'B     -- 非法的
```

或者用 XML 值记法：

```
sunnyDaysLastWeek1 ::=
  < DaysOfTheWeek >
     <sunday/><monday/><wednesday/>
  < DaysOfTheWeek >
sunnyDaysLastWeek2::= < DaysOfTheWeek >1101</ DaysOfTheWeek >
sunnyDaysLastWeek3::= < DaysOfTheWeek >1101000</ DaysOfTheWeek >
```

注意，如果位串值的长度小于 7 位，那么丢失的位指明那些天为阴天，也就是上面的最初三个值有相同的抽象值。

E.2.5.4 用位串类型为 *bit map* 值建模，是指明是否为每个对应的客体有序集合保持特定条件的逻辑变量的固定大小的有序集合。

```
DaysOfTheWeek ::= BIT STRING {
    sunday(0), monday(1), tuesday(2), wednesday(3),
    thursday(4), friday(5), saturday(6)} (SIZE (7))
sunnyDaysLastWeek1 DaysOfTheWeek ::= { sunday,monday,wednesday }
sunnyDaysLastWeek2 DaysOfTheWeek::= '1101'B  -- 非法的——与大小约束冲突
sunnyDaysLastWeek3 DaysOfTheWeek::= '1101000'B
sunnyDaysLastWeek4 DaysOfTheWeek::= '11010000'B  -- 非法的——与大小约束冲突
```

注意，第一个和第三个值有相同的抽象值。

E.2.5.5 用带已命名位的位串类型为相关逻辑变量集合的值建模。

例：

```
PersonalStatus::=BIT STRING
      {married(0),employed(1),veteran(2),collegeGraduate(3)}
billClinton PersonalStatus ::={ married,employed,collegeGraduate}
hillaryClinton PersonalStatus::= '110100'B
```

或者用 XML 值记法：

```
billClinton::=
   < PersonalStatus >
      <married/>
      <employed/>
      <collegeGraduate/>
   </PersonalStatus >
hillaryClinton::= < PersonalStatus > 110100</ PersonalStatus >
```

注意，“billClinton”和“hillaryClinton”有相同的抽象值。

E.2.6 八位位组串

E.2.6.1 用八位位组串类型为没有规定格式和长度，或者在其他地方规定，而且其位的长度是 8 的整数倍的二进制数据建模。

例：

```
G4FacsimileImage::=OCTET STRING
-- 符合 ITU-T T.5 和 ITU-T T.6 建议的八位位组序列。
image G4FacsimilePage::= '3FE2EBAD471005'B
```

或者用 XML 值记法：

```
image::= <G4FacsimileImage>3FE2EBAD471005</G4FacsimileImage>
```

E.2.6.2 在有合适的受限制字符串类型时，优先用它而不用八位位组串类型。

例：，

```
Surname::= PrintableString
president Surname::= "Clinton"
```

或者用 XML 值记法：

```
president::=<Surname>Clinton</Surname>
```

E.2.7 UniversalString、BMPString 和 UTF8String

用 BMPString 类型或 UTF8String 类型来为只由取自 GB/T 13000.1 基本多文种平面(BMP)的字符组成的任何信息串建模，而 UniversalString 或 UTF8String 则为由不属于 BMP 的 GB/T 13000.1 字符构成的任何串建模。

E.2.7.1 用 Level1 或 Level2 指明实现级数对使用组合字符的限制。

例：

```
RussianName::=Cyrillic(Level1)
    -- RussianName 用无组合字符。
SaudiName::= BasicArabic(SIZE (1..100) ^ Level2)
    -- SaudiName 用组合字符的子集。
```

Σ 的表达式：

```
greekCapitalLetterSigma BMPString::={0, 0, 3, 163}
```

或用 XML 值记法：

```
greekCapitalLetterSigma::= <BMPString>&#x03a3;</BMPString>
```

串"f→∝"的表达式：

```
rightwardsArrow UTF8String::={0, 0, 33, 146}
infinity UTF8String::={0, 0, 34, 30}
property UTF8String::={ "f",rightwardsArrow, "  ",infinity}
```

或用 XML 值记法：

```
property ::=<UTF8String>f &#x2192; &#x221E;</UTF8String>
```

E.2.7.2 通过使用"UnionMark"(见第 46 章)，集合可扩大成选择的子集(即：包括 BASIC LATIN 集合中的所有字符)。

例：

```
KatakanaAndBasicLatin::= UniversalString(FROM (Katakana | BasicLatin))
```

E.2.8 CHARACTER STRING

用无限制字符串类型来为不能用受限制字符串类型之一建模的任何信息串建模。确保规定了字符汇和八位位组代码。

例：

```
PackedBCDString::= CHARACTER STRING ( WITH COMPONENTS{
                                    identification (WITH COMPONENTS {
                                    fixed PRESENT})
  /*抽象和传送语法应该是下面定义的 PackedBCDString-AbstractSyntaxId
    和 PackedBCDString-TransferSyntaxId。
  */
                  })
  /* 字母为数字 0 至 9 的字符抽象语法(字符集)的客体标识符值。
  */
  PackedBCDString-AbstractSyntaxId OBJECT IDENTIFIER::=
      {joint-iso-itu-t asn1(1)  examples(123)  PackedBCD(2)  charSet(0)  }
  /* 每八位位组紧缩两位数字,每位数字编码为 0000 至 1001,1111₂ 用作填充的
     字符传送语法的客体标识符值。
  */
  PackedBCDString-TransferSyntaxId OBJECT IDENTIFIER::=
      {joint-iso-itu-t asn1(1)  examples(123)  PackedBCD(2)  characterTransferSyntax(1)}
/* PackedBCDString 的编码将只包含字符的已定义编码,有任意必需长的字段,
   以及在 BER 时,携带标记的字段。当"fixed"已规定时,不携带客体标识符值。
*/
```

或者用 XML 值记法:

```
PackedBCDString- AbstractSyntaxId::=
  <OBJECT_IDENTIFIER>
      joint-iso-itu-t. asn1(1). examples(123). packedBCD(2). charSet(0)
  </OBJECT_ IDENTIFIER>
packedBCDString-TransferSyntaxId::=
<OBJECT_IDENTIFIER>
joint-iso-itu-t. asn1(1). examples(123). packedBCD(2). CharacterTransferSyntax(1)
</OBJECT_IDENTIFIER>
```

或者

```
packedBCDString- AbstractSyntaxId::=
  <OBJECT_IDENTIFIER>2. 1. 123. 2. 0</OBJECT_ IDENTIFIER>
packedBCDString-TransferSyntaxId::=
  <OBJECT_IDENTIFIER>2. 1. 123. 2. 1</OBJECT_IDENTIFIER>
```

注:编码规则不必将类型 CHARACTER STRING 的值编码成总是包含客体标识符值的形式,尽管它们保证编码中保留抽象值。

E.2.9 空

用空类型来指明序列成分的有效缺失。

例:

```
PatientIdentifier ::=SEQUENCE{
  name   VisibleString,
  roomNumber   CHOICE{
      room      INTEGER,
      outPatient      NULL   -- 如果一个门诊病人 --
```

```
    }
}
lastPatient PatientIdentifier::={
    name "Jane Doe"
    roomNumber outPatient:  NULL
}
```

或者用 XML 值记法：

```
lastPatient::=
    < PatientIdentifier >
        <name>Jane Doe</name>
        <roomNumber><outPatient/></roomNumber>
    </ PatientIdentifier >
```

E.2.10 序列和单一序列

E.2.10.1 用单一序列类型来为类型相同、数字大或不可预知、顺序有意义的变量集合建模。

例：

```
NamesOfMemberNations::= SEQUENCE OF VisibleString
-- 按字母表顺序
firstTwo NamesOfMemberNations::={ " Australia"," Austria"}
```

或用可选标识符：

```
NamesOfMemberNations2::=SEQUENCE OF memberNation VisibleString
-- 按字母表顺序
firstTwo2 NamesOfMemberNations2::=
    { memberNation " Australia",memberNation "Austria"}
```

采用 XML 值记法，上面的两个值如下：

```
firstTwo ::=
    <NamesOfMemberNations >
        < VisibleString > Australia </ VisibleString >
        < VisibleString > Austria </ VisibleString >
    </ NamesOfMemberNations >
firstTwo2 ::=
    < NamesOfMemberNations2>
        < memberNation > Australia </ memberNation >
        < memberNation > Austria </ memberNation >
    </ NamesOfMemberNations 2>
```

E.2.10.2 用序列类型来为类型相同、数字已知和适中、及顺序有意义的变量集合建模，只要集合的构成不可能从协议的一个版本改变到下一个。

例：

```
NamesOfOfficers::=SEQUENCE {
    president           VisibleString,
    vicePresident       VisibleString,
    secretary           VisibleString}
acmeCorp NamesOfOfficers::= {
    president           " Jane Doe",
```

```
        vicePresident           " John Doe",
        secretary               " Joe Doe"}
```

或者用 XML 值记法：

```
    acmeCorp::=
        < NamesOfOfficers >
            < president >Jane Doe</ president >
            < vicePresident >John Doe</ vicePresident >
            < secretary >Joe Doe</ secretary >
        </ NamesOfOfficers >
```

E.2.10.3 用不可扩展的序列类型来为类型不同、数字已知和适中、顺序有意义的变量集合建模，只要集合的构成不可能从协议的一个版本改变到下一个。

例：

```
    Credentials::=SEQUENCE {
        userName            VisibleString,
        password            VisibleString,
        accountNumber   INTEGER}
```

E.2.10.4 用可扩展序列类型来为顺序有意义、数字当前已知和适中但预计会增加的变量集合建模。

例：

```
    Record::=SEQUENCE {   -- 包含"Record"的协议的第一版
        userName            VisibleString,
        password            VisibleString,
        accountNumber   INTEGER,
        ...,
        ...
    }
```

在预料中：

```
    Record::=SEQUENCE {   -- 包含"Record"的协议的第二版
        userName            VisibleString,
        password            VisibleString,
        accountNumber   INTEGER
        ...,
        [[2: --协议第二版中增加的扩展附加
            lastLoggedIn        GeneralizedTime OPTIONAL,
            minutesLastLoggedIn INTEGER
        ]],
        ...
    }
```

且后来还有(没有附加到纪录的协议的第三版)：

```
    Record::=SEQUENCE {   -- 包含"Record"的协议的第三版
        userName            VisibleString,
        password            VisibleString,
        accountNumber   INTEGER,
        ...,
```

```
[[2: --协议第二版中增加的扩展附加
  lastLoggedIn          GeneralizedTime OPTIONAL,
  minutesLastLoggedIn   INTEGER
]],
[[4:  -- 协议第三版中增加的扩展附加
  certificate     Certificate,
  thumb           ThumbPrint OPTIONAL
]],
...
}
```

E.2.11 集合和单一集合

E.2.11.1 用集合类型来为数字已知和适中且顺序无意义的变量集合建模。如果自动置标记无效,用如下所示的特殊上下文置标记来标识每个变量。(有自动置标记时,不必加标记。)

例:

```
UserName::=SET{
        personalName            [0] VisibleString,
        organizationName        [1] VisibleString,
        countryName             [2] VisibleString}
user UserName::={
        countryName             " Nigeria",
        personalName            " Jonas Maruba",
        organizationName        " Meteorology, Ltd. "}
```

或者用 XML 值记法:

```
user  ::=
        <UserName>
        < countryName > Nigeria </ countryName >
        < personalName >Jonas Maruba</ personalName >
        < organizationName >Meteorology, Ltd. < organizationName >
        </ UserName >
```

E.2.11.2 用有 OPTIONAL 的集合类型来为数字已知和合理的小且顺序无意义的变量的另一个集的(正确或不正确)子集的变量集合建模。如果自动置标记无效,用如下所示的特殊上下文置标记来标识每个变量。(有自动置标记时,不必加标记。)

例:

```
UserName::=SET{
        personalName            [0] VisibleString,
        organizationName        [1] VisibleString OPTIONAL
        -- 默认为当地组织的那个--
        countryName             [2] VisibleString OPTIONAL
        -- 默认为当地国家的那个 -- }
```

E.2.11.3 用可扩展集合类型来为构成可能从一个协议版本改变到下一个的变量集合建模。下面假设模块定义中规定了 AUTOMATIC TAGS。

例:

```
UserName::=SET{
```

```
        personalName            VisibleString,  -- "UserName"的第一版
        organizationName        VisibleString OPTIONAL,
        countryName             VisibleString OPTIONAL,
        ...,
        ...
    }
    user userName ::= {personalName " Jonas Maruba"}
```

或者用 XML 值记法：

```
    user  ::=
        < UserName >
            < personalName >Jonas Maruba</ personalName >
        </ UserName >
```

在预料中：

```
    UserName::=SET{     -- "UserName"第二版
        personalName            VisibleString,
        organizationName        VisibleString OPTIONAL,
        countryName             VisibleString OPTIONAL,
        ...,
        [[2:            -- 协议第二版中增加的扩展附加
            internetEmailAddress  VisibleString,
            faxNumber             VisibleString OPTIONAL
        ]],
        ...
    }
    user  UserName::={
        personalName            " Jonas Maruba",
        internetEmailAddress    " jonas@meteor. ngo. com"
    }
```

或者用 XML 值记法：

```
    user::=
    <UserName>
        < personalName >Jonas Maruba</ personalName >
        < internetEmailAddress >jonas@meteor. ngo. com</ internetEmailAddress >
    <UserName>
```

且后来还有(没有附加到用户名的协议的第三和四版)：

```
    UserName::=SET{     -- 包含"UserName"的协议第五版
        personalName            VisibleString,
        organizationName        VisibleString OPTIONAL,
        countryName             VisibleString OPTIONAL,
        ...,
        [[2:            -- 第二版中增加的扩展附加
            internetEmailAddress  VisibleString,
            faxNumber             VisibleString OPTIONAL
```

```
        ]],
        [[5:        -- 第五版中增加的扩展附加
        phoneNumber             VisibleString OPTIONAL
        ]],
        ...
    }
    user    UserName::={
        personalName            " Jonas Maruba",
        internetEmailAddress    " jonas@meteor. ngo. com"
    }
```

或者用 XML 值记法：

```
user: :=
    < UserName >
        < personalName >Jonas Maruba</ personalName >
        < internetEmailAddress >jonas@meteor. ngo. com</ internetEmailAddress >
        </ UserName >
```

E. 2. 11. 4　用单一集合类型来为类型相同和顺序无意义的变量集合建模。

例：

```
Keywords::= SET OF VisibleString    --按任意顺序
someASN1Keywords Keywords::={ " INTEGER", " BOOLEAN", " REAL"}
```

或者采用可选标识符：

```
Keywords2::=SET OF keyword VisibleString    -- 按任意顺序
someASN1 Keywords2 Keywords2::={keyword " INTEGER", keyword " BOOLEAN",
        keyword "REAL"}
```

采用 XML 值记法，上面的两个值如下：

```
someASN1Keywords::=
    <Keywords>
        <VisibleString>INTEGER</VisibleString>
        <VisibleString>BOOLEAN</VisibleString>
        <VisibleString>REAL</VisibleString>
    </Keywords>
someASN1Keywords2::=
    <Keywords2>
        <keyword>INTEGER</keyword>
        <keyword>BOOLEAN</keyword>
        <keyword>REAL</keyword>
    </ Keywords2>
```

E. 2. 12　已标记

介绍 AUTOMATIC TAGS 结构之前，ASN. 1 规范常常包含标记。下面各条描述典型应用置标记的方法。随着 AUTOMATIC TAGS 的介绍，新 ASN. 1 规范不必采用标记记法，尽管那些修改旧记法可能使其本身与标记有关。鼓励 ASN. 1 记法的新用户使用 AUTOMATIC TAGS，因为，它使记法更可读。

E. 2. 12. 1　通用类别标记仅用于本部分内。提供记法[UNIVERSAL 30](示例)只是为了“Useful-

Types"(见 41.1)的定义精确。它不应该用在其他地方。

E.2.12.2 使用标记经常碰到的形式是在整个规范中分配应用层类别标记精确地一次,用它标识在规范内发现宽度、分散、用途的类型。也常常使用应用层类别标记(只有一次)来标记应用的最外CHOICE的类型,如果通过应用类别标标识个人信息。下面是前面情形的举例用法:

例:

```
FileName::= [APPLICATION 8] SEQUENCE {
        directoryName                VisibleString,
        directoryRelativeFileName    VisibleString}
```

E.2.12.3 特定上下文置标记常常以代数方式应用于 SET、SEQUENCE 或者 CHOICE 的所有成分中。然而,要注意,AUTOMATIC TAGS 设施为你轻松地完成这些。

例:

```
CustomerRecord ::= SET {
    name              [0] VisibleString,
    mailingAddress    [1] VisibleString,
    accountNumber     [2] INTEGER,
    balanceDue        [3] INTEGER -- 以分计 -}
CustomerAttribute::=CHOICE{
    name              [0] VisibleString,
    mailingAddress    [1] VisibleString,
    accountNumber     [2] INTEGER,
    balanceDue        [3] INTEGER -- 以分计 -}
```

E.2.12.4 专用类别置标记通常不应该用在国际标准化规范(尽管没有禁止这样做)。企业生产的应用通常使用应用层和特定上下文标记类别。然而,特定企业的规范试图扩大国际标准化规范可能是常见的情形,并且在这种情况下,采用专用类别标记可能会在部分地保护特定企业规范变化成国际标准化规范中获得一些利益。

例:

```
AcmeBadgeNumber::= [PRIVATE 2] INTEGER
badgeNumber AcmeBadgeNumber::=2345
```

或者用 XML 值记法:

```
badgeNumber::= < AcmeBadgeNumber>2345</ AcmeBadgeNumber>
```

E.2.12.5 文本使用带各个标记的 IMPLICIT 通常只能在较早的规范中找得到。使用隐式置标记时,BER 产生比使用显式时更不简洁的表达式。PER 在两种情况下产生同样简洁的编码。用 BER 和显式置标记,已编码的数据中的重要类型(INTEGER、REAL、BOOLEAN,等等)具有更好的可视性。在这样做合法的示例中,这些指南采用隐式置标记。根据编码规则,这样可能产生在某些应用程序中非常渴望的简洁表达式。在其他应用中,例如,简洁可能没有进行强类型检查的能力重要。在后面的情况下,可以采用显式置标记。

例:

```
CustomerRecord::= SET {
        name              [0] IMPLICIT VisibleString,
        mailingAddress    [1] IMPLICIT VisibleString,
        accountNumber     [2] IMPLICIT INTEGER,
        balanceDue        [3] IMPLICIT INTEGER -- 以分计 -}
CustomerAttribute::=CHOICE{
```

```
        name            [0] IMPLICIT VisibleString,
        mailingAddress  [1] IMPLICIT VisibleString,
        accountNumber   [2] IMPLICIT INTEGER,
        balanceDue      [3] IMPLICIT INTEGER    -- 以分计 --}
}
```

E.2.12.6 使用引用本部分的新 ASN.1 规范中的标记的指南十分简单:不要用标记。将 AUTOMATIC TAGS 放在模块头中,然后,不必管标记。如果你需要在后面的版本中增加新的成分给 SET、SEQUENCE 或者 CHOICE,将它们加在末尾。

E.2.13 选择

E.2.13.1 用 CHOICE 来为从总数已知且适中的变量集中选择的变量建模。

例:

```
FileIdentifier::= CHOICE {
        relativeName VisibleString,
    -- 文件名称(例如,"MarchProgressReport")
        absoluteName VisibleString,
    -- 文件名称并包含目录(例如,"<Williams>MarchProgressReport")
        serialNumber INTEGER
    -- 指派的系统的文件的标识符 -- }
file FileIdentifier::= serialNumber:106448503
```

或者用 XML 值记法:

```
fileIdentifier::=
< FileIdentifier >
    < serialNumber >106448503</ serialNumber >
</ FileIdentifier >
```

E.2.13.2 用可扩展 CHOICE 来为构成可能从协议的一个版本改变为另一个的变量集合中选择的变量建模。

例:

```
FileIdentifier ::= CHOICE {   -- FileIdentifier 的第一版
        relativeName      VisibleString,
        absoluteName      VisibleString,
        ...,...
}
fileId1 FileIdentifier::= relativeName: "MarchProgressReport.doc"
```

或者用 XML 值记法:

```
fileId1::=
    < FileIdentifier >
        < relativeName > MarchProgressReport.doc</ relativeName >
    </ FileIdentifier >
```

在预料中:

```
FileIdentifier::= CHOICE{     -- FileIdentifier 的第二版
        relativeName      VisibleString,
        absoluteName      VisibleString,
        ...,
```

```
        serialNumber    INTEGER,  -- 第二版中增加的扩展附加
        ...
    }
    fileId1 FileIdentifier::= relativeName: " MarchProgressReport. doc"
    fileId2 FileIdentifier::= serialNumber:214
```

或者用 XML 值记法：

```
    fileId1 ::=
        < FileIdentifier >
            < relativeName > MarchProgressReport. doc</ relativeName >
        </ FileIdentifier >
    fileId2::=
        < FileIdentifier >
            < serialNumber >214</ serialNumber >
        </ FileIdentifier >
```

且后来还有：

```
    FileIdentifier::= CHOICE {  -- FileIdentifier 的第三版
        relativeName        VisibleString,
        absoluteName        VisibleString,
        ...,
        serialNumber        INTEGER,    -- 第二版中增加的扩展附加
        [[                              -- 第三版中增加的扩展附加
          vendorSpecific    VendorExt,
          unidentified      NULL
        ]],
        ...
    }
    fileId1 FileIdentifier ::= relativeName: " MarchProgressReport. doc"
    fileId2 FileIdentifier ::= serialNumber:214
    fileId3 FileIdentifier ::= unidentified:NULL
```

或者用 XML 值记法：

```
    fileId1 ::=
        < FileIdentifier >
            < relativeName > MarchProgressReport. doc</ relativeName >
        < FileIdentifier >
    fileId2 ::=
        < FileIdentifier >
            < serialNumber >214</ serialNumber >
        < FileIdentifier >
    fileId3 ::=
        < FileIdentifier >
            <unidentified/>
        </ FileIdentifier >
```

E.2.13.3　预计在将来允许可能不止一种类型时，使用仅有一个类型的可扩展 CHOICE。

例：

```
Greeting::=CHOICE{    -- " Greeting " 的第一版
        postCard      VisibleString,
        ...,
        ...
}
```

在预料中：

```
Greeting::=CHOICE{    -- "Greeting"的第二版
        Postcard      VisibleString,
        ...,
        [[2:      -- 第二版中增加的扩展附加
            audio     Audio,
            video     Video
        ]],
        ...
}
```

E.2.13.4 当选择值嵌套在另一个选择值内时，要求用多个冒号。

例：

```
Greeting::= [APPLICATION 12] CHOICE{
            postCard      VisibleString,
            recording     Voice}
Voice::=CHOICE {
            english       OCTET STRING,
            Swahili       OCTET STRING}
myGreeting Greeting ::= recording : English : '019838547E0'H
```

或者用 XML 值记法：

```
myGreeting::=``
    < Greeting >
        <recording><english>019838547E0</english></recording>
    </ Greeting >
```

E.2.14 精选类型

E.2.14.1 用精选类型为其类型是前面定义的 CHOICE 某些特定替换项的变量建模。

E.2.14.2 考虑定义：

```
FileAttribute ::=CHOICE{
            date-last-used   INTEGER,
            file-name        VisibleString
```

然后，可能是下面的定义：

```
AttributeList ::=SEQUENCE{
            first-attribute       date-last-used < FileAttribute,
            second- attribute     file-name < FileAttribute }
```

可能的值记法：

```
listOfAttributes AttributeList::={
            first - attribute       27,
```

```
                second - attribute      " PROGRAM " }
```

或者用 XML 值记法：

```
listOfAttributes::=
    < AttributeList >
        < first - attribute >27</ first - attribute >
        < second - attribute >PROGRAM</ second - attribute >
    </ AttributeList >
```

E.2.15 客体类别字段类型

E.2.15.1 用客体类别字段类型来标识用信息客体类别(见 GB/T 16262.2)定义的类型。例如，信息客体类别 ATTRIBUTE 的范围可用于定义类型、Attribute。

例：

```
ATTRIBUTE ::= CLASS{
        &AttributeType,
        &attributeId        OBJECT IDENTIFIER UNIQUE
            }
Attribute::=SEQUENCE {
        attributeID         ATTRIBUTE.& attributeId,  -- 这样通常是约束的
        attributeValue      ATTRIBUTE.& AttributeType  -- 这样通常是约束的
            }
```

ATTRIBUTE.& attributeId 和 ATTRIBUTE.& AttributeType 都是客体类别字段类型，因为它们是引用信息客体类别(ATTRIBUTE)定义的类型。因为类型 ATTRIBUTE.& attributeId 在 ATTRIBUTE 中显式地定义为 OBJECT IDENTIFIER，因此它是固定的。然而，类型 ATTRIBUTE.&AttributeType 能携带用 ASN.1 定义的任何类型的值，因为其类型在信息客体类别 ATTRIBUTE 的定义中没有固定。具有能携带任何类型值的这一特性的记法称为“开放类型记法”，即：ATTRIBUTE.&AttributeType 是开放类型。

E.2.16 嵌入式 pdv

E.2.16.1 用嵌入式 pdv 类型来为其类型没有规定、或在不限制用来规定类型的记法的其他地方规定的变量建模。

例：

```
FileContent::=EMBEDDED PDV
DocumentList::=SEQUENCE OF document EMBEDDED PDV
```

E.2.17 外部

外部类型与嵌入式 pdv 类型相似，但是标识选择更少。新规范通常更愿意使用嵌入式 pdv，因为其更大的灵活性和用某些编码规则对其值进行编码更有效的事实。

E.2.18 单一实例

E.2.18.1 用单一实例规定包含客体标识符字段和其类型是客体标识符确定的开放类型值的类型。如果用从 TYPE-IDENTIFIER(见 GB/T 16262.2 的附录 A 和附录 C)衍生来的类别的信息客体规定客体标识符值和类型之间的联系，只采用单一实例类型。

例：

```
ACCESS-CONTROL-CLASS::=TYPE-IDENTIFIER
Get-Invoke::=SEQUENCE {
        objectClass         ObjectClass,
        objectInstance      ObjectInstance,
```

```
        accessControl       INSTANCE OF ACCESS-CONTROL-CLASS,-- 这样通常是受
约束的。
        attributeID         ATTRIBUTE. & attributeId
        }
```

那么，调用相当于：

```
    Get-Invoke::=SEQUENCE {
        ObjectClass         ObjectClass,
        ObjectInstance      ObjectInstance,
        AccessControl       [UNVERSAL 8] IMPLICIT SEQUENCE {
            type-id         ACCESS-CONTROL-CLASS. &id,  -- 这样通常是受约束的。
            Value           [0] ACCESS-CONTROL-CLASS. &Type  -- 这样通常是受约束的。
        },
        attributeID ATTRIBUTE. & attributeId
    }
```

单一实例类型的真实效果要到用信息客体集合约束它才能看到，但是这样的示例超出了本部分的范围。信息客体集合的定义见 GB/T 16262.3，而怎样用信息客体集合约束单一实例类型见 GB/T 16262.3的附录 A。

E.2.19 相对客体标识符

E.2.19.1 用相对客体标识符类型来在已知客体标识符值的早期部分的上下文中以更简洁的形式转送客体标识符值。可能出现三种情况：

a) 客体标识符值的早期部分对给定规范(这是一个特定工业标准，而且所有的 OIDs 是相对于分配给标准化成员体的 OID 的)是固定的。这时，使用：

```
RELATIVE-OID  -- 相对客体标识符值是相对于{iso identified-organization set(22)}
```

b) 客体标识符值的早期部分常常是在规范的时间内已知的值，但是，通常可能是更通用的值。这时，使用：

```
CHOICE
    { a     RELATIVE-OID  -- 值是相对{1 3 22} --,
    b       OBJECT IDENTIFIER  -- 任意客体标识符值 --}
```

c) 客体标识符值的早期部分直到通信时才知道，但是，常常将是许多需要发送的值共同的，而且在规范时间总是已知的值。这时，使用：

```
SEQUENCE
    {oid-root       OBJECT IDENTIFIER DEFAULT {1  3  22},
    reloids         SEQUENCE OF RELATIVE-OID  -- 相对于 odi-root--}
```

E.3 标识抽象语法

E.3.1 常常用语义与单个 ASN.1 类型(在选择类型时典型)的每个值相关来定义协议。(该 ASN.1 类型有时非形式地引用为“应用的顶层类型”。)抽象值的集合形式上称为应用的抽象语法。抽象语法能通过给它一个 ASN.1 类型客体标识符的抽象语法名来标识。

E.3.2 能用 GB/T 16262.2 中定义的固有信息客体类别 ABSTRACT-SYNTAX 完成抽象语法的客体标识符赋值。这也用来清楚地标识应用层的顶层类型。

E.3.3 下面是可能在应用规范中出现的文本示例：

例：

```
Application-ASN1 DEFINITIONS ::=
```

```
        BEGIN
        EXPORTS Application -PDU;
        Application -PDU ::=CHOICE{
                connect-pdu ......,
                data-pdu CHOICE{
                    .....,
                    .....
                },
                ......
            }
            ......
        END
    Abstract-Syntax-Module DEFINITIONS ::=
        BEGIN
        IMPORTS Application -PDU FROM Application -ASN1;
    -- 本应用定义下面的抽象语法:
        Abstract-Syntax ABSTRACT-SYNTAX::=
            { Application -PDU IDENTIFIED BY
                application-abstract-syntax-object-id}
        application-abstract-syntax-object-id OBJECT IDENTIFIER ::=
            {joint-iso-itu-t asn1(1) examples(123) application-abstract-syntax(3)}
    -- 对应的客体描述符是:
    application -abstract-syntax-descriptor ObjectDescriptor::=
            " Example Application Abstract Syntax"
    -- ASN.1 客体标识符和客体描述符值:
        -- 编码规则客体标识符
        -- 编码规则客体描述符
    -- 指派给 GB/T 16263.1 或 GB/T 16263.2 的编码规则能用作和
    -- 这一传送语法一起的传送语法标识符。
    END
```

E.3.4 为了保证能互工作,标准可能额外地标识一个强制传送语法(典型是,GB/T 16263.1 或 GB/T 16263.2或 ISO/IEC 8825-3 的编码规则中定义的那些之一)。

E.4 子类型

E.4.1 用子类型限制特殊情况下允许的现存类型的值。

例:

```
AtomicNumber::= INTEGER (1..104)
TouchToneString::=IA5String
            (FROM (" 0123456789" | "*" | "# ")) (size (1..63))
ParameterList::=SET SIZE (1..63) OF Parameter
SmallPrime::=INTEGER (2|3|5|7|11|13|17|19|23|29)
```

E.4.2 用可扩展子类型约束来为允许的值集小且很好定义了,但预计会增加的 INTEGER 类型建模。

例:

```
SmallPrime::=INTEGER (2 | 3,...)  -- SmallPrime 的第一版
```

在预料中：

```
SmallPrime::=INTEGER (2 | 3,...,5 | 7 | 11)
-- SmallPrime 的第二版的
```

且以后还有：

```
SmallPrime::=INTEGER (2 | 3,...5 | 7 | 11 | 13 | 17 | 19)
-- SmallPrime 的第三版
```

注：对于某些类型，某些编码规则(如 PER)为子类型约束扩展根值(即："..."之前出现的值)提供极好的优化编码，而为子类型约束扩展附加值(即："..."之后出现的值)提供较少的优化编码，而在某些其他编码规则(如 BER)中子类型约束不影响编码。

E.4.3 当两个或多个相关的类型有重要共性时，考虑显性定义其共同的双亲为一个类型而对单独的类型使用子类型。这样使其关系和共性清楚，并鼓励(尽管不强迫)这样随类型发展而继续进行。因此，他便于采用处理这些类型的值的共同实现方法。

例：

```
Envelope ::=SET {
            typeA    TypeA,
            typeB    TypeB    OPTIONAL,
            typeC    TypeC    OPTIONAL}
   -- 共同的双亲
ABEnvelope::= Envelope (WITH  COMPONENTS
            {...,
               typeB  PRESENT,  typeC  ABSENT})
   -- typeB 必须总是出现，而 typeC 不能出现
ACEnvelope::= Envelope (WITH  COMPONENTS
            {...,
typeB  ABSENT,  typeC  PRESENT})
   -- typeC 必须总是出现，而 typeB 不能出现
```

后者的定义能用下面的表达式替换：

```
ABEnvelope ::= Envelope (WITH COMPONENTS {typeA, typeB})
ACEnvelope ::= Envelope (WITH COMPONENTS {typeA, typeC})
```

根据双亲类型中成分数量、以及那些可选择的数量、个体类型之间差异的程度和可能发展的策略等因素考虑替换项之间的选择。

E.4.4 用子类型来部分地定义值，例如，在测试仅关心 PDU 的某些成分时，一致性测试中用来测试的协议数据单元。

例：

```
Given:
   PDU::=SET
      {alpha      INTEGER,
      beta        IA5String  OPTIONAL,
      gamma       SEQUENCE OF Parameter,
      delta       BOOLEAN}
```

那么在要求布尔为假和整数为负的综合测试中，写作：

```
TestPDU::=PDU(WITH COMPONENTS
```

```
                    {...,
                    delta(FALSE),
                    alpha(MIN..<0)})
```

而进一步，如果 IA5String、beta 出现，而且长度为 5 或 12 个字符，则写成：

```
FurtherTestPDU ::= TestPDU (WITH COMPONENTS {..., beta (SIZE (5 | 12))PRESENT
})
```

E.4.5 如果一般目的数据类型已经被定义为 SEQUENCE OF，用子类型来定义一般类型的受限制子类型。

例：

```
Text-block ::= SEQUENCE OF VisibleString
Address ::= Text-block(SIZE(1..6))(WITH COMPONENT (SIZE (1..32)))
```

E.4.6 如果一般目的数据类型已经被定义为 CHOICE，用子类型来定义一般类型的受限制子类型。

例：

```
Z ::=CHOICE{
        a    A,
        b    B,
        c    C,
        d    D,
        e    E
    }
V ::=Z (WITH COMPONENTS {..., a  ABSENT,  b  ABSENT }) --a 和 b 必须缺失，
                                                --c、d 或 e 可在值中出现。
W ::=Z (WITH COMPONENTS {...,a  PRESENT }) - 只有 a 能出现(见 47.8.9.2)。
X ::= Z (WITH COMPONENTS {a  PRESENT }) - 只有 a 能出现(见 47.8.9.2)。
Y ::= Z (WITH COMPONENTS {a  ABSENT, b,c }) -- a、d 和 e 必须缺失，
                                                --b 或 c 可在值中出现。
```

注：W 和 X 语义相同。

E.4.7 用包含的子类型来从现存的子类型中形成新子类型。

例：

```
Months ::=ENUMERATED {
        january         (1),
        february        (2),
        march           (3),
        april           (4),
        may             (5),
        june            (6),
        july            (7),
        august          (8),
        september       (9),
        october         (10),
        november        (11),
        december        (12)}
First-quarter ::=Months(january | february | march)
```

```
Second-quarter::= Months (april | may | june)
Third-quarter::= Months (july | august | september)
Fourth-quarter::= Months (october | november | december)
First-half::= Months(First-quarter | Second-quarter)
Second-half::= Months(Third-quarter | Fourth-quarter)
```

附 录 F
（资料性附录）
ASN.1 字符串的辅导附录

F.1 ASN.1 中的字符串支持

F.1.1 在 ASN.1 中有四个字符串支持组。这四个组是：

a) 基于与转义序列一起使用的编码字符集的 ISO 国际登记（即：基于 GB/T 1988 的结构）和相关编码字符集国际登记的，以及类型 VisibleString、IA5String、TeletexString、VideotexString、GraphicString 和 GeneralString 提供的字符串类型；

b) 基于 GB/T 13000.1 的，用有 GB/T 13000.1 中定义的子集的 UniversalString、UTF8String 或 BMPString 类型的分成子集或用已命名字符提供的字符串类型；

c) 提供本部分中规定的字符的简单小集的，以及预定特殊用途的字符串类型。这些是 NumericString 和 PrintableString 类型；

d) 使用类型 CHARATER STRING，与所用的字符集协商（或宣称使用的集）；它允许实现采用赋给了 OBJECT IDENTIFIERs（包括与转义序列一起使用的编码字符集的 ISO 国际登记）、ISO/IEC 7350、GB/T 13000.1 的字符集和编码的任意集合，以及字符和编码的专用集合（文件可能强制要求或限制所用的字符集-字符抽象语法）。

F.2 UniversalString，UTF8String 和 BMPString 类型

F.2.1 UniversalString 和 UTF8String 类型携带取自 GB/T 13000.1 的任何字符。GB/T 13000.1 中的字符集对于所要求的意义相一致的场合通常太大，而且正常情况下，应该分成 GB/T 13000.1—1993 附录 A 中字符的标准集合组合的子集。

F.2.2 BMPString 类型携带取自 GB/T 13000.1 基本多文种平面的任何字符。正常情况下，基本多文种平面应该分成 GB/T 13000.1 附录 A 中字符的标准集合组合的子集。

F.2.3 对于 GB/T 13000.1 附录 A 中定义的集合，存在固有 ASN.1 模块“ASN1-CHARACTER-MODULE”（见第 38 章）中定义的类型引用。“子类型约束”机制允许定义现存子类型组合的 UniversalString 的新子类型。

F.2.4 在 ASN1-CHARACTER-MODULE 中定义的类型引用的示例与它们对应 GB/T 13000.1 的汇集名是：

BasicLatin	BASIC LATIN
Latin-1 Supplement	LATIN-1 SUPPLEMNT
LatinExtended-a	LATIN EXTENDED-A
LatinExtended-b	LATIN EXTENDED-B
IpaExtensions	IPA EXTENSIONS
SpacingModifierLetters	SPACING MODIFIER LETTERS
CombiningDiacriticalMarks	COMBINING DIACRITICAL MARKS

F.2.5 GB/T 13000.1 规定三个“实现级”，并要求 GB/T 13000.1 的所有用法规定实现级。

实现级与为字符汇中组合用字符给出支持的程度有关，因此，用 ASN.1 术语，定义 UniversalString 和 BMPString 受限字符串类型的子集。

在 1 级实现中，不允许组合字符，而且通常 ASN.1 字符串中的抽象字符与串的物理表现中的打印字符一一对应。

在2级实现中,可以使用某些组合用字符(列于GB/T 13000.1附录B中),但是也有禁止其使用的其他情况。

在3级实现中,不限制使用组合用字符。

F.2.6 用如下的子类型记法能限制BMPString和UniversalString排除所有的控制功能:

```
VanillaBMPString::=BMPString(FROM (ALL EXCEPT ({0,0,0,0}..{0,0,0,31}|
                                                {0,0,0,128}..{0,0,0,159})))
```

或相当的:

```
C0::= BMPString(FROM ({0,0,0,0}..{0,0,0,31}))    -- C0控制功能
C1::= BMPString(FROM ({0,0,0,128}..{0,0,0,159}))  -- C1控制功能
VanillaBMPString::=BMPString (FROM (ALL EXCEPT (C0 | C1)))
```

F.3 关于GB/T 13000.1一致性要求

使用ASN.1类型定义中UniversalString、BMPString或UTF8String(或其子类型)要求指出GB/T 13000.1的一致性要求。

这些一致性要求需要使用这种ASN.1类型的标准(例如X)实现者为其标准X的实现提供GB/T 13000.1合适的子集和实现级(支持组合字符)的声明。

使用规范中UniversalString、BMPString或UTF8String的ASN.1子类型要求实现支持那个ASN.1子类型中包括的全部GB/T 13000.1字符,也就是,实现的合适子集中(至少)出现那些字符。也要求对所有那种ASN.1子类型支持陈述的级。

注:ASN.1规范(缺少抽象语法参数和例外规范)确定被传输的(最大)字符集和收到后进行处理的(最小)字符集。被采用的GB/T 13000.1的集要求不传输超过这个集的字符,而且收方支持这个集内的所有字符。因此,采用的集要求确切地是ASN.1规范允许的所有字符的集。下面讨论抽象语法有参数的情形。

F.4 对ASN.1用户的GB/T 13000.1一致性建议

ASN.1用户应该清楚,如果满足他们的标准的实现,将形成采用的实现子集(和要求的实现水平)的GB/T 13000.1字符集。

通过定义包含标准需要的所有字符的UniversalString、UTF8String或BMPString的ASN.1子类型可以方便地完成这一任务,如果合适的话,并通过限制它到"Level 1"或"Level 2"。这个类型的方便名字可能是"ISO-10646-String"。

例:

```
ISO-10646-String::=BMPString
    (FROM (Level2 INTERSECTION(BasicLatin UNION HebrewExtended UNION Hiragana)))
    -- 这是定义为实现级实现该标准采用的子集中字符最小集的类型。
    -- 要求该次至少是2级。
```

在OSI环境中,OSI协议实现一致性声明那时将包含一个简单的声明:采用的GB/T 13000.1子集是"ISO-10646-String"定义的、有限子集(和级),而且,"ISO-10646- String"(可能分成了子类型)将在包含ISO/IEC 10646-1串的整个标准中使用。

EXAMPLE COMFORMANCE STATEMENT

采用的GB/T 13000.1子集是由模块<模块名称>中定义的ASN.1类型"ISO-10646-String"中的所有字符组成的、有限子集,具有2级实现。

EXAMPLE USE IN PROTOCOL

```
Message::= SEQUENCE {
    first-field ISO-10646-String,     -- 采用子集中的所有字符会出现
```

```
        second-field ISO-10646-String
            (FROM  (latinSmallLetterA..latinSmallLetterZ))  -- 只有小写拉丁字母
        third-field ISO-10646-String, (FROM  (digitZero..digitNine))  -- 只有数字
    }
```

F.5 用子集作为抽象语法的参数

GB/T 13000.1 要求显式地定义实现采用的子集和级数。当 ASN.1 用户不希望约束被定义标准的某些部分中的 GB/T 13000.1 字符字段时，这可以通过定义 ISO-10646-String(例如)为 UniversalString、BMPString 或 UTF8String 的子类型，它们带由(或包括)留下作为抽象语法参数的 ImplementorsSubset 构成的子类型约束。

告诫 ASN.1 用户，在这样的情况下，合格的发送者可传送不能被接受者处理的字符给合格的接收者，因为它们处于接收者(实现依赖)采用的子集或级数之外，同时建议在这样的情况下，ISO-10646-String 定义中包括例外处理规范。

例：

```
ISO-10646-String{UniversalString:ImplementorsSubset,ImplementationLevel}::=
    UniversalString (FROM((ImplementorsSubset UNION BasicLatin)
        INTERSECTION ImplementationLevel)! characterSetProblem)
-- GB/T 13000.1 采用的子集应该包括"BasicLatin"，但可能也包括抽象语法参数
-- "ImplementorsSubset"中规定的任何附加字符。抽象语法参数"ImplementationLevel"
-- 定义实现级数。合格的接收者必须准备接收它采用的子集和实现级数之外的字符。
-- 这时，调用关于"characterSetProblem"的第<这里增加你的章号>章规定的例外处理。
-- 注意，如果通信实例中采用的实际字符限制为"BasicLatin"，
-- 合格的接收者从不可能调用它。
my-Level2-String::= ISO-10646-String{{HebrewExtended UNION Hiragana},Level2}
```

F.6 CHARACTER STRING 类型

F.6.1 CHARACTER STRING 类型给出字符集的选择和编码方法中的完整灵活性。

注：当单个连接提供端到端数据传送(无中继)时，而且使用 OSI 协议，那么，所用的字符集协商及其编码能作为字符抽象语法的 OSI 表达式上下文的定义的一部分完成。否则，由一对客体标识符值声明抽象和传送字符语法(字符表和编码)。

F.6.2 在形式术语中，字符抽象语法是对可能值(它们是所有字符串，而且确实是取自字符的某些集合形成的所有字符串)有某些限制的普通抽象语法。因此，对字符抽象和传送语法的客体标识符值的分配以普通的方式进行。

F.6.3 CHARACTER STRING 编码要声明所用字符汇中的抽象和传送语法(即，字符集和编码)。在 OSI 环境中，这两种语法都可能协商。

F.6.4 字符抽象语法(和对应的字符传送语法)已经在许多我国标准、ITU-T 建议和国际标准中定义，而且任何能分派客体标识符的组织能定义附加字符抽象语法(和/或字符传送语法)。

F.6.5 在 GB/T 13000.1 中，具有为整个字符集合、为每个子集(BASIC LATIN，BASIC SYMBOLSN 等等)定义的字符集合，以及为字符集合定义的每个可能组合所定义的字符抽象语法(和分派的客体标识符)。还定义了两个字符传送语法来标识 GB/T 13000.1 中的各种选择(特别是 16 位和 32 位)。

附 录 G
（资料性附录）
类型扩展 ASN.1 的辅助附录

G.1 概述

G.1.1 ASN.1 类型可能会随着时间的过去通过称为扩展附加的系列扩展的手段从 extension root 类型转变而来。

G.1.2 能进行特定实现的 ASN.1 类型可能是扩展根类型，或可能是扩展根类型加一个或多个扩展附加。每个这种包含扩展附加的 ASN.1 类型也包含所有前面定义的扩展附加。

G.1.3 将本系列中的 ASN.1 类型定义说成是相关扩展（“相关扩展”的更精确定义见 3.6.32），而且要求编码规则以下面的方式对相关扩展类型编码：如果两个系统使用相关扩展的两个不同类型，在两个系统之间的转换将成功地传送两个系统共同的相关扩展类型那部分的信息内容。也要求不属于两个系统共同的那部分能解除限制并在后面的传输中重新传输（也许对第三方），倘若使用相同的传送语法。

注：发送者可能用扩展附加系列中较早或者较晚的类型。

G.1.4 逐步增加根类型获得的系列类型称为扩展系列。为了这些编码规则有合适的准备来传输相关扩展类型（可能要求行里有多位），这种类型（包括扩展根类型）需要语法上加标志。标志是一个省略号（...），并称作扩展标志。

例：

```
Extension root type     1st extension          2nd extension          3rd extension
A ::=SEQUENCE {         A ::= SEQUENCE {       A ::=SEQUENCE {        A ::=SEQUENCE {
    a INTEGER,              a INTEGER,             a INTEGER,             a INTEGER,
    ...                     ...,                   ...,                   ...,
    }                       b BOOLEAN,             b BOOLEAN,             b BOOLEAN,
                            c INTEGER              c INTEGER,             c INTEGER,
                        }                          d SEQUENCE {           d SEQUENCE{
                                                     e INTEGER,             e INTEGER,
                                                       ...,                   ...,
                                                       ...,                 g BOOLEAN OPTIONAL,
                                                     f IA5STring            h BMPString,
                                                   }                          ...,
                                                }                           f IA5String
                                                                          }
                                                                      }
```

G.1.5 将序列、集和选择类型中的所有扩展附加插入在扩展标志对之间。允许有单个扩展标志，只要（在扩展根类型中）它在类型中作为最后一项出现，这时，假设一个匹配的扩展标志正好在类型的结束括号之前存在；在这些情况下，将所有扩展附加插入在类型的末尾。

G.1.6 有扩展标志的类型能嵌套在没有的类型内，或能嵌套在扩展根中的类型内，或者能嵌套在扩展附加类型内。在这些情况下，应独立地对待扩展系列，而且有扩展标志的被嵌套类型没有与它嵌套其中的类型相冲突。在任何规定的结构中只能出现一个扩展插入点（如果使用单个扩展标志，则在类型末尾，或者如果使用一对扩展标志，则刚好在第二个扩展标志之前）。

G.1.7 单个扩展附加组（一个或多个嵌套在“[[”“]]”内的类型）或单个加在扩展插入点上的类型的术

语中定义了扩展系列中的新扩展附加。在下面的示例里，第一种扩展定义了扩展附加组，其中 b 和 c 必须在类型“A”的值中同时出现或同时缺失。第二种扩展定义了单个成分类型 d，它可能在类型“A”的值中缺失。第三种扩展定义了扩展附加组，其中只要值中出现新增加的扩展附加组，h 必须在类型“A”的值中出现。

例：

```
Extension root type      1st extension          2nd extension          3rd extension
  A ::=SEQUENCE {  A ::= SEQUENCE {  A ::=SEQUENCE {  A ::=SEQUENCE {
    a INTEGER,       a INTEGER,        a INTEGER,       a INTEGER,
      ...              ...,              ...,             ...,
    }                [[                [[               [[
                       b BOOLEAN,        b BOOLEAN,       b BOOLEAN,
                       c INTEGER         c INTEGER        c INTEGER
                     ]]                ]],              ]],
                   }                   d SEQUENCE {     d SEQUENCE{
                                         e INTEGER,       e INTEGER,
                                           ...,             ...,
                                           ...,             [[
                                         f IA5String      g BOOLEAN OPTIONAL,
                                       }                  h BMPString,
                                     }                    ]],
                                                            ...,
                                                          f IA5String
                                                        }
                                                    }
```

G.1.8 也可能为版本括号增加新的版本号，但是只有它出现在模块内的所有括号中，而且只有模块中的所有扩展在版本括号内时才如此。建议使用版本号。省略号码和版本括号的能力是历史的原因。(本部分的较早版本中不允许有版本括号和版本号。)(也见 G.3。)

G.1.9 随着时间的过去，对扩展附加将增加正常实用时，重要的 ASN.1 模块和规范不包括时间。如果一个能通过扩展附加从另一个“生成”，则这两个类型是相关扩展。也就是，一个包含另一个的所有成分。可能有在相反方向(尽管不太可能)“生成”的类型。随着时间的过去，它甚至可能是一个以许多逐步删去的扩展附加开头的类型。ASN.1 及其编码规则关注的一切是一对扩展规范是否是相关扩展。如果它们是，那么所有 ASN.1 编码规则保证它们的用户之间能互工作。

G.1.10 我们从一个类型开始，然后，如果我们后来要扩展它，决定是否要与较早的版本实现互工作。如果是的话，我们要包含现在的扩展标志。然后，我们能增加后面的扩展附加给带通过较早的系统定义好处理扩展值的类型。然而，注意增加一个扩展标志给以前没有(或删去扩展标志)的类型可能阻碍互工作。

注：当使用 ECN 时，在第二版中可能在第一版中没有扩展标志的地方增加扩展，而且，在第一版与第二版之间仍然保持互工作。

G.1.11 表 G.1 表明能形成 ASN.1 扩展系列的扩展根类型的 ASN.1 类型，以及那一类型(当然，连续或作为扩展组一起能够成多扩展附加)允许的单个扩展附加的性质。

表 G.1 扩展附加

扩展根类型	扩展附加性质
ENUMERATED	在“AdditionalEnumeration”后面附加单个进一步枚举，用比已经增加的任何枚举的都大的枚举值
SEQUENCE 和 SET	在“ExtensionAdditionList”后面附加单个类型或扩展附加组。是扩展附加(不包含在扩展附加组中)的“ComponentType”不要求被标志 OPTIONAL 或 DEFAULT，尽管这将是经常发生的情况
CHOICE	在“ExtensionAdditionAlternativesList”后面附加单个“NamedType”
约束记法	将单个“AdditionalElementSetSpec”附加到“ElementSetSpecs”记法中

G.2 版本号的意义

G.2.1 在 BER 和 PER 编码中不使用版本号。由 ECN 规范确定它们在 ECN 编码中的用途(如果有的话)。

G.2.2 当它们与解码完整 PDU 的方式而不是与单个的类型相关时版本号最有用。类型用作几个协议的成分并且因此贡献给不同的完整 PDU 时，那个类型的附加通常要求增加它贡献的所有 PDU 的版本号。

G.2.3 当用来提供被开发系统间的互工作时，版本号应该以下面的方式用在扩展附加组：被开发的系统有带给出版本号(不管它们在协议内哪里出现)的所有扩展附加组的语法和语义知识，以及带较早版本号的所有扩展附加组的知识。ECN 规定者通常将假设已经按照这一原则分配了版本号(给适用于 ECN 的类型的所有部分)。

G.3 编码规则要求

G.3.1 抽象语法能定义为可扩展类型的单个 ASN.1 类型的值。然后，它含有所有能用加上或删去扩展附加获得的值。这样的抽象语法称作相关扩展抽象语法。

G.3.2 相关扩展抽象语法的很好形成的编码规则集满足 G.3.3 至 G.3.5 中描述的附加要求。

注：所有 ASN.1 编码规则满足这些要求。

G.3.3 将抽象值转换成编码来传输和将收到的编码转换成抽象值的程序定义应该认识到，发送者和接收者使用不等同，但均为相关扩展的抽象语法的可能性。

G.3.4 在这种情况下，编码规则应保证发送者有比接收者在扩展系列中较早的类型规范时，发送者的值应该以接收者能确定扩展附加不出现的方式传送。

G.3.5 编码规则应该保证发送者有比接收者在扩展系列中更晚的类型规范时，应能将那个类型的值传送到接受者。

G.4 (可能可扩展的)约束的组合

G.4.1 模块

G.4.1.1 应用约束的基本 ASN.1 模块是简单的：类型是抽象值集，而且加在其上的约束选择这些抽象值的子集。如果未约束的类型是不可扩展的，那么产生的类型如果且只有所应用的约束被定义为可扩展的时才定义为可扩展的。

G.4.1.2 即使在这种简单的情况下，要澄清一个特点：类型可能在形式上可扩展，即使从来不可能有任何扩展附加。考虑：

```
A ::= INTEGER (MIN .. MAX,..., 1..10)
```

像本附录中给出的许多示例一样，这是没人曾经写过的东西，但是哪个工具零售商都必须写代码，

因为 ASN.1 标准给出的是简单和通用的，而因此，这个示例是合法的 ASN.1。在这个示例中，A 在形式上是可扩展整数，并具有根中整数值的整个范围。

G.4.1.3 复杂性的三个主要来源：

——约束应用于已经有可扩展约束作用于它的类型(约束的系列应用——见 G.4.2)

——用 UNION 和 INTERSECTION 和 EXCEPT 组合可扩展约束(集算法——见 G.4.3)

——当类型引用去引用可扩展类型时(也许有实际扩展附加——见 G.4.4)，使用约束的集算法中的类型引用(包含的子类型)。

G.4.2 约束的系列应用

G.4.2.1 当类型受到约束(在类型引用的赋值中)和类型引用随后与应用于它的进一步约束一起使用时，出现约束的系列应用。

G.4.2.2 当类型以系列的方式有多种约束直接应用于它时，它也能出现，但是不常见。本附录中的很多示例采用后面的形式，但是类型引用链接两个(或多个)约束的情况是实际规范中通常出现系列应用的形式。

G.4.2.3 在约束的系列应用中有以下两个重点：

——如果受约束类型是可扩展的(和也许已扩展的)，如果随后系列应用进一步约束，则丢弃“可扩展”标志和所有扩展附加。受约束类型(和任何扩展附加)的可扩展性仅仅依赖于所用的最后约束，能只引用要进一步受约束(双亲类型)的类型的根中的值。在根中或结果类型的扩展附加中包含的值只能是双亲类型根中的值。

——约束的系列应用(对复杂的情况)与集算法交叉不一样，即使在不包括可扩展性时也如此。首先，解释 MIN 和 MAX 的环境，其次是能在第二个约束中引用的抽象值在系列应用中与规定两个约束为取自共同双亲的交叉值时的情况完全不同。

注：在整数类型的约束中使用 20..28 这样的范围是合法的，如果(也只有)20 和 28 都在双亲类型(根)中，但是该范围规范引用的值仅仅是双亲(根)中的那些。因此，如果已经约束排除值 24 和 25，则范围 20..28 只引用 20 至 23 和 26 至 28。

下面有一些示例：

```
A1 ::=INTEGER (1..32,..., 33..128)
    -- A1 是可扩展的，并且包含值 1 至 218，1 至 32 在根中，而且 33 至 128 作为扩展附加。
B1 ::=A1 (1..128)
    --或相当的
B1 ::= INTEGER (1..32,..., 33..128) (1..128)
    --这些是非法的，因为 128 不在双亲类型中，当进一步约束它时失去它的扩展附加
B2 ::=A1(1..16)
    -- 这是合法的。B2 不是可扩展的，而且包含 1 至 16。
A2 ::=INTEGER (1..32)(MIN..63)
    --MIN 是 1，而 63 是非法的。
A3 ::=INTEGER ((1..32)  INTERSECTION (MIN..63))
    --这是合法的。MIN 是负无穷，而 A3 包含 1 至 32。
```

G.4.3 使用集算法

G.4.3.1 结果在很大程度上凭直觉，并遵循交叉、联合和集差(EXCEPT)的通常数学规则。特别是，交叉和联合都是可交换的，即：

(<some set 1 of values> INTERSECTION <some set 2 of values>

和

(<some set 2 of values > INTERSECTION< some set 1 of values >

对于 UNION 类似地一样。

G.4.3.2 可交换性是真实的，不管什么值集是可扩展的，也不管出现什么扩展附加。

G.4.3.3 如果交叉不能使扩展附加值存在将出现误解。这与 INTEGER(MIN..MAX,...)情况类似。

G.4.3.4 例如：

```
A ::= INTEGER ((1..256,...,B)  INTERSECTION (1..256))
-- A 总是(只)包含值 1..256，不管 B 包含什么值，但是在形式上仍然可扩展。
```

G.4.3.5 记住这点也很重要：在进一步、且包含的子类型失去其可扩展性和扩展附加时，只要双亲类型失去其可扩展性和扩展附加，集算法中直接规定的值集既不失去其可扩展性也不失去其扩展附加。

G.4.3.6 由集算法产生的值集可扩展性规则已在 46.4 和 46.4 中清楚地陈述了，而且不依赖于集算法是否可能产生实际的扩展附加。

G.4.3.7 这里，为了完整而总结了规则，其中用 E 指明有“可扩展”标志集的值集，用 N 指明在形式上无可扩展的值集。用 R 指明每个集的根中的值，用 R 指明扩展附加(若有的话)，而且为每种情况给出产生的内容。

注 1：为了本附录的目的，并为了简化说明，如果值集不是可扩展的，那么我们将它的各个值都描述为根值。

注 2：如果系列应用约束中使用的值的任何产生集的根是空，这是非法的规范。

注 3：为避免在下面冗长，在“扩展附加”更正确的地方使用“扩展”。

G.4.3.8 这些规则是：

```
N1 INTERSECTION N2=>N
        Root:R1 INTERSECTION R2
N1 INTERSECTION E2=>E
        Root: R1 INTERSECTION R2, Extensions: R1 INTERSECTION X2
E1 INTERSECTION E2=>E
        Root: R1 INTERSECTION R2, Extensions: ( (R1 UNION X1)
                                              INTERSECTION
                                              (R2 UNION X2)
                                              EXCEPT
                                              (R1 INTERSECTION R2)
N1 UNION N2=>N
        Root: R1 UNION R2
N1 UNION E2=>E
        Root: R1 UNION R2, Extensions: X2
E1 UNION E2 =>E
        Root: R1 UNION R2, Extensions: (R1 UNION X1 UNION R2 UNION X2)
                                              EXCEPT
                                              (R1 UNION R2)
N1 EXCEPT N2=>N
        Root: R1 EXCEPT R2
N1 EXCEPT E2=>E
        Root: R1 EXCEPT R2
E1 EXCEPT N2=>E
        Root: R1 EXCEPT R2, Extensions: (X1 EXCEPT R2)
                                              EXCEPT
```

(R1 EXCEPT R2)

E1 EXCEPT E2=>E

Root: R1 EXCEPT R2, Extensions: (X1EXCEPT (R2 UNION X2))

EXCEPT

(R1 EXCEPT R2)

N1...N2=>E

Root: R1, Extensions:R2 EXCEPT R1

E1...N2=>E

Root: R1, Extensions:X1 UNION R2

EXCEPT

R1

N1...E2=>E

Root: R1, Extensions: R2 UNION X2

EXCEPT

R1

E1...E2=>E

Root: R1, Extensions: X1 UNION R2 UNION E2

EXCEPT

R1

注：如果值的可扩展集的集算法结果没有实际的扩展附加，或者甚至决不可能有实际扩展附加(不管什么扩展附加加到可扩展输入上)，对于上面的结果E，其结果在形式上仍然被定义为可扩展的。

G.4.4 包含的子类型可能是可扩展的，也可能不是可扩展的，但是当它用在集算法中时，它常常当作不可扩展的来处理，而且所有它的扩展附加被丢弃。

附 录 H
（资料性附录）
ASN.1 记法总结

在第 11 章中定义的下列词法项：

typereference
identifier
valuereference
modulereference
comment
empty
number
realnumber
bstring
hstring
cstring
xmlbstring
xmlhstring
xmlcstring
xmlasn1typename
"true"
"false"
"::="
"[["
"]]"
".."
"..."
"</"
"/>"
"{"
"}"
"<"
">"
","
"."
"("
")"
"["
"]"
"-"
":"
"="

" " " (QUOTATION MARK)
" ' " (APOSTROPHE)
" " (SPACE)
";"
"@"
"|"
"!"
"~"
ABSENT
ABSTRACT-SYNTAX
ALL
APPLICATION
AUTOMATIC
BEGIN
BIT
BMPString
BOOLEAN
BY
CHARACTER
CHOICE
CLASS
COMPONENT
COMPONENTS
CONSTRAINED
CONTAINING
DEFAULT
DEFINITIONS
EMBEDDED
ENCODED
END
ENUMERATED
EXCEPT
EXPLICIT
EXPORTS
EXTENSIBILITY
EXTERNAL
FALSE
FROM
GeneralizedTime
GeneralString
GraphicString
IA5String
IDENTIFIER

IMPLICIT
IMPLIED
IMPORTS
INCLUDES
INSTANCE
INTEGER
INTERSECTION
ISO646String
MAX
MIN
MINUS-INFINITY
NULL
NumericString
OBJECT
ObjectDescriptor
OCTET
OF
OPTIONAL
PATTERN
PDV
PLUS-INFINITY
PRESENT
PrintableString
PRIVATE
REAL
RELATIVE-OID
SEQUENCE
SET
SIZE
STRING
SYNTAX
T61String
TAGS
TeletexString
TRUE
TYPE-IDENTIFIER
UNION
UNIQUE
UNIVERSAL
UniversalString
UTCTime
UTF8String
VideotexString

```
    VisibleString
    WITH
```

在本部分中使用的下列产生式，并且用上面的词法项作为终止符号：

```
    ModuleDefinition ::= ModuleIdentifier
            DEFINITIONS
            TagDefault
            ExtensionDefault
            "::="
            BEGIN
            ModuleBody
            END
    ModuleIdentifier ::= modulereference
                         DefinitiveIdentifier
    DefinitiveIdentifier ::="{" DefinitiveObjIdComponentList "}"  |
                         empty
    DefinitiveObjIdComponentList ::=
            DefinitiveObjIdComponent  |
            DefinitiveObjIdComponent DefinitiveObjIdComponentList
    DefinitiveObjIdComponent ::=
            NameForm |
            DefinitiveNumberForm  |
            DefinitiveNameAndNumberForm
    DefinitiveNumberForm ::= number
    DefinitiveNameAndNumberForm ::= identifier "(" DefinitiveNumberForm ")"
    TagDefault ::=
              EXPLICIT TAGS  |
              IMPLICIT TAGS  |
              AUTOMATIC TAGS  |
              empty
    ExtensionDefault ::=
              EXTENSIBILITY IMPLIED | empty
    ModuleBody ::= Exports Imports AssignmentList  |
                   empty
    Exports ::= EXPORTS SymbolsExported ";"
                   EXPORTS ALL ";"  |
                   empty
    SymbolsExported ::= SymbolList  |
                        empty
    Imports ::= IMPORTS SymbolsImported ";"  |
                empty
    SymbolsImported ::= SymbolsFromModuleList
                empty
    SymbolsFromModuleList ::=
```

```
                SymbolsFromModule   |
                SymbolsFromModuleList SymbolsFromModule
SymbolsFromModule ::= SymbolList FROM GlobalModuleReference
GlobalModuleReference ::= modulereference AssignedIdentifier
AssignedIdentifier ::= ObjectIdentifierValue   |
        DefinedValue   |
        empty
SymbolList ::= Symbol | SymbolList "," Symbol
Symbol ::= Reference | ParameterizedReference
Reference ::=
        Typereference        |
        valuereference       |
        objectclassreference |
        objectreference      |
        objectsetreference
AssignmentList ::= Assignment | AssignmentList Assignment
Assignment ::=
        TypeAssignment          |
        ValueAssignment         |
        XMLValueAssignment      |
        ValueSetTypeAssignment  |
        ObjectClassAssignment   |
        ObjectAssignment        |
        ObjectSetAssignment     |
        ParameterizedAssignment
DefinedType ::=
        ExternalTypeReference   |
        typereference           |
        ParameterizedType       |
        ParameterizedValueSetType
ExternalTypeReference ::=
        modulereference
        "."
        typereference
NonParameterizedTypeName ::=
        ExternaTypeReference   |
        typereference          |
        xmlasn1typename
DefinedValue ::=
        ExternalValueReference  |
        valuereference          |
        ParameterizedValue
ExternalValueReference ::=
```

```
        modulereference
        "."
        valuereference
AbsoluteReference ::= "@" ModuleIdentifier
        "."
        ItemSpec
ItemSpec ::=
        typereference  |
        ItemId "." ComponentId
ItemId ::= ItemSpec
ComponentId ::=
        identifier | number | "*"
TypeAssignment ::= typereference
                   "::="
                   Type
ValueAssignment ::= valuereference
                    Type
                    "::="
                    Value
XMLValueAssignment ::=
        valuereference
        "::="
        XMLTypedValue
XMLTypedValue ::=
        "<" & NonParameterizedTypeName ">"
        XMLValue
        "</" & NonParameterizedTypeName ">"  |
        "<" & NonParameterizedTypeName "/>"
ValueSetTypeAssignment ::= typereference
                           Type
                           "::="
                           ValueSet
ValueSet ::= "{" ElementSetSpecs "}"
Type ::= BuiltinType | ReferencedType | ConstrainedType
BuiltinType ::=
        BitStringType        |
        BooleanType          |
        CharacterStringType  |
        ChoiceType           |
        EmbeddedPDVType      |
        EnumeratedType       |
        ExternalType         |
        InstanceOfType       |
```

```
        IntegerType          |
        NullType             |
        ObjectClassFieldType |
        ObjectIdentifierType |
        OctetStringType      |
        RealType             |
        RelativeOIDType      |
        SequenceType         |
        SequenceOfType       |
        SetType              |
        SetOfType            |
        TaggedType
NamedType ::= identifier Type
ReferencedType ::=
        DefinedType          |
        UsefulType           |
        SelectionType        |
        TypeFromObject       |
        ValueSetFromObjects
Value ::= BuiltinValue | ReferencedValue | ObjectClassFieldValue
XMLValue ::= XMLBuiltinValue | XMLObjectClassFieldValue
BuiltinValue ::=
        BitStringValue       |
        BooleanValue         |
        CharacterStringValue |
        ChoiceValue          |
        EmbeddedPDVValue     |
        EnumeratedValue      |
        ExternalValue        |
        InstanceOfValue      |
        IntegerValue         |
        NullValue            |
        ObjectIdentifierValue |
        OctetStringValue     |
        RealValue            |
        RelativeOIDValue     |
        SequenceValue        |
        SequenceOfValue      |
        SetValue             |
        SetOfValue           |
        TaggedValue
XMLBuiltinValue ::=
        XMLBitStringValue    |
```

```
        XMLBooleanValue          |
        XMLCharacterStringValue  |
        XMLChoiceValue           |
        XMLEmbeddedPDVValue      |
        XMLEnumeratedValue       |
        XMLExternalValue         |
        XMLInstanceOfValu        |
        XMLIntegerValue          |
        XMLNullValue             |
        XMLObjectIdentifierValue |
        XMLOctetStringValue      |
        XMLRealValue             |
        XMLRelativeOIDValue      |
        XMLSequenceValue         |
        XMLSequenceOfValue       |
        XMLSetValue              |
        XMLSetOfValue            |
        XMLTaggedValue
ReferencedValue ::=
        DefinedValue  |
        ValueFromObject
NamedValue ::= identifier Value
XMLNamedValue ::=
        "<" & identifier ">" XMLValue "</" & identifier ">"
BooleanType ::=BOOLEAN
BooleanValue::= TRUE | FALSE
XMLBooleanValue ::=
        "<" & "true" "/>"  |
        "<" & "false" "/>"
IntegerType ::=
                INTEGER  |
                INTEGER "{" NamedNumberList "}"
NamedNumberList ::=
                NamedNumber  |
                NamedNumberList "," NamedNumber
NamedNumber ::=
                identifier "(" SignedNumber ")"  |
                identifier "(" DefinedValue ")"
SignedNumber ::= number | "-" number
IntegerValue ::= SignedNumber | identifier
XMLIntegerValue ::=
                SignedNumber |
                "<" & identifier "/>"
```

```
EnumeratedType ::=
                ENUMERATED "{" Enumerations "}"
Enumerations ::= RootEnumeration  |
                RootEnumeration "," "..." ExceptionSpec  |
                RootEnumeration "," "..." ExceptionSpec "," AdditionalEnumeration
RootEnumeration ::= Enumeration
AdditionalEnumeration ::= Enumeration
Enumeration ::= EnumerationItem | EnumerationItem "," Enumeration
EnumerationItem ::= identifier | NamedNumber
EnumeratedValue ::= identifier
XMLEnumeratedValue ::= "<" & identifier "/>"
RealType ::= REAL
RealValue ::=
        NumericRealValue | SpecialRealValue
NumericRealValue ::=
        Realnumber  |
        "-" realnumber  |
        SequenceValue  --相关序列类型的值
SpecialRealValue ::=
         PLUS-INFINITY | MINUS-INFINITY
XMLRealValue ::=
         XMLNumericRealValue | XMLSpecialRealValue
XMLNumericRealValue ::=
         Realnumber  |
         "-" realnumber
XMLSpecialRealValue ::=
         "<" & PLUS-INFINITY "/>" | "<" & MINUS-INFINITY "/>"
BitStringType ::=  BIT STRING | BIT STRING "{" NamedBitList "}"
NamedBitList::=  NamedBit | NamedBitList "," NamedBit
NamedBit ::=  identifier "(" number ")" |
         identifier "(" DefinedValue ")"
BitStringValue ::= bstring | hstring | "{" IdentifierList "}" | "{" "}" | CONTAINING Value
IdentifierList ::=  identifier | IdentifierList "," identifier
XMLBitStringValue ::=
         XMLTypedValue  |
         Xmlbstring  |
         XMLIdentifierList  |
         empty
XMLIdentifierList ::=
         "<" & identifier "/>" |
         XMLIdentifierList "<" & identifier "/>"
OctetStringType ::= OCTET STRING
OctetStringValue ::= bstring | hstring | CONTAINING Value
```

```
XMLOctetStringValue ::=
        XMLTypedValue   |
        xmlhstring
NullType ::=  NULL
NullValue ::=  NULL
XMLNullValue ::= empty
SequenceType ::=
        SEQUENCE "{" "}"   |
        SEQUENCE "{" ExtensionAndException OptionalExtensionMarker "}"   |
        SEQUENCE "{" ComponentTypeLists "}"
ExtensionAndException ::= "..." | "..." ExceptionSpec
OptionalExtensionMarker ::= "," "..." | empty
ComponentTypeLists ::=
        RootComponentTypeList   |
        RootComponentTypeList "," ExtensionAndException ExtensionAdditions
            OptionalExtensionMarker   |
        RootComponentTypeList "," ExtensionAndException ExtensionAdditions
            ExtensionEndMarker "," RootComponentTypeList   |
        ExtensionAndException ExtensionAdditions ExensionEndMarker ","
            RootComponentTypeList   |
        ExtensionAndException ExtensionAdditions OptionalExtensionMarker
RootComponentTypeList ::= ComponentTypeList
ExtensionEndMarker ::= "," "..."
ExtensionAdditions ::= "," ExtensionAdditionList | empty
ExtensionAdditionList ::= ExtensionAddition   |
        ExtensionAdditionList "," ExtensionAddition
ExtensionAddition ::= ComponentType | ExtensionAdditionGroup
ExtensionAdditionGroup ::= "[[" VersionNumber ComponentTypeList "]]"
VersionNumber ::= empty | number ":"
ComponentTypeList ::= ComponentType   |
                        ComponentTypeList "," ComponentType
ComponentType ::=
        NamedType |
        NamedType OPTIONAL |
        NamedType DEFAULT Value |
        COMPONENTS OF Type
SequenceValue ::= "{" ComponentValueList "}" | "{" "}"
ComponentValueList ::= NamedValue |
                        ComponentValueList "," NamedValue
XMLSequenceValue ::=
        XMLComponentValueList   |
        empty
XMLComponentValueList ::=
```

```
        XMLNamedValue   |
        XMLComponentValueList XMLNamedValue
SequenceOfType ::= SEQUENCE OF Type | SEQUENCE OF NamedType
SequenceOfValue ::= "{" ValueList "}" | "{" NamedValueList "}" | "{" "}"
ValueList ::= Value | ValueList "," Value
XMLSequenceOfValue ::=
        XMLValueList   |
        XMLDelimitedItemList   |
        XMLSpaceSeparatedList   |
        empty
XMLValueList ::=
        XMLValueOrEmpty   |
        XMLValueOrEmpty XMLValueList
XMLValueOrEmpty ::=
        XMLValue   |
        "<" & NonParameterizedTypeName "/>"
XMLSpaceSeparatedList ::=
        XMLValueOrEmpty   |
        XMLValueOrEmpty " " XMLSpaceSeparatedList
XMLDelimitedItemList ::=
        XMLDelimitedItem   |
        XMLDelimitedItem XMLDelimitedItemList
XMLDelimitedItem ::=
        "<" & NonParameterizedTypeName ">" XMLValue
            "</" & NonParameterizedTypeName ">"   |
        "<" & identifier ">" XMLValue "</" & identifier ">"
SetType ::= SET "{" "}"   |
                SET "{" ExtensionAndException OptionalExtensionMarker "}" |
                SET "{" ComponentTypeLists "}"
SetValue ::= "{" ComponentValueList "}" | "{" "}"
XMLSetValue ::= XMLComponentValueList | empty
SetOfType ::= SET OF Type | SEQUENCE OF NamedType
SetOfValue ::= "{" ValueList "}" | "{" NamedValueList "}" | "{" "}"
XMLSetOfValue ::=
        XMLValueList           |
        XMLDelimitedItemList   |
        XMLSpaceSeparatedList   |
        empty
ChoiceType ::= CHOICE "{" AlternativeTypeLists "}"
AlternativeTypeLists ::=
        RootAlternativeTypeList   |
        RootAlternativeTypeList ","
            ExtensionAndException ExtensionAdditionAlternatives
```

```
                OptionalExtensionMarker
RootAlternativeTypeList ::= AlternativeTypeList
ExtensionAdditionAlternatives ::= "," ExtensionAdditionAlternativesList | empty
ExtensionAdditionAlternativesList ::= ExtensionAdditionAlternative  |
        ExtensionAdditionAlternativesList "," ExtensionAdditionAlternative
ExtensionAdditionAlternative ::= ExtensionAdditionAlternativesGroup | NamedType
ExtensionAdditionAlternativesGroup ::= "[[" VersionNumber AlternativeTypeList "]]"
AlternativeTypeList ::= NamedType  |
                                AlternativeTypeList "," NamedType
ChoiceValue ::= identifier ":" Value
XMLChoiceValue ::= "<" & identifier ">" XMLValue "</" & identifier ">"
SelectionType ::= identifier "<" Type
TaggedType ::= Tag Type  |
                Tag IMPLICIT Type  |
                Tag EXPLICIT Type
Tag ::=  "[" Class ClassNumber "]"
ClassNumber ::= number | DefinedValue
Class ::= UNIVERSAL     |
          APPLICATION  |
          PRIVATE      |
          empty
TaggedValue ::= Value
XMLTaggedValue ::= XMLValue
EmbeddedPDVType ::=  EMBEDDED PDV
EmbeddedPDVValue ::=  SequenceValue
XMLEmbeddedPDVValue ::= XMLSequenceValue
ExternalType ::= EXTERNAL
ExternalValue ::= SequenceValue
XMLExternalValue ::= XMLSequenceValue
ObjectIdentifierType ::=  OBJECT IDENTIFIER
ObjectIdentifierValue ::= "{" ObjIdComponentsList "}"  |
                          "{" DefinedValue ObjIdComponentsList "}"
ObjIdComponentsList ::= ObjIdComponents  |
                        ObjIdComponents ObjIdComponentsList
ObjIdComponents ::= NameForm            |
                    NumberForm          |
                    NameAndNumberForm   |
                    DefinedValue
NameForm ::= identifier
NumberForm ::= number | DefinedValue
NameAndNumberForm ::= identifier "(" NumberForm ")"
XMLObjectIdentifierValue ::=
        XMLObjIdComponentList
```

```
XMLObjIdComponentList ::=
        XMLObjIdComponent   |
        XMLObjIdComponent & "." & XMLObjIdComponentList
XMLObjIdComponent ::=
        NameForm            |
        XMLNumberForm       |
        XMLNameAndNumberForm
XMLNumberForm ::= number
XMLNameAndNumberForm ::=
        identifier & "(" & XMLNumberForm & ")"
RelativeOIDType ::=
        RELATIVE-OID
RelativeOIDValue ::=
        "{" RelativeOIDComponentsList "}"
RelativeOIDComponentsList ::=
        RelativeOIDComponents   |
        RelativeOIDComponents RelativeOIDComponentsList
RelativeOIDComponents ::= NumberForm   |
        NameAndNumberForm   |
        DefinedValue
XMLRelativeOIDValue ::=
        XMLRelativeOIDComponentList
XMLRelativeOIDComponentList ::=
        XMLRelativeOIDComponent   |
        XMLRelativeOIDComponent & "." & XMLRelativeOIDComponentList
XMLRelativeOIDComponent ::=
        XMLNumberForm   |
        XMLNameAndNumberForm
CharacterStringType ::= RestrictedCharacterStringType | UnrestrictedCharacterStringType
RestrictedCharacterStringType ::=
        BMPString
        GeneralString   |
        GraphicString   |
        IA5String       |
        ISO646String
        NumericString   |
        PrintableString |
        TeletexString   |
        T61String       |
        UniversalString
        UTF8String      |
        VideotexString  |
        VisibleString
```

```
RestrictedCharacterStringValue ::= cstring | CharacterStringList | Quadruple | Tuple
CharacterStringList ::= "{" CharSyms "}"
CharSyms ::= CharsDefn | CharSyms "," CharsDefn
CharsDefn ::= cstring | Quadruple  Tuple | DefinedValue
Quadruple ::= "{" Group "," Plane "," Row "," Cell "}"
Group ::= number
Plane ::= number
Row ::= number
Cell ::= number
Tuple ::= "{" TableColumn "," TableRow "}"
TableColumn ::= number
TableRow ::= number
XMLRestrictedCharacterStringValue ::= xmlcstring
UnrestrictedCharacterStringType ::= CHARACTER STRING
CharacterStringValue ::= RestrictedCharacterStringValue | UnrestrictedCharacterStringValue
XMLCharacterStringValue ::=
        XMLRestrictedCharacterStringValue  |
        XMLUnrestrictedCharacterStringValue
UnrestrictedCharacterStringValue ::= SequenceValue
XMLUnrestrictedCharacterStringValue ::= XMLSequenceValue
UsefulType ::= typereference
```

在 37.1 中定义的下列字符串类型：

```
NumericString     VisibleString
PrintableString   ISO646String
TeletexString     IA5String
T61String         GraphicString
VideotexString    GeneralString
UniversalString   BMPString
```

在第 42 章到第 44 章中定义的下列实用类型：

```
GeneralizedTime
UTCTime
ObjectDescriptor
```

在第 45 章到第 47 章中使用的下列产生式：

```
ConstrainedType ::=
        Type Constraint |
        TypeWithConstraint
TypeWithConstraint ::=
        SET Constraint OF Type          |
        SET SizeConstraint OF Type      |
        SEQUENCE Constraint OF Type     |
        SEQUENCE SizeConstraint OF Type |
        SET Constraint OF NamedType     |
        SET SizeConstraint OF NamedType |
```

```
        SEQUENCE Constraint OF NamedType |
        SEQUENCE SizeConstraint OF NamedType
Constraint ::= "(" ConstraintSpec ExceptionSpec ")"
ConstraintSpec ::= SubtypeConstraint |
                   GeneralConstraint
ExceptionSpec ::= "!" ExceptionIdentification | empty
ExceptionIdentification ::= SignedNumber |
        DefinedValue |
        Type ":" Value
SubtypeConstraint ::= ElementSetSpecs
ElementSetSpecs ::=
        RootElementSetSpec |
        RootElementSetSpec "," "..." |
        RootElementSetSpec "," "..." "," AdditionalElementSetSpec
RootElementSetSpec ::= ElementSetSpec
AdditionalElementSetSpec ::= ElementSetSpec
ElementSetSpec ::= Unions | ALL Exclusions
Unions ::= Intersections |
        UElems UnionMark Intersections
UElems ::= Unions
Intersections ::= IntersectionElements |
        IElems IntersectionMark IntersectionElements
IElems ::= Intersections
IntersectionElements ::= Elements | Elems Exclusions
Elems ::= Elements
Exclusions ::= EXCEPT Elements
UnionMark ::= "|" | UNION
IntersectionMark ::= "^" | INTERSECTION
Elements ::=
        SubtypeElements |
        ObjectSetElements |
        "(" ElementSetSpec ")"
SubtypeElements ::=
        SingleValue |
        ContainedSubtype |
        ValueRange |
        PermittedAlphabet |
        SizeConstraint |
        TypeConstraint |
        InnerTypeConstraints |
        PatternConstraint
SingleValue ::= Value
ContainedSubtype ::= Includes Type
```

```
Includes ::= INCLUDES | empty
ValueRange ::= LowerEndpoint ".." UpperEndpoint
LowerEndpoint ::= LowerEndValue | LowerEndValue "<"
UpperEndpoint ::= UpperEndValue | "<" UpperEndValue
LowerEndValue ::= Value | MIN
UpperEndValue ::= Value | MAX
SizeConstraint ::= SIZE Constraint
PermittedAlphabet ::= FROM Constraint
TypeConstraint ::= Type
InnerTypeConstraints ::=
        WITH COMPONENT SingleTypeConstraint |
        WITH COMPONENTS MultipleTypeConstraints
SingleTypeConstraint::= Constraint
MultipleTypeConstraints ::= FullSpecification | PartialSpecification
FullSpecification ::= "{" TypeConstraints "}"
PartialSpecification ::= "{" "..." "," TypeConstraints "}"
TypeConstraints ::=
        NamedConstraint |
        NamedConstraint "," TypeConstraints
NamedConstraint ::=
        identifier ComponentConstraint
ComponentConstraint ::= ValueConstraint PresenceConstraint
ValueConstraint ::= Constraint | empty
PresenceConstraint ::= PRESENT | ABSENT | OPTIONAL | empty
PatternConstraint ::= PATTERN Value
```