ICS 35.100.60
L 79

中华人民共和国国家标准

GB/T 16262.4—2006/ISO/IEC 8824-4:2002

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

Information technology—Abstract Syntax Notation One(ASN.1)—Part 4: Parameterization of ASN.1 specifications

(ISO/IEC 8824-4:2002,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

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4部分，等同采用国际标准ISO/IEC 8824-4:2002《信息技术　抽象语法记法一(ASN.1)：ASN.1规范的参数化》(英文版)。与该项国际标准的等同文本是ITU-T建议X.683。

按照GB/T 1.1—2000的规定，本部分对ISO/IEC 8824-4:2002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。

本部分的附录A和附录B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、安金海、徐云驰。

引　言

应用设计者需编写某些方面留待定义的规范。这些方面随后将由一个或几个小组予以定义(每一个用自己的方式),以产生一个供抽象语法定义用的全面定义的规范(每个小组一个)。

在某些情况,规范的若干方面(例如,界限)可以留待定义,甚至留待抽象语法定义时,而由国际标准化轮廓或某个其他团体功能轮廓规范来完成。

注 1:鉴于本部分的需求,在定义抽象语法之前,并不是要单独实现约束应用的任何方面。

在极端情况,规范的某些方面留待实施者完成,并按照协议实现一致性声明予以规定。

GB/T 16262.2 和 GB/T 16262.3 的规定提供稍后完成规范某些部分的框架,它们本身不能解决上面的要求。

另外,一个设计者有时需要定义许多类型,或许多信息客体类别,或许多信息客体集合,或许多信息客体,或许多值,它们具有相同的外部层次结构,但是内部层次上使用的类型,或信息客体类别,或信息客体集合,信息客体,或值是不同的。不写出每一个情况的外部层次结构,而只写一个情况,留待今后定义的部分引用它并提供附加信息的做法是有用的。

本部分的参数化引用名和参数化赋值的条款就是针对所有这些需求的。

参数化引用名的语法形式与相应的正式引用名相同,但应顾及下列附加考虑:

——当在参数化赋值语句赋值时,其后是括号中的虚设引用名表,每个可能伴随一个支配者;这些引用名具有赋值语句右边的范围和参数表。

注 2:这正是识别出参数化引用名的根据。

——当被出口或入口时,随后是一对空括号以便将它标识为参数化引用名。

——当它用于任一构造时,随后是语法结构表,每个虚设引用名一个,它提供仅对此用途的虚设引用名的赋值。

虚设引用名具有与相应正式引用名相同的语法形式,并可用于赋值语句右边使用相应正式引用名的任何地方。要求所有这些用法一致。

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

1　范围

GB/T 16262的本部分是抽象语法记法一(ASN.1)的一个部分,并定义了ASN.1规范的参数化的记法。

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 16262.1—2006　信息技术　抽象语法记法一(ASN.1)　第1部分:基本记法规范(ISO/IEC 8824-1:2002,IDT)

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)

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　基本记法规范

本部分使用GB/T 16262.1—2006中定义的术语。

3.2　信息客体规范

本部分使用GB/T 16262.2—2006中定义的术语。

3.3　约束规范

本部分使用GB/T 16262.3—2006中定义的术语。

3.4　附加定义

3.4.1

标准引用名称　normal reference name

由"Assignment"方法而不是"ParameterizedAssignment"定义的、无参数的引用名称。这种名称引用完整的定义,并且在使用时不向其提供实际的参数。

3.4.2

参数化引用名称　parameterized reference name

使用参数化赋值定义的引用名称,它引用不完整的定义,因此在使用时必须提供实际的参数。

3.4.3

参数化类型　parameterized type

使用参数化类型赋值定义的类型,其成分是不完整的定义,在使用这种类型时必须提供实际的参数。

3.4.4

参数化值 parameterized value

使用参数化值赋值定义的值，其值没有完整地予以规定，在使用时必须向其提供实际的参数。

3.4.5

参数化值集合 parameterized value set

使用参数化值集合赋值定义的值，其值没有完整地予以规定，在使用时必须向其提供实际的参数。

3.4.6

参数化客体类别 parameterized object class

使用参数化客体类别赋值定义的信息客体类别，其字段规范没有完整地予以规定，在使用时必须向其提供实际的参数。

3.4.7

参数化客体 parameterized object

使用参数化客体赋值定义的信息客体，其成分没有完整地予以规定，在使用时必须向其提供实际的参数。

3.4.8

参数化客体集合 parameterized object set

使用参数化客体集合赋值定义的信息客体集合，其客体没有完整地予以规定，在使用时必须向其提供实际的参数。

3.4.9

可变约束 variable constraint

规定参数化抽象语法时采用的约束，此约束视抽象语法的某个参数而定。

4 缩略语

本部分使用下列缩略语：

ASN.1 抽象语法记法一

5 约定

本部分采用 GB/T 16262.1—2006 的第 5 章定义的记法约定。

6 记法

本章综述本部分定义的记法。

6.1 赋值

本部分定义了下列记法，此记法能用来替代“Assignment”(见 GB/T 16262.1—2006 的第 12 章)：

——ParameterizedAssignment(见 8.1)。

6.2 参数化定义

6.2.1 本部分定义了下列记法，此记法能用来替代“DefinedType”(见 GB/T 16262.1—2006 的 13.1)：

——ParameterizedType(见 9.2)。

6.2.2 本部分定义了下列记法，此记法能用来替代“DefinedValue”(见 GB/T 16262.1—2006 的 13.1)：

——ParameterizedValue(见 9.2)。

6.2.3 本部分定义了下列记法，此记法能用来替代“DefinedType”(见 GB/T 16262.1—2006 的 13.1)：

——ParameterizedValueSetType(见 9.2)。

6.2.4 本部分定义了下列记法，此记法能用来替代“ObjectClass”(见 GB/T 16262.2—2006 的 9.2)：

——ParameterizedObjectClass(见 9.2)。

6.2.5 本部分定义了下列记法，此记法能用来替代“Object”(见 GB/T 16262.2—2006 的 11.3)：

——ParameterizedObject (见 9.2)。

6.2.6 本部分定义了下列记法，此记法能用来替代“ObjectSet”(见 GB/T 16262.2—2006 的 12.3)：

——ParameterizedObjectSet(见 9.2)。

7 ASN.1 词项

本部分使用 GB/T 16262.1—2006 的第 11 章规定的词项。

8 参数化赋值

8.1 与 GB/T 16262.1 和 GB/T 16262.2 规定的每一赋值语句有对应的参数化赋值语句。“ParameterizedAssignment”结构是：

```
ParameterizedAssignment ::=
        ParameterizedTypeAssignment
        | ParameterizedValueAssignment
        | ParameterizedValueSetTypeAssignment
        | ParameterizedObjectClassAssignment
        | ParameterizedObjectAssignment
        | ParameterizedObjectSetAssignment
```

8.2 除了在初始词项之后有一个“ParameterList”之外，每个“Parameterized <X> Assignment”的语法与“<X> Assignment”相同。因此初始项成了参数的引用名(见 3.4.2)：

注：GB/T 16262.1 要求在一个模块中所赋予的所有引用名，不管是否参数化，必须是明确的：

```
ParameterizedTypeAssignment ::=
        typereference
        ParameterList
        "::="
        Type
ParameterizedValueAssignment ::=
        valuereference
        ParameterList
        Type
        "::="
        Value
ParameterizedValueSetTypeAssignment ::=
        typereference
        ParameterList
        Type
        "::="
        Value Set
ParameterizedObjectClassAssignment ::=
        objectclassreference
        ParameterList
        "::="
        Object Class
```

ParameterizedObjectAssignment::=
objectreference
ParameterList
DefinedObjectClass
" ::= "
Object
ParameterizedObjectSetAssignment::=
objectreference
ParameterList
DefinedObjectClass
" ::= "
ObjectSet

8.3 "ParameterList"是括号之间的"Parameter"表

ParameterList::= " { " Parameter " ," " " + " " } "

每个"Parameter"由"DummyReference"和或许还有"ParamGovernor"组成：

Parameter::=ParamGovernor " :" DummyReference| DummyReference
ParamGovernor::= Governor| DummyGovernor
Governor::=Type| DefinedObjectClass
DummyGovernor::= DummyReference
DummyReference::= Reference

"Parameter"中的"DummyReference"可以代表：

a) 在没有"ParamGovernor"的情况，代表"Type"或"DefinedObjectClass"；
b) 在应存在"ParamGovernor"的情况，代表"Value"或"ValueSet"；在"ParamGovernor"是"Governor"的情况，它应是"Type"；在"ParamGovernor"是"DummyGovernor"的情况，"ParamGovernor"的实际参数应是"Type"；
c) 在应存在"ParamGovernor"的情况，代表"Object"或"ObjectSet"；在"ParamGovernor"是"Governor"的情况，它应是"DefinedObjectClass"；在"ParamGovernor"是"DummyGovernor"的情况，"ParamGovernor"的实际参数应是"DefinedObjectClass"；

"DummyGovernor"应是没有"Governor"的"DummyReference"。

8.4 出现在"ParameterList"中的"DummyReference"的范围是"ParameterList"本身，以及在"::="之后的"ParameterizedAssignment"部分。"DummyReference"潜藏此范围中具有相同名称的其他"Reference"。

注：本条不适用于"NamedNumberList"、"Enumeration"和"NamedBitList"中定义的"identifier"，因为它们不是"Reference"。"DummyReference"不潜藏这些"identifier"(见 GB/T 16262.1—2006 的 18.11 和 19.10)。

8.5 "DummyReference"在其范围内的用法应与其语法形式、何处应用、支配者一致，相同"DummyReference"的所有用法应彼此一致。

注：在虚设引用名称的语法形式有二义性的时候(例如，是"objectclassreference"还是"typereference")，这种二义性通常可根据赋值语句右边首先使用虚设引用名称加以解决。据此，虚设引用名称的特性是已知的。然而当它依次仅用作参数化引用的实际参数时，只用赋值语句的右侧不能确定虚设引用名称的特性；在此情况中，虚设引用名称的性质必须通过检查此参数化引用的定义予以确定。需告诉此记法的使用者，这一实际做法会使 ASN.1 规范不太清晰，建议提供足够的注释以向读者说明。

示例：

考虑有下列参数化客体类别赋值：

PARAMETERIZED-OBJECT-CLASS { TypeParam, INTEGER: ValueParam, INTEGER: ValueSetParam}::=

```
CLASS {
&valueField1          TypeParam,
&valueField2          INTEGER DEFAULT valueParam,
&valueField3          INTEGER (ValueSetParam)
&ValueSetField    INTEGER DEFAULT{ValueSetParam},
}
```

为确定"ParameterizedAssignment"范围中的"DummyReference"的合适用法，且目的仅限于此，可考虑将"DummyReference"定义如下：

TypeParam::=UnspecifiedType

valueParam INTEGER::= unspecifiedIntegerValue

ValueSetParam INTEGER::= {UnspecifiedIntegerValueSet}

其中：

a) TypeParam 是代表"Type"的"DummyReference"。因此，凡是将"typereference"用作，例如固定类型值字段 valueField1 的"Type"时，可以使用 TypeParam；

b) ValueParam 是代表整数类型的值的"DummyReference"。因此，凡是能将整数值的"valuereference"用作，例如，固定类型值字段 valueField2 的默认值时，可以使用 valueParam；

C) ValueSetParam 是代表整数类型的值集合的"DummyReference"。因此，凡是能将整数值的"typereference"用作，例如，valueField3 和 ValueSetField 的"ConstrainedSubtype"记法中的"Type"时，可以使用 ValueSetParam。

8.6 在其范围内每个"DummyReference"应至少采用一次。

注：如果"Dummy Reference"没有这样出现过，对应的"Actual Parameter"不会影响此定义，好像被排除了，而对使用者而言，它们产生过某个规范。

包含对自身的直接或间接引用的"ParameterizedValueAssignment"、"ParameterizedValueSetAssignment"、"ParameterizedObjectAssignment"、"ParameterizedObjectSetAssignment"是不合法的。

8.7 在"ParameterizedType"、"ParameterizedValueSet"或"ParameterizedObjectClass"的定义中，不应将"DummyReference"作为已标记类型(作为实际参数)递归引用传递给"ParameterizedType"、"ParameterizedValueSet"或"ParameterizedObjectClass"(见 A.3)。

8.8 在"ParameterizedType"、"ParameterizedValueSet"或"ParameterizedObjectClass"的定义中，不应产生对被定义项的循环引用，除非这种引用直接或间接标记成 OPTIONAL 或在通过对选择类型的引用产生的"ParameterizedType"和"ParameterizedValueSet"情况，至少其替代记法之一在定义中是非循环的。

8.9 "DummyReference"的支配者不应包括对另一个同样具有支配者的"DummyReference"的引用。

8.10 在参数化赋值中"::="的右边不应仅由"DummyReference"组成。

8.11 "DummyReference"的支配者不应要求知道此"DummyReference"，或正被定义的参数化引用名。

8.12 当值或值集合作为实际参数提供给参数化类型时，要求此实际参数的类型与对应虚设参数的支配者兼容(详见 GB/T 16262.1—2006 的 B.6.2 和 B.6.3)。

8.13 在定义具有值或值集合虚设参数的参数化类型时，用来支配该虚设参数的这一类型应是其所有值用于赋值右边使用虚设参数的各处都有有效的类型(详见 GB/T 16262.1—2006 的 B.6.5)。

9 引用参数化的定义

9.1 在"SymbolList"(在"Exports"或"Imports"中)，参数化定义应通过"Parameterized Reference"予

以引用：

ParameterizedReference::= Reference| Reference"{" "}"

其中，如上述 8.2 规定，“Reference”是“ParameterizedAssignment”中的第一个词项。

注：提供“Parameterized Reference”第一个替代记法只是帮助人们理解。两种替代记法具有相同意义。

9.2 在“Exports”或“Imports”以外，参数化定义应通过“Parameterized<X>”结构予以引用，此结构可用作对应“ <X>”的替代记法：

```
ParameterizedType::=
        SimpleDefinedType
        ActualParameterList
SimpleDefinedType::=
        ExternalTypeReference|
        typereference
ParameterizedValue::=
        SimpleDefinedValue
        ActualParameterList
SimpleDefinedValue::=
        ExternalValueReference|
        valuereference
ParameterizedValueSetType::=
        SimpleDefinedType
        ActualParameterList
ParameterizedObjectClass::=
        DefinedObjectClass
        ActualParameterList
ParameterizedObjectSet::=
        DefinedObjectSet
        ActualParameterList
ParameterizedObject ::=
        DefinedObject
        ActualParameterList
```

9.3 “Defined<X>”中的引用名称应是“Parameterized Assignment”中对其产生赋值的引用名称。

9.4 关于使用“Defined<X>”替代记法的限制（在 GB/T 16262.1 和 GB/T 16262.2 中规定为标称）同样适用于相应的参数化引用名称。

注：大体上，限制如下：每个“Defined<X>”有两种替代记法，即“<X>reference”和“External<X>Reference”。前者在定义模块中使用，或如果已输入定义并且没有名称冲突时使用；后者在没有所列的输入（未赞同）或如果被输入的名称与本地定义之间（也未赞同）或在输入之间有冲突时使用。

9.5 “Actual Parameter List”是：

```
Actual Parameter List::=
        " { " Actual Parameter " ," + " } "
Actual Parameter ::=
        Type                    |
        Value                   |
        ValueSet                |
```

```
        DefinedObjectClass      |
        Object                  |
        ObjectSet
```

9.6 对应"ParameterizedAssignment"中的每个"Parameter"应正好有个"ActualParameter",并且以同样的顺序出现。对"ActualParameter"和支配者(如果有)的具体选择应通过此"Parameter"的语法形式和它出现在"ParameterizedAssignment"的环境的检查予以确定。"ActualParameter"的形式应是取代其范围内每处的"DummyReference"所要求的形式(见 8.4)。

示例:

例如,可以引用前面示例(见 8.5)的参数化客体类别定义如下:

MY—OBJECT—CLASS::=PARAMETERIZED-OBJECT-CLASS{BIT STRING,123,{4|5|6}}

9.7 在确定正由使用参数化引用名的实例引用的实际类型、值、值集合、客体类别或客体集合时,实际参数代替虚设引用名称。

9.8 出现在"Actual Parameter"中任一引用的含义和默认适用于也是如此出现的任何标记的标记,按照"Actual Parameter"的标记环境而不是对应的"Dummy Reference"的标记环境予以确定。

注:因此,参数化,尤其如引用、选择类型和 COMPONENT OF 的参数化,准确地说不是按照文字的替换。

示例:

假设有下列模块:

```
M1  DEFINITIONS AUTOMATIC TAGS::=BEGEN
      EXPORTS T1;
    T1::=SET{
      f1   INTEGER,
      f2   BOOLEAN
    }
END
M2  DEFINITIONS EXPLICIT  TAGS::= BEGIN
      IMPORTS T1 FROM M1;
    T3::=T2{T1}
    T2{X}::=SEQUENCE{
      a   INTEGER,
      b   X
    }
END
```

应用 9.8 意味着 T3 的成分 f1 的标记(即@T3.b.f1)将隐含地予以标记,因为虚设参数 X(名义上明显标记)的标记环境不影响实际参数 T1 的成分的标记。

假设有模块 M3:

```
M3  DEFINITIONS AUTOMATIC TAGS::= BEGIN
      IMPORTS T1 FROM M1;
    T5::=T4{T1}
    T4{Y}::=SEQUENCE{
      a   INTEGER,
      b   Y
    }
END
```

应用 GB/T 16262.1—2006 的 30.6 意味着 T5 的成分 b 的标记(即@T5.B)将明显地予以标记,因为虚设参数 Y 总是明显地予以标记,因此 T5 等效于:

```
T5::=SEQUENCE{
        a   [0]IMPLICIT   INTEGER,
        b   [1]EXPLICIT   SET{
        f1  [0]INTEGER,
        f2  [1]BOOLEAN
        }
    }
```

而 T3 等效于:

```
T3::=SEQUENCE{
        a     INTEGER,
        b     SET{
        f1    [0]IMPLICIT   INTEGER,
        f2    [1]IMPLICIT   BOOLEAN
        }
}
```

10 抽象语法参数

10.1 GB/T 16262.2—2006 的附录 B 提供了 ABSTRACT—SYNTAX 信息客体类别,并建议用其定义抽象语法。一个示例是将抽象语法定义成一个 ASN.1 类型的值集合,而这一 ASN.1 类型的外层未被参数化。

10.2 用来定义抽象语法的 ASN.1 类型予以参数化时,某些参数可作为定义抽象语法时的实际参数予以提供,而一些其他参数留作抽象语法本身的参数。

示例:

如果某一参数化类型已定义,叫做有二个虚设引用(假设第一个是某个已定义客体类别的客体集合,第二个是在一个边界内的整数值)的 YYY—PDU,那么:

```
YYY—Abstract—Syntax{INTEGER:bound}ABSTRACT—Syntax::=
    {YYY—PDU{{ValidObjects},bound}IDENTIFIED BY{YYY 5}}
```

定义其中已解决客体集合的参数化抽象语法,而 bound 留作抽象语法的一个参数。

抽象语法参数的使用方法是:

a) 直接或间接用于约束的上下文中;

b) 直接或间接用作最后用于约束上下文中的实际参数。

注:见 A.2 中的示例和 GB/T 16262.1—2006 的 F.5 中的示例。

10.3 其值集合视抽象语法一个或几个参数而定的约束是可变约束。在定义抽象语法之后确定这种约束(或许由国际标准化轮廓或在协议实现一致性声明中确定)。

注:如果在约束值规范包含的定义中的某处出现了抽象语法的参数,那么,此限别是可变约束。即使最后的约束的值集合与抽象语法的参数的实际值无关,它也是可变约束。

示例:

((1..3)EXCEPT a)UNION(1..3)的值总是 1..3,与 a 是什么值无关,若 a 是抽象语法的一个参数,它仍是可变约束。

10.4 形式上,可变约束不约束抽象语法中的值集合。

注:强力推荐希望留作抽象语法中可变约束的约束要有使用 GB/T 16262.1—2006 的 49.4 提供的记法的特殊规范。

附 录 A
（资料性附录）
示 例

A.1 使用参数化类型定义的示例

假设一名协议设计者常常需要携带具有一个或多个协议字段的鉴别符。这将按 BIT STRING 携带(位于此字段一侧)。没有参数化，Authenticator 需定义为 BIT STRING，因此不管 authenticator 是否要出现，需将它附加到文本以标识它应用于什么。换言之，此设计者要遵守将具有鉴别符的字段能换成此字段和 authenticator 的 SEQENCE 的约束。参数化机制为此工作提供便捷的方法。

首先，我们定义参数化类型 SIGNED{}：

```
SIGNED{ToBeSigned}::=SEQUENCE
    {
        authenticated-data      ToBeSigned,
        authenticator           BIT  STRING
    }
```

因此，在协议的主体中，此记法(作为示例)

```
SIGNED{OrderInFormation}
```

是下列的类型记法：

```
SEQUENCE
    {
        authenticated-data      OrdeInFormation,
        authenticator           BIT  STRING
    }
```

进一步假设发送者必须作出对某些字段是否加鉴别符的选择。这可通过产生任选的 BIT STRING 完成，但是更好的方案(线上比特更少)是定义另一种参数化类型。

```
OPTIONALLY—SIGNED{ToBeSigned}::=CHOICE
    {
        unsigned—data   [0]   ToBeSigned,
        signed—data     [1]   SIGNED{ ToBeSigned}
    }
```

注：如果编制者保证在使用参数化类型中没有一次产生 BIT STRING 的实际自变量(SIGNED 的类型)，则 CHOICE 中的标记不是必需的，但是在防止规范其他部分中的差错是有用的。

A.2 使用参数化定义以及信息客体类别的示例

使用信息客体类别以集中抽象语法的所有参数。用此方法能使抽象语法的参数数目减少到收集类别的一个实例。可以使用“InformationFromObject”生成式提取参数客体中的信息。

示例：

一这种类别的一个实例包含抽象语法 Message—PDU 的所有参数。

```
MESSAGE—PARAMETERS::=CLASS{
        &maximum—priority—level              INTEGER,
        &maximum—message—buffer—size         INTEGER,
```

```
                &maximum—reference—buffer—size      INTEGER
        }
        WITH SYNTAX{
                THE MAXIMUM PRIORITY LEVEL IS &maximum-priority-level
                THE MAXIMUM MESSAGE BUFFER SIZE IS &maximum-message-buffer-size
                THE MAXIMUM REFERENCE BUFFER SIZE IS &maximum-reference-buffer-size
        }
        --"Value From Object"生成式用来提供抽象语法参数"Param"
        --中的值。这些值只能用于约束中。
        --此外,此参数能一直被传送给另一个参数化类型。
    Message—PDU{MESSAGE—PARAMETERS :param}::=SEQUENCE{
        priority—level      INTEGER(0.. param. &maximum—priority—level),
        message             BMPString(SIZE(0.. param. &maximum-message-buffer-size)),
        reference           Reference{param}
        }
    Reference{ MESSAGE—PARAMETERS :param}::=
            SEQUENCE OF IA5String(SIZE(0..param. &maximum-reference-buffer-size))
        --参数化抽象语法信息客体的定义。
        --此抽象语法参数只能用于约束。
    Message—Abstract—Syntax{MESSAGE—PARAMETERS:param}
        ABSTRACT—SYNTAX::=
        {
            Message—PDU{param}
            IDENTIFIED  BY{joint—is o—ccitt  asn1(1) examples(123)0}
        }
```

类别 MESSAGE—PARAMETERS 和参数化抽象语法客体 message—Abstract—Syntax 使用如下:

```
    —MASSAGE—PARAMETERS 的实例定义此抽象语法的参数值。
    My—message—parameters MESSAGE—PARAMETERS::={
            THE MAXIMUM PRIORITY LEVEL IS 10
            THE MAXIMUM MESSAGE BUFFER SIZE IS  2000
            THE MAXIMUM REFERENCE  BUFFER SIZE IS 100
        }
    --现在可用规定的可变约束定义此抽象语法。
    My—message—Abstract—Syntax ABSTRACT—SYNTAX::=
            message—Abstract—Syntax{ My—message—parameters}
```

A.3 有限的参数化类型定义的示例

当规定作为一个通用表形式的参数化类型时,规定此种类型会使产生的 ASN.1 记法是有限的方法。例如,我们规定:

```
    List1{ElementTypeParam}::=SEQUENCE{
```

```
        elemElementTypeParam,
        next List1{ ElementTypeParam}OPTIONAL
    }
```

为了使用下列形式时是有限的：

```
IntegerList1::= List1{INTEGER}
```

产生的 ASN.1 记法正如它通常定义的解释：

```
IntegerList1::=SEQUENCE{
        elem   INTEGER,
        next   IntegerList1 OPTIONAL
    }
```

将此与下列定义比较：

```
List2{ ElementTypeParam}::=SEQUENCE{
    elem     ElementTypeParam,
        next List2{[0] ElementTypeParam} OPTIONAL
    }
IntegerList2::= List2{INTEGER}
```

在此场合，产生的 ASN.1 记法是无限的：

```
IntegerList2::= SEQUENCE{
    elem   INTEGER,
    next   SEQUENCE{
        Elem   [0] INTEGER,
        next   SEQUENCE{
            elem   [0] [0] INTEGER,
            next       SEQUENCE{
                elem     [0] [0] [0] INTEGER,
                next       SEQUENCE{
            --等等
                } OPTIONAL
            } OPTIONAL
        } OPTIONAL
    } OPTIONAL
        }
```

A.4 参数化值定义的示例

如果定义参数化串值如下：

```
genericBirthdayGreeting{IA5String:name} IA5String::={ " Happy birthday,",
name," !!" }
```

那么下面二个串值是相同的：

```
greering1 IA5String::= genericBirthdayGreeting{ " John " }
    greering2 IA5String::= " Happy birthday,John!!"
```

A.5 参数化值集合定义的示例

如果定义二个参数化值集合如下：

```
QuestList1 { IA5String:extra Quest}IA5String::={ " Jack " | " John " | extraQuest}
    QuestList2{ IA5String:ExtraQuest} IA5String::={ " Jack " | " John " |
    extraQuest}
```

那么下列值集合表示此相同的值集合：

```
SetofQuest1 IA5String::={ QuestList1{ " Jill " }}
    SetofQuest2 IA5String::={ QuestList2{{  " Jill " }}}
    SetofQuest3 IA5String::={ " Jack " | " John " | " Jill " }
```

以及下列值集合表示此相同的值集合：

```
SetofQuest4 IA5String::={ QuestList2{{  " Jill " " Mary " }}}
    SetofQuest5 IA5String::={ " Jack " | " John " | " Jill " | " Mary " }
```

注意，一个值集合总是在花括号中规定的，即使它是一个参数化值集合引用时也是如此。通过在对曾在值集合赋值中创建的“identifier”的引用中略去花括号或在对“ParameterizedValueSetType”的引用中略去花括号，此记法则是“Type”的记法，而不是值集合记法。

A.6 参数化类别定义的示例

下列参数化类别可用来定义包含不同类型的差错代码的差错类别。注意，ErrorCodeType 参数只能用作 ValidErrorCodes 参数的“DummyGovernor”

```
GENERIC—ERROR{ErrorCodeType,ErrorCodeType:ValidErrorCodes}::=CLASS{
        &errorCode      ValidErrorCodes
    }
    WITH SYNTAX{
        CODE   &errorCode
    }
```

可以使用如下参数化类别定义来定义像同样的定义语法那样共享一些特性的不同类别：

```
ERROR—1::= GENERIC—ERROR{ INTEGER,{1|2|3}}
ERROR—2::= GENERIC—ERROR{ ErrorCodeString,{ StringErrorCode}}
ERROR—3::= GENERIC—ERROR{ EnumeratedErrorCode,{fatal |error}}
ErrorCodeString::=IA5String(SIZE(4))
StringErrorCodes ErrorCodeString::={ " E001 " | " E002 " | " E003 " }
EnumeratedErrorCode::=ENUMERATED{fatal,error,warning}
```

因此可使用定义类别如下：

```
MY—Errors ERROR—2::={{CODE " E001 " }|{CODE " E002 "}}
FatalError ERROR—3::={CODE   fatal}
```

A.7 参数化客体集合定义的示例

参数化客体集合定义 A11 Type 构成包含基本客体集合 BaseType 和以参数 additionalType 提供的附加客体集合的客体集合：

```
AllTypes{TYPE—IDENTIFIER:AdditionalTypes}TYPE—IDENTIFIER::=
{BaseTypes| AdditionalTypes}
BaseTypes TYPE—IDENTIFIER::={
            { BasicTypes-1 IDENTIFIFD  BY  basic-type-obj-id-value-1}|
            { BasicTypes-2 IDENTIFIFD  BY  basic-type-obj-id-value-2}|
            { BasicTypes-3 IDENTIFIFD  BY  basic-type-obj-id-value-3}
```

```
        }
```

参数化客体集合定义可使用如下：

```
    MY-ALL-Types TYPE-IDENTIFIER::={AllTypes{{
                    {MY-Type-1 IDENTIFIED BY my-obj-id-value-1}|
                    {MY-Type-2 IDENTIFIED BY my-obj-id-value-2}|
                    {MY-Type-3 IDENTIFIED BY my-obj-id-value-3}
        }}}
```

A.8 参数化客体集合定义的示例

在如下的参数化抽象语法定义中可使用 GB/T 16262.3—2006 的 A.4 定义类型：

```
    --Possible Body Types 是一种抽象语法的参数。
      Message-abstract-syntax { MHS-BODY-CLASS: PossibleBodyTypes } ABSTRACT-SYN-
TAX::={
                    INSTANCE OF MHS-BODY-CLASS({PossibleBodyTypes} )
                    IDENTIFIED BY{joint-iso-itu asn1(1)examples(1)123}
        }
    --此客体集合列出单纯实例类型所有可能成对的值和 type-ids。
    --此客体集合用作参数化抽象语法定义的实际参数。
    MY-BODY-TYPES MHS-BODY-CLASS::={
                    {MY-First-Type      IDENTIFIED BY my-first-obj-id}|
                    {MY-Second-Type     IDENTIFIED BY my-second-obj-id}|
        }
    my-message-abstract-syntax   ABSTRACT-SYNTAX:: =
                    message-abstract-syntax{{ MY-BODY-TYPES}}
```

附 录 B
（资料性附录）
记 法 综 述

在 GB/T 16262.1—2006 中定义了下列词项并用于本部分：

typereference

valuereference

" ::= "

" {"

" }"

" , "

在 GB/T 16262.2—2006 中定义了下列词项并用于本部分：

objectclassreference

objectreference

objectsetreference

在 GB/T 16262.1—2006 中定义了下列生成式并用于本部分：

DefinedType

DefinedValue

Reference

Type

Value

ValueSet

在 GB/T 16262.2—2006 中定义了下列生成式并用于本部分：

DefinedObjectClass

DefinedObject

DefinedObjectSet

ObjectClass

Object

ObjectSet

本部分中定义了下列生成式：

ParameterizedAssignment ::=

    ParameterizedTypeAssignment |
    ParameterizedValueAssignment |
    ParameterizedValueSetTypeAssignment |
    ParameterizedObjectClassAssignment |
    ParameterizedObjectAssignment |
    ParameterizedObjectSetAssignment

ParameterizedTypeAssignment ::=

    typereference ParameterList " ::= " **Type**

ParameterizedValueAssignment ::=

    valuereference ParameterList Type " ::= " **Value**

ParameterizedValueSetTypeAssignment ::=

typereference ParameterList Type " ::= " ValueSet

ParameterizedObjectClassAssignment ::=

objectclassreference ParameterList " ::= " ObjectClass

ParameterizedObjectAssignment ::=

objectreference ParameterList DefinedObjectClass " ::= " Object

ParameterizedObjectSetAssignment ::=

objectsetreference ParameterList DefinedObjectClass " ::= " ObjectSet

ParameterList ::= " {" Parameter " , " + " }"

Parameter ::= ParamGovernor " : " DummyReference | DummyReference

ParamGovernor ::= Governor | DummyGovernor

Governor ::= Type | DefinedObjectClass

DummyGovernor ::= DummyReference

DummyReference ::= Reference

ParameterizedReference ::=Reference | Reference " {" " }"

SimpleDefinedType ::= ExternalTypeReference | typereference

SimpleDefinedValue ::= ExternalValueReference | valuereference

ParameterizedType ::= SimpleDefinedType ActualParameterList

ParameterizedValue ::= SimpleDefinedValue ActualParameterList

ParameterizedValueSetType ::= SimpleDefinedType ActualParameterList

ParameterizedObjectClass ::= DefinedObjectClass ActualParameterList

ParameterizedObjectSet ::= DefinedObjectSet ActualParameterList

ParameterizedObject ::= DefinedObject ActualParameterList

ActualParameterList ::= " {" ActualParameter " , " + " }"

ActualParameter ::= Type | Value | ValueSet | DefinedObjectClass | Object | ObjectSet