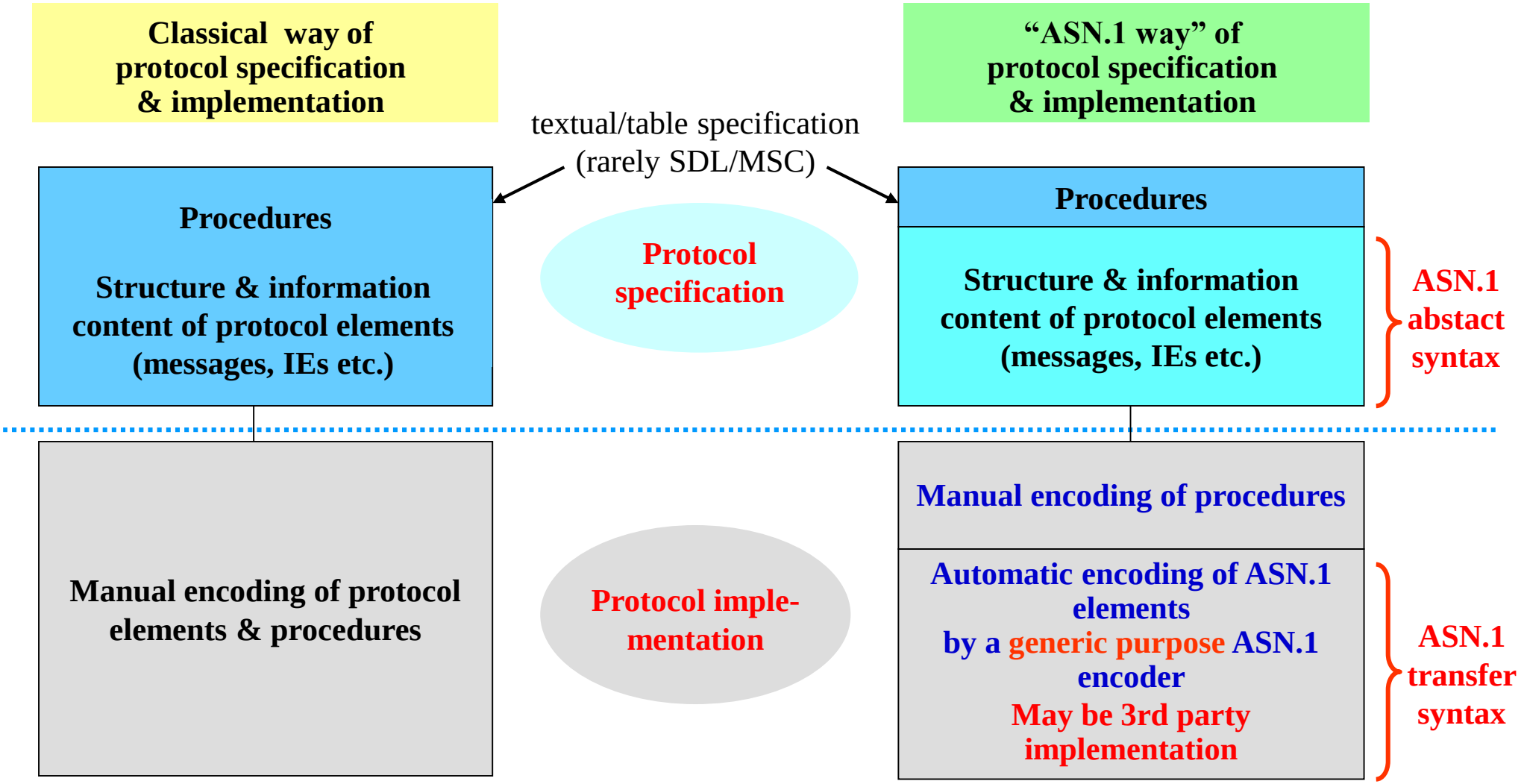
ASN.1: abstract syntax

György RÉTHY, János Zoltán SZABÓ
Test Competence Center, Ericsson Hungary

Gusztáv ADAMIS
BME TMIT
adamis@tmit.bme.hu

ASN.1 abstract syntax

INTRODUCTION



ASN.1 Standards

- **Standards describing the ASN.1 notation:**
  - ITU-T Rec. X.680 | ISO/IEC 8824-1 (Specification of basic notation)
  - ITU-T Rec. X.681 | ISO/IEC 8824-2 (Information object specification)
  - ITU-T Rec. X.682 | ISO/IEC 8824-3 (Constraint specification)
  - ITU-T Rec. X.683 | ISO/IEC 8824-4 (Parameterization of ASN.1 specifications)
- **Standards describing the ASN.1 encoding rules:**
  - ITU-T Rec. X.690 | ISO/IEC 8825-1 (BER, CER and DER)
  - ITU-T Rec. X.691 | ISO/IEC 8825-2 (PER)
  - ITU-T Rec. X.692 | ISO/IEC 8825-3 (ECN)
  - ITU-T Rec. X.693 | ISO/IEC 8825-4 (XER)
  - ITU-T Rec. X.694 | ISO/IEC 8825-5 (XSD mapping)
  - ITU-T Rec. X.695 | ISO/IEC 8825-6 (PER registration and application)
  - ITU-T Rec. X.696 | ISO/IEC 8825-7 (OER)

ASN.1 abstract syntax

CONTENTS

1. Basic conventions
2. Built-in ASN.1 types (basics of tagging and extensibility)
   NULL, BOOLEAN, INTEGER, BIT STRING, OCTET STRING, ENUMERATED, SEQUENCE (OF), SET (OF), CHOICE, Character string types, time types, REAL, ANY & other hole types: EMBEDDED PDV, EXTERNAL, Unrestricted character string
3. Tagging - additions
   Types of tags, IMPLICIT, EXPLICIT, AUTOMATIC tagging, tagging rules
4. Subtyping & constraints
   Single value, value range, size constraint, type constraint, permitted alphabet, contained subtype, inner subtyping
5. Extensibility - further rules
   ASN.1 model of extensions, extension marker, exception identifier, extensibility rules, version brackets

ASN.1 abstract syntax

Naming conventions -1

- Characters used in names

  A to Z *(latin capital)*
  a to z *(latin small)*
  0 to 9 *(digits)*
  "-" *(hyphen)*

- *Naming rules*
  - General rules:
    first character is never a digit or hyphen
    last character is never a hyphen
    hyphen shall not be followed by another hyphen
  - First character is a capital letter for:
    Type name
    Module name
  - First character is a small letter for:
    value name
    identifier (component, named value, enumeration etc.)

ASN.1 abstract syntax

Assignments

- Type

  UpperIdentifier ::= <type>

- Value

  lowerIdentifier <type> ::= <value>

ASN.1 abstract syntax

CONTENTS

1. Basic conventions, ASN.1 module definition

2. Built-in ASN.1 types

   NULL, BOOLEAN, INTEGER, BIT STRING, OCTET STRING, ENUMERATED, SEQUENCE (OF), SET (OF), CHOICE, selection type, Character string types, REAL

ASN.1 abstract syntax

NULL, BOOLEAN

- NULL
  - The only information it carries: was it SENT or NOT

```
Type-1          ::=     NULL
```

- BOOLEAN
  - Logical variable or constant

```
Type-2                  ::=     BOOLEAN
value-2     BOOLEAN     ::=     TRUE    --  FALSE
```

ASN.1 abstract syntax

INTEGER

- INTEGER
  - positive, negative integer number or (+) 0 *-- "-0" is illegal*
  - named numbers (does NOT restrict assignment of other values)

```
Sample1 ::=       INTEGER { zero (0), one (1), other (8) }
value-3       Sample1 ::= zero                    --  value is zero
value-4       Sample1 ::= 5
```

ASN.1 abstract syntax

BIT STRING

- Sequence of individual bits

  b1 BIT STRING ::= ‘000101110011010’B

  b2 BIT STRING ::= ‘F0F0’H --> ‘1111000011110000’B

ASN.1 abstract syntax

OCTET STRING

- Sequence of individual octets
  - value notation using hexstring

missing hex character is interpreted as zero

o1 OCTET STRING ::= '278EF'H --> '278EF0' H

most significant — less

  - value notation using bitstring

missing bits are interpreted as zero

o2 OCTET STRING ::= '000101101101'B --> '16E8'H

ASN.1 abstract syntax — ENUMERATED

- ENUMERATED: List of named items
  - ==>> from ASN.1 abstract syntax point of view has no numeric value !

    E1 ::= ENUMERATED { first, second, third, fourth, fifth }

  - At coding an integer value is assigned: starting at “0”, step 1

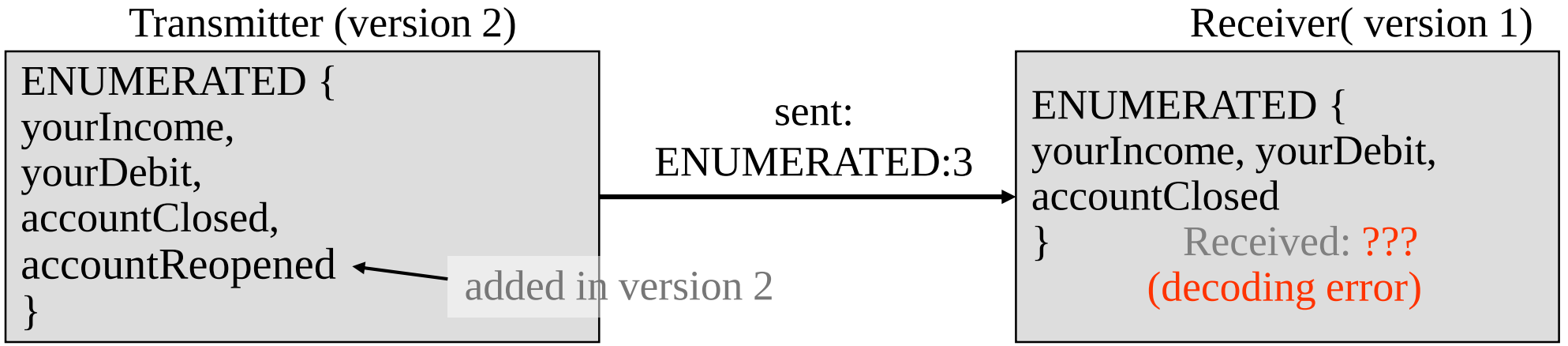
  - The value to be coded can be specified: ‘hardcoded’ values jumped over in automatic numbering

    E2 ::= ENUMERATED { first, second, third(5), fourth, fifth(1) }

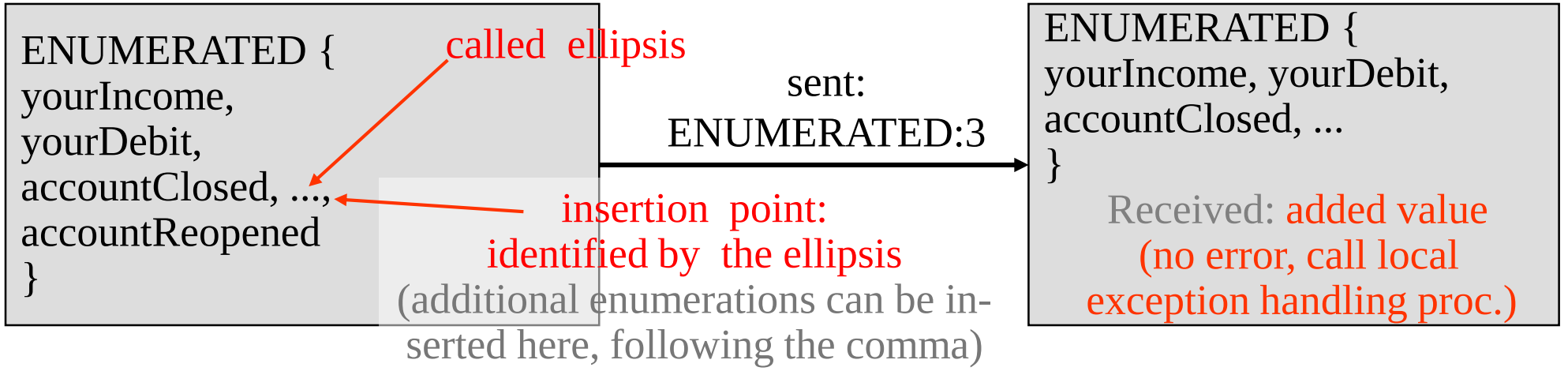
    

ASN.1 abstract syntax — ENUMERATED, extension

- The problem:



- Solution: indicate possible additions in the type definition



ASN.1 abstract syntax — Rules to extend ENUMERATED

- All values - root and additional - shall be unique
  - A ::= ENUMERATED { a, b, … , c(0) } --> *invalid as a = c*
  - B ::= ENUMERATED { a, b, … , c(2) } --> *valid*
- Values of additional enumerations shall monotonously increase
  - C ::= ENUMERATED { a, b, … , c, d(2) } --> *invalid as c = d*
  - D ::= ENUMERATED { a, b, … , c(3), d(2) } --> *invalid, it should be c < d*
  - E ::= ENUMERATED { a, b, … , c(2), d(3) } --> *valid*
- First additional “automatic” value will be the smallest not used value in the root
  - F ::= ENUMERATED { a, b, c(0), … , d, e } --> *d=3, e=4 (c=0, a=1, b=2)*
  - G ::= ENUMERATED { a, b, c(10), … , d, e } --> *d=2, e=3 (a=0, b=1, c=10)*
  - H ::= ENUMERATED { a, b, c(0), … , d, e, f(4) } --> *invalid as e = f*

The numeric value assigned to an enumeration has NO significance within the abstract syntax !

ASN.1 abstract syntax — SEQUENCE

- SEQUENCE: ordered group of types
  - Sending order of “contained” types (components) is determined by the textual order in the ASN.1 spec.
  - Each component consists of a unique (within the type) identifier and a Type => called a “named type”
  - Components may be mandatory, optional or have a default value (default: a mandatory information not necessarily sent on the line)

```
S1  ::=   SEQUENCE {  first INTEGER,
                      second      BIT STRING,
                      third       OCTET STRING   OPTIONAL,
                      fourth      BOOLEAN        DEFAULT  TRUE   }
```

  - It can be empty, i.e. without components:

```
S2  ::=   SEQUENCE { }
```

ASN.1 abstract syntax

Tagging example -1

- The problem:

Transmitter

```
SEQUENCE {
yourIncome      INTEGER OPTIONAL,
yourDebit       INTEGER OPTIONAL,
accountClosed   BOOLEAN
}
```

sent:
INTEGER:5000,
BOOLEAN:FALSE

Receiver

Is yourIncome
OR
yourDebit
received?

- Solution: add a distinguishing label

```
SEQUENCE {
yourIncome      [0] INTEGER OPTIONAL,
yourDebit           INTEGER OPTIONAL,
accountClosed       BOOLEAN
}
```

sent:
0+INTEGER:5000,
BOOLEAN FALSE

Be happy:
it is
yourIncome !

ASN.1 abstract syntax When tags shall be distinct - SEQUENCE

- In a SEQUENCE
  - Components following an OPTIONAL or DEFAULT component shall have distinct tags up to the first mandatory component (including alternatives of all referenced CHOICEs!)

```
F  ::=  SEQUENCE {  first          INTEGER,
                    second         INTEGER,
                    third          INTEGER   OPTIONAL,
                    fourth         INTEGER               }   --> invalid
```

```
G  ::=  SEQUENCE {  first          INTEGER,
                    second         INTEGER,
                    third    [0]   INTEGER   OPTIONAL,
                    fourth         OCTET STRING          }   --> valid
```

```
H  ::=  SEQUENCE {  first          INTEGER,
                    second         INTEGER,
                    third    [0]   INTEGER   OPTIONAL,
                    fourth   [1]   INTEGER               }   --> valid
```

ASN.1 abstract syntax — SEQUENCE - additions

- COMPONENTS OF
  - transports components of another SEQUENCE: for BER coded signals saves the header of the referenced SEQUENCE (!)

```
ParentSeq  ::= SEQUENCE  {  first          INTEGER          OPTIONAL,
                            second         ENUMERATED { one, two }
                                                 DEFAULT one    }

NewSeq     ::= SEQUENCE {  COMPONENTS OF ParentSeq,
                            new            INTEGER                  }
```

  - Identifiers in the parent type and in the new type shall be different
  - Before coding, the new type is created by a COMPONENTS OF transformation: components of the referenced type are inserted into the new type
  - any constraint to the referenced (outer SEQUENCE) type is ignored

```
NewSeq   =>   SEQUENCE {   first          INTEGER          OPTIONAL,
                           second         ENUMERATED { one, two }
                                                 DEFAULT one ,
                           new            INTEGER                  }
```

ASN.1 abstract syntax — Extension of a SEQUENCE

- The general case: type extensible at the middle of root components

Example:

```
I ::=  SEQUENCE {     a        A,
                      b        B,
                      ... !1,
                      ... ,
                      c        C   }
```

root components

insertion point: identified by the 1st ellipsis

end of insertions: identified by the 2nd ellipsis (following the comma there are root elements again); note, that no exception handling can be specified for this point

--> type extensible in later versions

```
I' ::=  SEQUENCE {    a        A,
                      b        B,
                      ... !1,
                      d        D,
                      e        E,
                      f        F,
                      ... ,
                      c        C   }
```

additional components

--> type added in version 2

--> type added in version 2

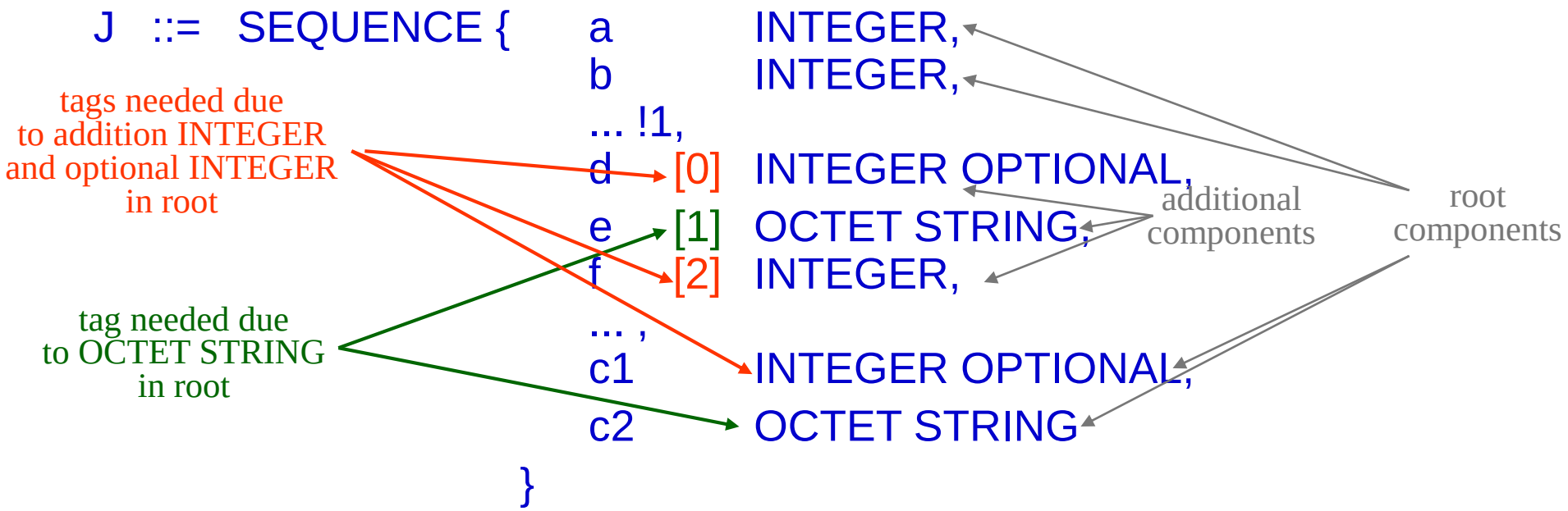
--> type added in version 3

--> type extended in versions 2 and 3

ASN.1 abstract syntax — Rules for extended SEQUENCE

– All additional types up to ad including the first mandatory root component shall have distinct tags

Example (non-automatic tagging is supposed!)



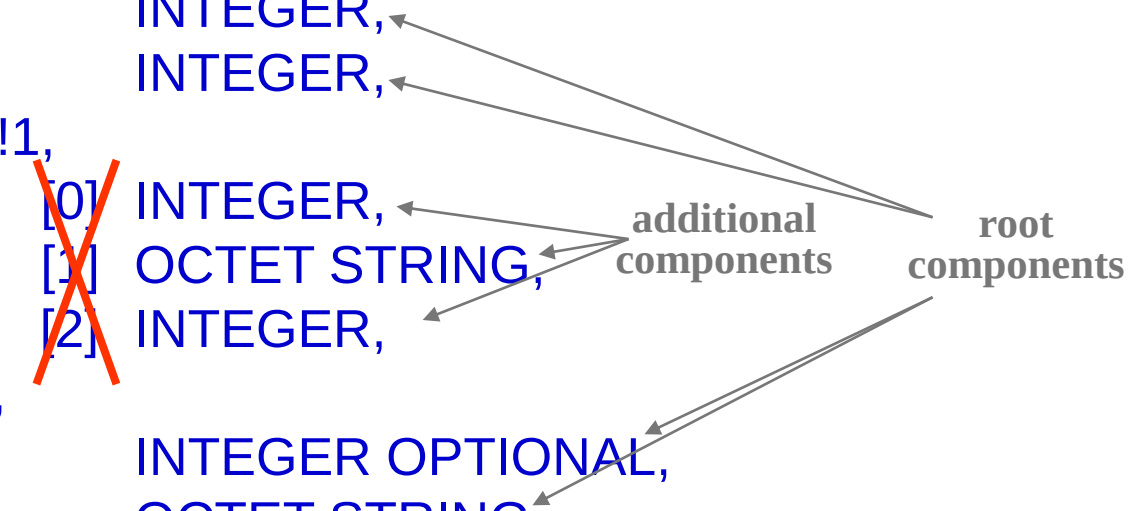
ASN.1 abstract syntax — Rules for extended SEQUENCE

– In automatic tagging environment, if none of the root components is a tagged type, the additional components shall not be tagged types

Example (automatic tagging is supposed!)

```
K   ::=   SEQUENCE {      a       INTEGER,
                          b       INTEGER,
                          ... !1,
                          d   [0] INTEGER,
                          e   [1] OCTET STRING,
                          f   [2] INTEGER,
                          ... ,
                          c1      INTEGER OPTIONAL,
                          c2      OCTET STRING

                  }
```



ASN.1 abstract syntax — Extension of a SEQUENCE

- Type extensible at the end of root components

```
J ::= SEQUENCE {     a        A,
                     b        B,
                     ... !1
                     ...
                     X
                   }
```

root components

insertion point: identified by the 1st ellipsis

"end of extensions" marker is optional

- Type extensible at the beginning

```
K ::= SEQUENCE {     ... !1,
                     ... ,
                     a        A,
                     b        B
                   }
```

root components

insertion point: identified by the 1st ellipsis

ASN.1 abstract syntax

SET

- SET: similar to SEQUENCE but unordered
  - Syntax is the same (including extension) but additional rules apply:

    1. Sending order of components is undetermined

```
S3  ::=   SET          {  first          INTEGER,
                          second         OCTET STRING    OPTIONAL,
                          third          BOOLEAN         DEFAULT  TRUE   }
```

ASN.1 abstract syntax

SET

– all components, root and additional, shall have distinct tags (including alternatives of all referenced CHOICEs!)

```
I ::= SET       { first          INTEGER,
                  second         INTEGER,
                  third    [0]   INTEGER   OPTIONAL  }   --> invalid
```

```
J ::= SET       { first          INTEGER,
                  second   [1]   INTEGER,
                  third    [0]   INTEGER   OPTIONAL  }   --> valid
```

```
K ::= SET       { first          INTEGER,
                  second   [0]   INTEGER   OPTIONAL,
                  third          OCTET STRING        }   --> valid
```

```
I' ::= SET      {first      [0]  INTEGER,
                 second          INTEGER,
                 ...,
                 third           INTEGER             }   --> invalid
```

ASN.1 abstract syntax — CHOICE

- Set of alternative types
  - Each alternative shall be marked by a unique (within this type) identifier

```
SampleChoice1  ::=     CHOICE    {
                            first                INTEGER,
                            second               BIT STRING,
                            third                OCTET STRING
                       }
```

- Additional alternatives can be inserted following the root alternatives only:

```
SampleChoice2  ::=     CHOICE   {
                            first                INTEGER,
                            second               BIT STRING,
                            third                OCTET STRING,
                            ...!1,
                            fourth               ENUMERATED,
                            fifth                NULL,
                            ...                  X
                       }
```

root alternatives

additional alternatives

"end of extensions" marker is optional

ASN.1 abstract syntax

Tagging example -2

- The problem:

Transmitter

```
CHOICE{
yourIncome      INTEGER,
accountClosed   BOOLEAN,
yourDebit       INTEGER
}
```

sent:
INTEGER:5000

Receiver

Is yourIncome
OR
yourDebit
received?

- Solution: add a distinguishing label

```
CHOICE{
yourIncome [0] INTEGER,
accountClosed   BOOLEAN,
yourDebit       INTEGER
}
```

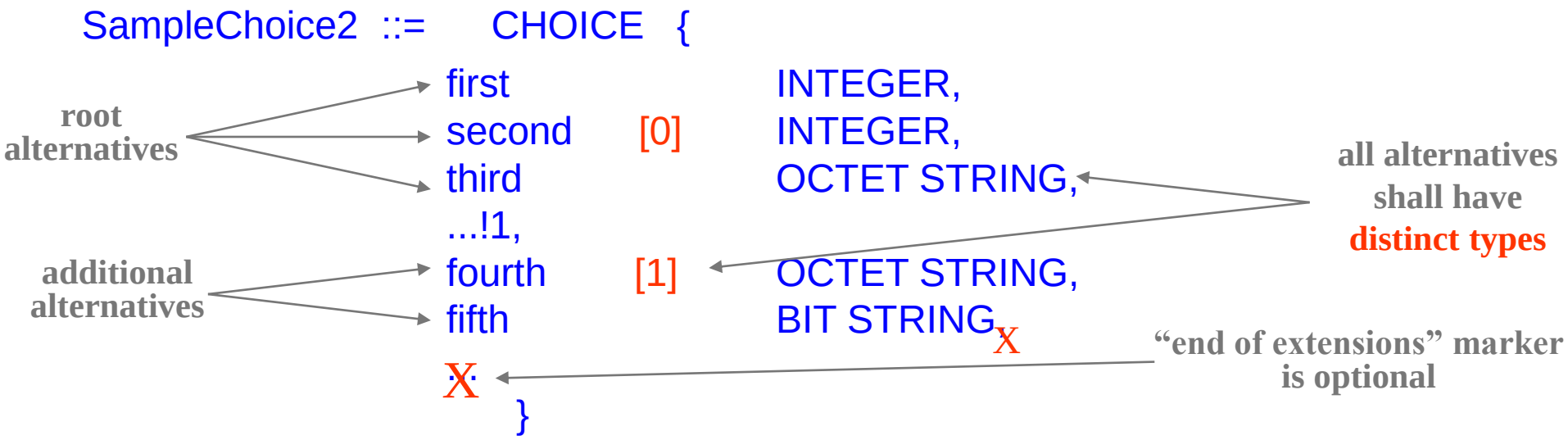
sent:
0+INTEGER:5000

Be happy:
it is
yourIncome !

ASN.1 abstract syntax

CHOICE

- All alternatives (root and additional) shall have distinct tags:



ASN.1 abstract syntax — Version brackets

- Grouping of elements added in a given version
  - can be used in SEQUENCE, SET and CHOICE, ~~NOT in ENUMERATED~~

```
Sample  ::=  SEQUENCE {       a    A,
                              b    B,
                              ...,
                          [[  c    C,
                              d    D ]], -- stuff added in version 2
                              ...        }          -- version 2 specification

Sample  ::=  SEQUENCE {       a    A,
                              b    B,
                              ...,
                          [[  c    C,
                              d    D ]], -- stuff added in version 2
                          [[  e    E,
                              f    F ]], -- stuff added in version 3
                              ...        }          -- version 3 specification
```

ASN.1 abstract syntax — SEQUENCE OF and SET OF

- SEQUENCE OF, SET OF: repeated values of a single type
  - SEQUENCE OF: sending order of values has semantic significance
  - SET OF: sending order of values does NOT have semantic significance
  - Each component is marked by an identifier (mandatory)
  - Components may be mandatory, optional or have a default value

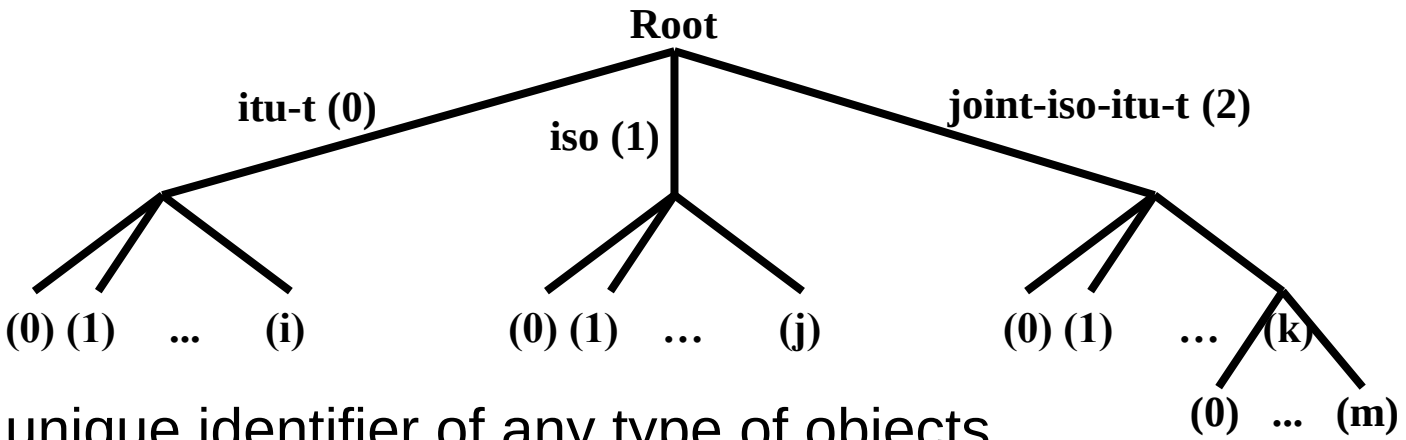
```
S1  ::=   SEQUENCE   OF  AnotherType

where     AnotherType  ::=   INTEGER        -- or another definition

          AnotherType  ::=    CHOICE {  first        INTEGER,
                                        second    BIT STRING,
                                        third      OCTET STRING,
                                        fourth     BOOLEAN   }
valueRef-1  S1  ::=  {  0, 1, 15,  7  }                  --  when AnotherType is integer
valueRef-2  S1  ::=  { first : 1, third : '0F'H }        --  when AnotherType is choice
```

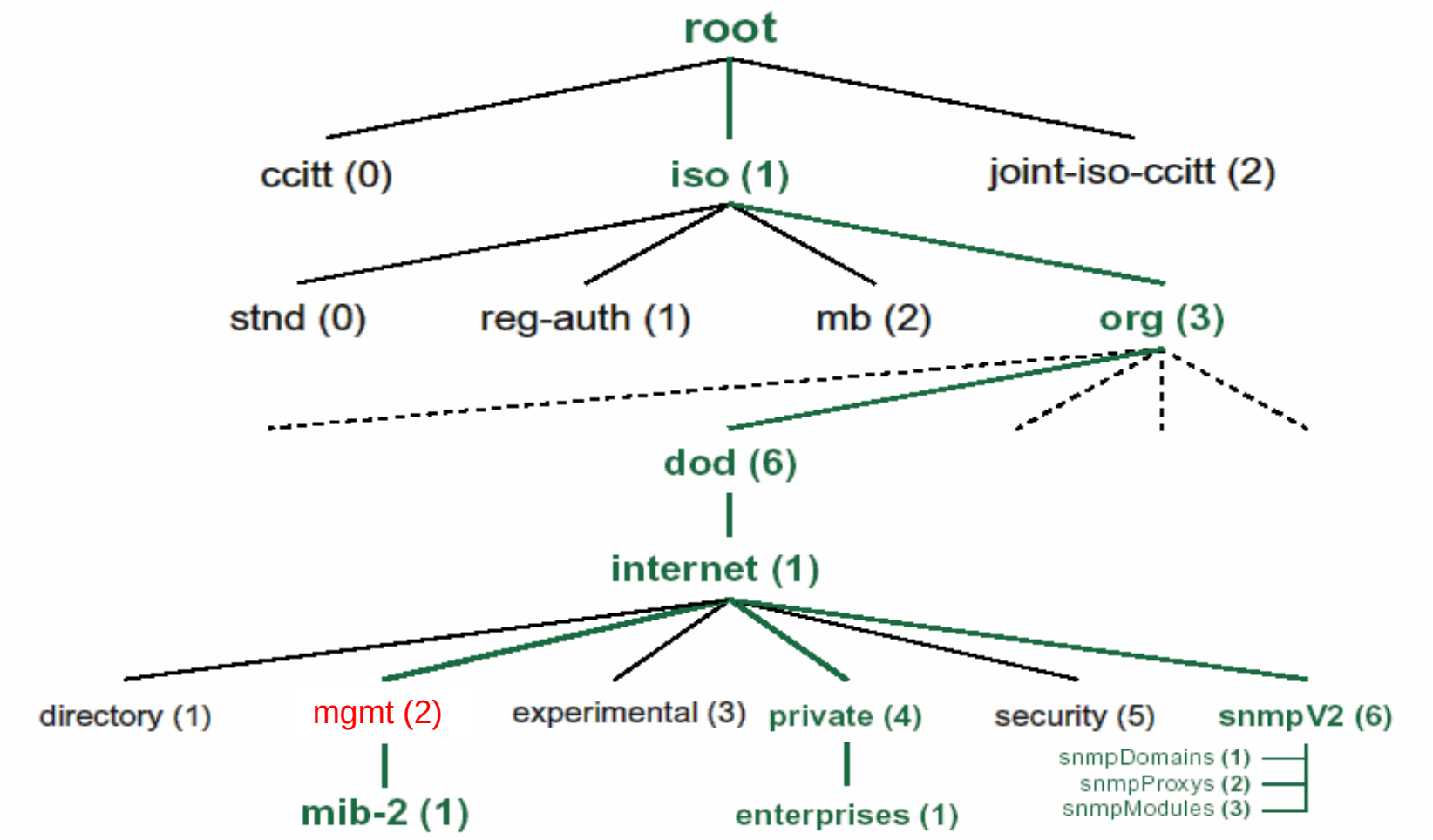
ASN.1 abstract syntax

OBJECT IDENTIFIER

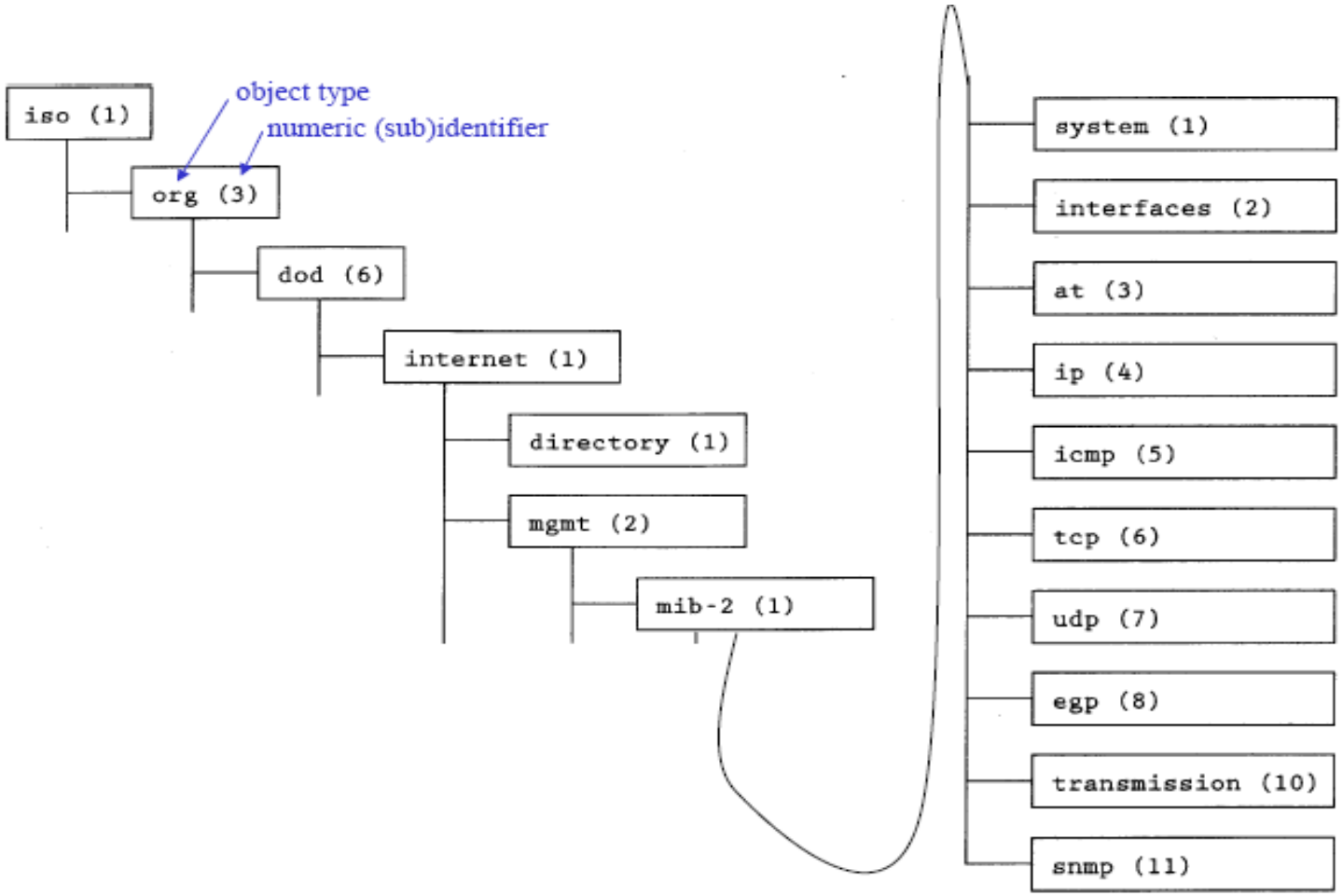


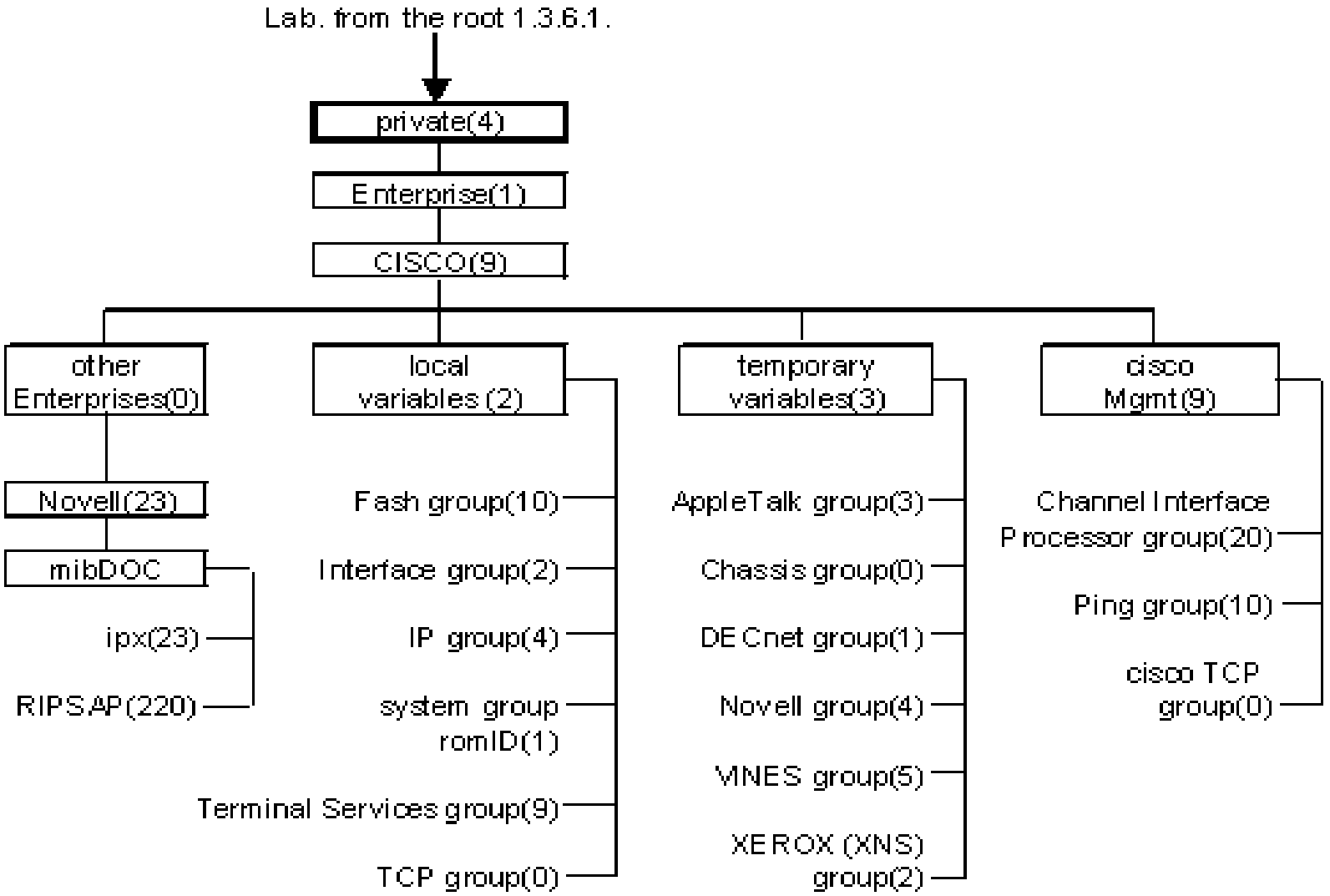
- Worldwide unique identifier of any type of objects
  - OID contains of a series of Object ID components in curly brackets
  - A component may be:
    a name (only for names defined in ITU-T Rec. X.660)
    an integer number
    a name and a number

MIB


root
ccitt (0)
iso (1)
joint-iso-ccitt (2)
stnd (0)
reg-auth (1)
mb (2)
org (3)
dod (6)
internet (1)
directory (1)
mgmt (2)
experimental (3)
private (4)
security (5)
snmpV2 (6)
snmpDomains (1)
snmpProxys (2)
snmpModules (3)
mib-2 (1)
enterprises (1)

MIB-2

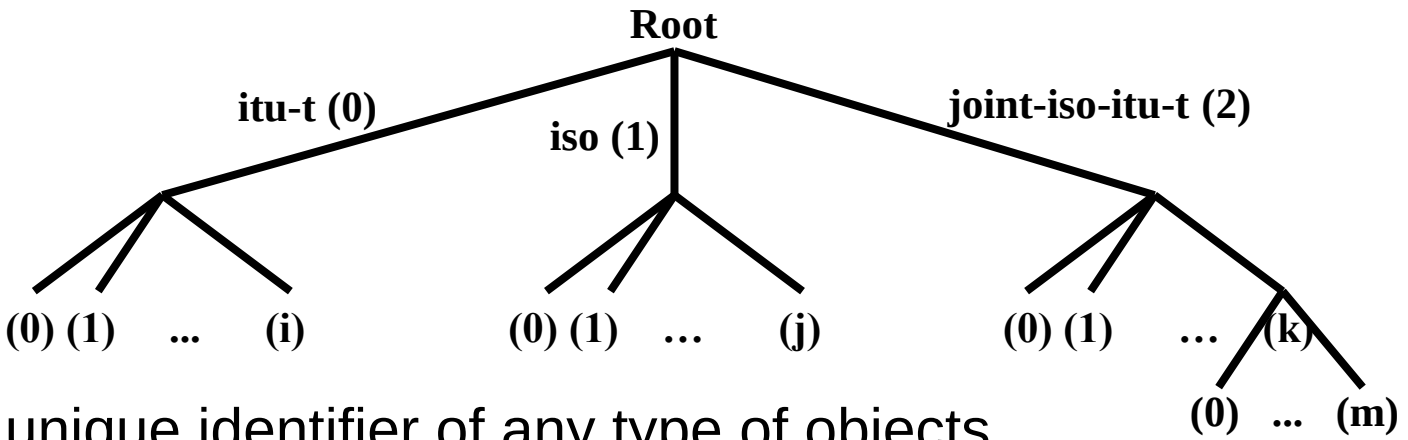

iso (1)
object type
numeric (sub)identifier
org (3)
dod (6)
internet (1)
directory (1)
mgmt (2)
mib-2 (1)
system (1)
interfaces (2)
at (3)
ip (4)
icmp (5)
tcp (6)
udp (7)
egp (8)
transmission (10)
snmp (11)



Cisco MIB rész egy eleme: 1.3.6.1.4.1.9.2.1.46 bufferFail

ASN.1 abstract syntax

OBJECT IDENTIFIER



- Worldwide unique identifier of any type of objects
  - OID contains of a series of Object ID components in curly brackets
  - A component may be:
    a name (only for names defined in ITU-T Rec. X.660)
    an integer number
    a name and a number

ericsson OBJECT IDENTIFIER ::= { itu-t(0) identified-organization(4)
etsi(0) reserved(127) etsi-identified-organization(0) ericsson(5) }

ASN.1 abstract syntax — Restricted character string types

- Restricted character string types
  - BMPString | GeneralString | GraphicString | IA5String | ISO646String | NumericString | PrintableString | TeletexString | T61String | UniversalString | UTF8String | VideotexString | VisibleString
  - NumericString has the object identifier and object descriptor:
    { joint-iso-itu-t asn1(1) specification(0) characterStrings(1) numericString(0) }
    and
    "NumericString character abstract syntax”
  - PrintableString has the object identifier and object descriptor:
    { joint-iso-itu-t asn1(1) specification(0) characterStrings(1) printableString(1) }
    and
    ”PrintableString character abstract syntax"
  - See more details at coding!

ASN.1 abstract syntax — Character string types

UNIVERSAL

- 12 UTF8String = UniversalString
- 18 NumericString -> 0..9 (space)
- 19 PrintableString -> A .. Z a .. z 0 .. 9 ‘ ( ) + , - . / : = ? (space)
- 20 TeletexString (T61String) -> ITU-T Rec. T.61
- 21 VideotexString -> ITU-T Rec. T.100 & T.101
- 22 IA5String = UniversalString (BasicLatin) (including control characters!)
- 25 GraphicString -> All G sets + SPACE
- 26 VisibleString (ISO646String) -> ISO/IEC 646:1991characters
- 27 GeneralString -> All G and C sets + SPACE + DELETE
- 28 UniversalString -> ISO/IEC 10646-1 (all characters in the Universe)
  (character space: 128 groups/256 plane/ 256 row/256 cell = 2 147 483 648)
- 30 BMPString = UniversalString (Bmp) (Basic Multilingual Plane)
  (all characters in all living languages; can be encoded on 16 bits!)

Note1: further details see in Table 3/X.680

Note2: A "CharacterStringList" notation can also be used for UniversalString, UTF8String, BMPString or IA5String -> {0,0,7,15}; for IA5String only also -> {0,7}

ASN.1 abstract syntax — Time types

- GeneralizedTime
  - Its type definition (a visible string with changed type ID):

    GeneralizedTime ::= [UNIVERSAL 24] IMPLICIT VisibleString
  - Contains calendar date, time of day (any precision) and local time differential factor according to ISO 8601
  - Formats : YYYYMMDDhhmm[ss](”Z”) | (”+|-”hhmm) |empty
    local time: “20010921214525.3” (2001. Sept. 21, 21h 45m 25.3 s)
    coordinated universal time (GMT): “20010921214525.3Z”
    local time, 5 hours retarded from GMT: “20010921214525.3-0500”
- Universal time
  - Its type definition (a visible string with changed type ID):

    UTCTime ::= [UNIVERSAL 23] IMPLICIT VisibleString
  - Contains calendar date, time (precision of 1 min. or 1 sec.) and local time differential factor from coordinated universal time
  - Formats: YYMMDDhhmm[ss](”Z”) | (”+|-”hhmm)
    local time, 5 hours retarded from UTC: “010921214525.3-0500”
    coordinated universal time (UTC): “010921214525.3Z”

ASN.1 abstract syntax

CONTENTS

1. Basic conventions, ASN.1 module definition
2. Built-in ASN.1 types

   NULL, BOOLEAN, INTEGER, OCTET STRING, BIT STRING, SEQUENCE (OF), SET (OF), CHOICE, Character string types, REAL, ANY, EMBEDDED PDV
3. Tagging

   Why we need?, types of tags, IMPLICIT, EXPLICIT, AUTOMATIC tagging, tagging rules

ASN.1 abstract syntax

Tagging example -3

- The problem (again):

Transmitter

```
SET  {
  toBePaid          INTEGER,
  haveMoney         BOOLEAN,
  toBeGet           INTEGER
}
```

sent:
INTEGER, BOOLEAN,
INTEGER

Receiver

was the order toBePaid, haveMoney and toBeGet
OR
toBeGet, haveMoney and toBePaid?

- Solution: additional label

```
SET  {
  toBePaid      [0] INTEGER,
  haveMoney         BOOLEAN,
  toBeGet           INTEGER
}
```

sent:
INTEGER, BOOLEAN,
0+INTEGER

the order was toBeGet, haveMoney and toBePaid !

ASN.1 abstract syntax

Tagging - 1

- Tagging: a transformation creating a new type from the old one
  - Commonly used: to identify a new (application or implementation specific) type, to make type components of a constructed type distinct

```
A  ::=  SEQUENCE {  first          INTEGER  OPTIONAL,
                    second         INTEGER  OPTIONAL }
B  ::=  SEQUENCE {  first    [0]   INTEGER  OPTIONAL,
                    second         INTEGER  OPTIONAL }
```

Types not distinct

Distinct types

- Tag classes:
  - UNIVERSAL: used by the ASN.1 standard only
    --> built-in ASN.1 types
  - APPLICATION: user defined types in application standards
  - PRIVATE: vendor-specific user defined type
  - context-specific: default if no tag class defined
    --> most commonly used to make type components distinct
  - only the use of UNIVERSAL is strictly limited

ASN.1 abstract syntax

Tagging - 1

- Tagging mechanisms: implicit, explicit, automatic
  - Default for the module defined in the module header
  - In IMPLICIT tagging environment:

```
C  ::=  SEQUENCE {  first       [0]                 INTEGER   OPTIONAL,
                    second      [1] EXPLICIT  INTEGER   OPTIONAL }
```

implicitly tagged explicitly tagged

  - In EXPLICIT tagging environment :

```
D  ::=  SEQUENCE {  first       [0]                 INTEGER   OPTIONAL,
                    second      [1] IMPLICIT  INTEGER   OPTIONAL }
```

explicitly tagged implicitly tagged

  - If default is AUTOMATIC: components of all SEQUENCEs, SETs and CHOICEs are tagged in implicit environment with context-specific tags before encoding by the tool, except types containing even a single hand-written tag in the ASN.1 spec.

in ASN.1

```
E  ::=  SEQUENCE {  first       INTEGER          OPTIONAL,
                    second      OCTET STRING     OPTIONAL }
```

tagged before encoding

```
E  ::=  SEQUENCE {  first       [0]    INTEGER          OPTIONAL,
                    second      [1]    OCTET STRING     OPTIONAL }
```

ASN.1 abstract syntax

UNIVERSAL class tags

- UNIVERSAL 0 *Reserved for use by the encoding rules*
- UNIVERSAL 1 BOOLEAN
- UNIVERSAL 2 INTEGER
- UNIVERSAL 3 BIT STRING
- UNIVERSAL 4 OCTET STRING
- UNIVERSAL 5 NULL
- UNIVERSAL 6 OBJECT IDENTIFIER
- UNIVERSAL 7 ObjectDescriptor
- UNIVERSAL 8 EXTERNAL | INSTANCE OF
- UNIVERSAL 9 REAL
- UNIVERSAL 10 ENUMERATED
- UNIVERSAL 11 EMBEDDED PDV
- UNIVERSAL 12 UTF8String
- UNIVERSAL 13 Relative OID
- UNIVERSAL 14-15 *Reserved*
- UNIVERSAL 16 SEQUENCE | SEQUENCE OF
- UNIVERSAL 17 SET | SET OF
- UNIVERSAL 18-22 Character strings
- UNIVERSAL 23 UTCTime
- UNIVERSAL 24 GeneralizedTime
- UNIVERSAL 25-28 Character strings
- UNIVERSAL 29 CHARACTER STRING
- UNIVERSAL 30 BMPString
- UNIVERSAL 31-... *Reserved*

ASN.1 abstract syntax — Character string types

UNIVERSAL

- **12 UTF8String = UniversalString**
- **18 NumericString** -> 0..9 (space)
- **19 PrintableString** -> A .. Z a .. z 0 .. 9 ‘ ( ) + , - . / : = ? (space)
- **20 TeletexString (T61String)** -> ITU-T Rec. T.61
- **21 VideotexString** -> ITU-T Rec. T.100 & T.101
- **22 IA5String = UniversalString (BasicLatin)** (including control characters!)
- **25 GraphicString** -> All G sets + SPACE
- **26 VisibleString (ISO646String) ->** ISO/IEC 646:1991characters
- **27 GeneralString** -> All G and C sets + SPACE + DELETE
- **28 UniversalString** -> ISO/IEC 10646-1 (all characters in the Universe)
  (character space: 128 groups/256 plane/ 256 row/256 cell = 2 147 483 648)
- **30 BMPString =** UniversalString (Bmp) (Basic Multilingual Plane)
  (all characters in all living languages; can be encoded on 16 bits!)

Note1: further details see in Table 3/X.680

Note2: A "CharacterStringList" notation can also be used for UniversalString, UTF8String, BMPString or IA5String -> {0,0,7,15}; for IA5String only also -> {0,7}

ASN.1 abstract syntax — Module definition

Example (fields are separeted by white-space)

Module-Name { <optional object identifier> } *-- name is mandatory*

DEFINITIONS -- mandatory keyword

EXPLICIT TAGS -- or
IMPLICIT TAGS -- or
AUTOMATIC TAGS
optional (see later)

EXTENSIBILITY IMPLIED *-- optional*

::= -- mandatory keyword

BEGIN -- mandatory keyword

EXPORTS *<comma separated list>* ; -- optional

IMPORTS *<comma separated list>* FROM *<module identifier>* -- optional
<comma separated list> FROM *<module identifier>*;

<assignments> *-- definitions in the module body*

END

ASN.1 abstract syntax

CONTENTS

1. Basic conventions, ASN.1 module definition
2. Built-in ASN.1 types
   NULL, BOOLEAN, INTEGER, OCTET STRING, BIT STRING, SEQUENCE (OF), SET (OF), CHOICE, Character string types, REAL, ANY, EMBEDDED PDV
3. Tagging
   Why we need?, types of tags, IMPLICIT, EXPLICIT, AUTOMATIC tagging, tagging rules
4. Subtyping
   General, single value, value range, size constraint,
   type constraint, permitted alphabet, contained subtype, inner subtyping

ASN.1 abstract syntax

Subtyping: features

- Creates a new type from the parent type!

  Parent ::=INTEGER (0..31) NewOne ::= Parent (0..15)

- May be an expression using set aritmetics
  - INTERSECTION (^), UNION (|) & EXCEPT

  Expr1 ::= INTEGER (ALL EXCEPT (0..15)) Expr2 ::= INTEGER (0..1|4..6)

- May contain extension markers (...) and exceptions(!)

  Values ::= INTEGER (0..15, … !1) Values ::= INTEGER (0..15, …, 0..31 !1)

- Very often parameterized

  ThisIsIt {INTEGER:maxIterations, INTEGER:maxNumber} ::=

  SEQUENCE (SIZE (0.. maxIterations)) OF INTEGER (0.. maxNumber)

ASN.1 abstract syntax

Subtyping - 1

- Single value constraint

  Zero ::= INTEGER (0) Yes-No ::= IA5String ("Yes" |"No")

- Value range constraint

  Seven-bit-range ::= INTEGER (0..127)

  PI ::= REAL ({mantissa 3141, base 10, exponent -3} ..
  {mantissa 3142, base 10, exponent -3})

- Permitted alphabet
  - applies to restricted character string types only (except CHARACTER STRING!)

    CapitalA ::= IA5String (FROM ("A"))
    String1 ::= IA5String (FROM ("A".. "Z") UNION FROM ("a".. ”z”))
    String2 ::= IA5String (FROM ("A".. "Z" UNION "a".. "z"))
    String1 = String2 ??? String1≠ String2 !!
    String1: strings containing all capital or all small letters only,
    String 2: capital and small letters can be mixed within a string

ASN.1 abstract syntax

Subtyping - 2

- Size constraint
  - applies to BIT STRING, OCTET STRING, SET OF, SEQUENCE OF and character string types (including CHARACTER STRING)

    String-sequence ::= SEQUENCE (SIZE(1..5)) OF IA5String (SIZE(10))

- Contained sub-type constraint
  - uses another ASN.1 type(s) to create the subtype

    Subtype1 ::= INTEGER (0..63) Subtype2 ::= INTEGER (16..31)

    NewSubtype ::= INTEGER (Subtype1 EXCEPT Subtype2)

ASN.1 abstract syntax

CONTENTS

1. Basic conventions
2. Built-in ASN.1 types
   NULL, BOOLEAN, INTEGER, OCTET STRING, BIT STRING, SEQUENCE (OF), SET (OF), CHOICE, Character string types, REAL, ANY, EMBEDDED PDV
3. Tagging
   Why we need?, types of tags, IMPLICIT, EXPLICIT, AUTOMATIC tagging, tagging rules
4. Subtyping
   General, single value, value range, size constraint,
   type constraint, permitted alphabet, contained subtype, inner subtyping
5. Extensibility
   ASN.1 model of extensions, extension marker, exception identifier, extensibility rules, version brackets

ASN.1 abstract syntax — Extensibility - general

- The only legal way to add elements to defined types LATER

- Any unknown element at an insertion point shall NOT be treated as an error by the ASN.1 decoder!!!
- Extensibility implied
  - implicitly for all extensible types by the module header
  - explicitly including an extension marker at the insertion point
- What can be extended
  - TYPES: CHOICE, SEQUENCE, SET, ENUMERATED
  - Type CONSTRAINTS (see later) : single value, value range, size constraint and permitted alphabet

ASN.1 abstract syntax — Exception handling

- Instructs the APPLICATION!, how to handle
  - type constraint violation (see later)
  - unknown elements at an extension insertion point
- Consist of:
  an exception identifier(!) and
  an exception handling rule optional Type:value

Examples

```
A ::= SEQUENCE ( SIZE ( 0 .. 15 !1 ) ) OF INTEGER ( 0 .. 65535 !2 )
    .....                                          INTEGER is assumed
        -- Error handling rules: 1 - ignore additional elements ; 2 - relay value  .....
```

```
B ::= SEQUENCE ( a  A,
                 b  B,
                 ... ! IA5String:"not comprehended element received")
```

```
C ::= SEQUENCE ( SIZE ( 0 .. 15 ! ) ) OF INTEGER
        --> comment in ASN.1 or specified in the procedures description or
        --> take implementation-dependent action
```

ASN.1 abstract syntax

Extensibility rules -2

- ASN.1 extensibility rules (cont.)
  - If an extensible type is further constrained, the new type does NOT inherit extensibility (for extensibility ellipsis shall explicitly be included)

```
A   ::=   INTEGER ( 0..63, ... )   --> extensible (parent) type
B   ::=   A ( 0..31)               --> inextensible, range 0..31
C   ::=   A ( 0..31, ... )         --> extensible, range 0..31
D   ::=   A                        --> extensible, range 0..63
```

ASN.1 abstract syntax — Parameterization

- Parameterization - general

```
FirstType  { INTEGER: param-value }  ::=
                    SEQUENCE   {
                        first          INTEGER     DEFAULT   param-value,
                        second     SecondType                 { TRUE, 5 },
                    }

SecondType { BOOLEAN: paramLOG, INTEGER: paramINT }  ::=
                    CHOICE {
                        altern1                BOOLEAN ( paramLOG ),
                        altern2                INTEGER   ( paramINT )
                        }
```

formal parameters

actual parameters

ASN.1 abstract syntax — Parameterization

- Parameterization of type notations
  - Formal parameter may be:
    type: {Type },
    value: { Type: value }

```
Sample1 { Par-Type } ::= CHOICE { altern1 BOOLEAN,
                                  altern2 Par-Type }
Sample2 { INTEGER: par-value } ::= SEQUENCE {
                                  first INTEGER DEFAULT par-value }
Sample3 { ParType, ParType: parValue } ::= SEQUENCE {
                                  seethis ParType ( parValue ) }
```