

TECHNICAL REPORT

ISO/IEC TR 9572

First edition
1989-09-15

Information technology — Open Systems Interconnection — LOTOS description of the session protocol

*Traitement de l'information — Interconnexion de systèmes ouverts — Description
en LOTOS du protocole de session*



Reference number
ISO/IEC TR 9572:1989 (E)

AVA Documents
Document Sharing Hub

Contents

	page
Foreword	iii
Introduction	iv
1 Scope	1
2 Normative references	1
3 Definitions	2
4 Symbols and abbreviations	2
5 Conventions	2
6 Introduction to the formal description	3
7 Overall constraints of the session protocol	6
8 Constraints for a single session protocol machine	7
9 Session protocol data types	8
9.1 Session protocol data unit	8
9.2 Session variables	35
9.3 Additional functions on session and transport service primitives	36
9.4 Auxiliary data types	36
9.5 Types for components of the internal event structure	39
10 Processes for the SPM session to transport service boundary relation	41
10.1 First decomposition of the session to transport boundary relation	42
10.2 Direct mapping between SSPs and TSPs	44
10.3 Relation between SSPs and SPDUs	45
10.4 Relation between SPDUs and TSPs	50
10.5 SPDU constraints	63

© ISO/IEC 1989

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) together form a system for worldwide standardization as a whole. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The main task of a technical committee is to prepare International Standards but in exceptional circumstances, the publication of a technical report of one of the following types may be proposed:

- type 1, when the necessary support within the technical committee cannot be obtained for the publication of an International Standard, despite repeated efforts;
- type 2, when the subject is still under technical development requiring wider exposure;
- type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/IEC/TR 9572, which is a technical report of type 2, was prepared by ISO/IEC JTC 1, *Information technology*.

Introduction

In view of the complexity and widespread use of Open Systems Interconnection standards it is imperative to have precise and unambiguous definitions of these standards. Formal Description Techniques form an important approach for providing such definitions. The use of Formal Description Techniques in this area is however relatively new and their application on a wide scale cannot be expected overnight. Formal descriptions should be introduced gradually in standards if initially the number of Member Bodies that are able to contribute to their development is too small, thus allowing time to gain experience and to develop educational material.

An ad-hoc group for the formal description of the Session Layer, i.e. of the Session Service ISO 8326 and the Session Protocol ISO 8327, was established in November 1985. This group applied the Formal Description Technique LOTOS, defined in ISO 8807, which at that time was still under development. In September 1986 two Working Documents were produced which contained the LOTOS draft specifications of ISO 8326 and ISO 8327 respectively. As a byproduct, the group also produced a number of Defect Reports on the standards, most of which have been accepted and incorporated in the standards.

A Ballot was then issued requesting Member Bodies to state their position concerning the progression of the formal descriptions. Based on this Ballot, SC21 decided in June 1987 to progress both formal descriptions as Type 2 Technical Reports. The main reason for not incorporating them into the standards was that Member Bodies expressed their current lack of expertise on the subject. It seemed therefore appropriate that a period of time passed, during which the formal descriptions can be read and compared with the standards, and after which the status and progression of the formal descriptions can be re-evaluated.

The purpose of this Technical Report is to provide a complete, consistent and unambiguous description of ISO 8327. It forms therefore a companion document to ISO 8327. It takes account of the Defect Reports incorporated in the standard (annex D of ISO 8327), however, it does not necessarily take account of subsequent amendments or addenda to the standard.

Information technology - Open Systems Interconnection - LOTOS description of the session protocol

1 Scope

This Technical Report contains a formal description of the OSI Basic Connection Oriented Session Protocol defined in ISO 8327. The formal definitions presented in this Technical Report are expressed in the formal description technique LOTOS, which is defined in ISO 8807.

These formal definitions are related to the formal descriptions in LOTOS of the OSI Connection Oriented Session Service, ISO 8326, and of the OSI Transport Service, ISO 8072. Moreover, formal definitions of these services, contained in ISO/IEC/TR 9571 and ISO/IEC/TR 10023, respectively, are used and referenced in this Technical Report.

The formal description is not limited to a single session protocol machine, but also describes multiple session protocol machines that result from support of multiple session connections either in parallel or in sequence. Therefore, it also formalizes aspects of multiplicity which are not presented in ISO 8327 directly, but by way of reference to the OSI Basic Reference Model, ISO 7498.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this Technical Report. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreement based on this Technical Report are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO 7498: 1984, *Information processing systems - Open Systems Interconnection - Basic Reference Model*.

ISO 8072: 1986, *Information processing systems - Open Systems Interconnection - Transport service definition*.

ISO 8326: 1987, *Information processing systems - Open Systems Interconnection - Basic connection oriented session service definition*.

ISO 8327: 1987, *Information processing systems - Open Systems Interconnection - Basic connection oriented session protocol specification*.

ISO 8807: 1988, *Information processing systems - Open Systems Interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour.*

ISO/IEC/TR 9571: 1989, *Information processing systems - Open Systems Interconnection - LOTOS description of the session service.*

ISO/IEC/TR 10023: - ¹⁾, *Information processing systems - Open Systems Interconnection - LOTOS description of the transport service.*

3 Definitions

For the purpose of this Technical Report the definitions given in ISO 8327 apply.

4 Symbols and abbreviations

This Technical Report uses the symbols defined in clause 6 (formal syntax) and annex A (data type library) of ISO 8807, and uses the abbreviations contained in clause 4 of ISO 8327.

The following additional abbreviations are employed in this Technical Report:

SC	session connection
SCEP	session connection endpoint
SCEI	session connection endpoint identifier
SSP	session service primitive
TC	transport connection
TS	Transport Service
TSDU	transport service data unit

5 Conventions

Clauses 6 through 10 of this Technical Report constitute LOTOS text. All informal explanations in these clauses form LOTOS comments. They are thus separated from the LOTOS specifications (of data types and dynamic behaviour) according to the rules for comment delimitation. Moreover, informal explanations precede the formal definitions to which they refer and contain a final line of only "-" characters. Informal explanations following formal definitions contain a first line of only "-" characters.

Formal definitions, as well as formal symbols and identifiers referenced in informal explanations, are printed in italics.

¹⁾ To be published.

(* start of LOTOS text -----

6 Introduction to the formal description

The formal description relates to the dynamic behaviour of an unbounded number of SPMs, each of which supports the provision of a single SC or, in case of re-use of the underlying TC, a sequence of SCs. The SS boundary is formally represented by a single gate *s* and the TS boundary by a single gate *t*. Events at these gates are structured as defined in the service formal descriptions, ISO/IEC/TR 9571 and ISO/IEC/TR 10023. Constraints on connection identification, acceptance of connections and backpressure flowcontrol (from the TS-provider) apply to the Session Protocol, as well as to the related services, and the processes representing these constraints are in fact imported from the service formal descriptions. Figure 1 shows the conjunction of processes resulting from this top level decomposition.

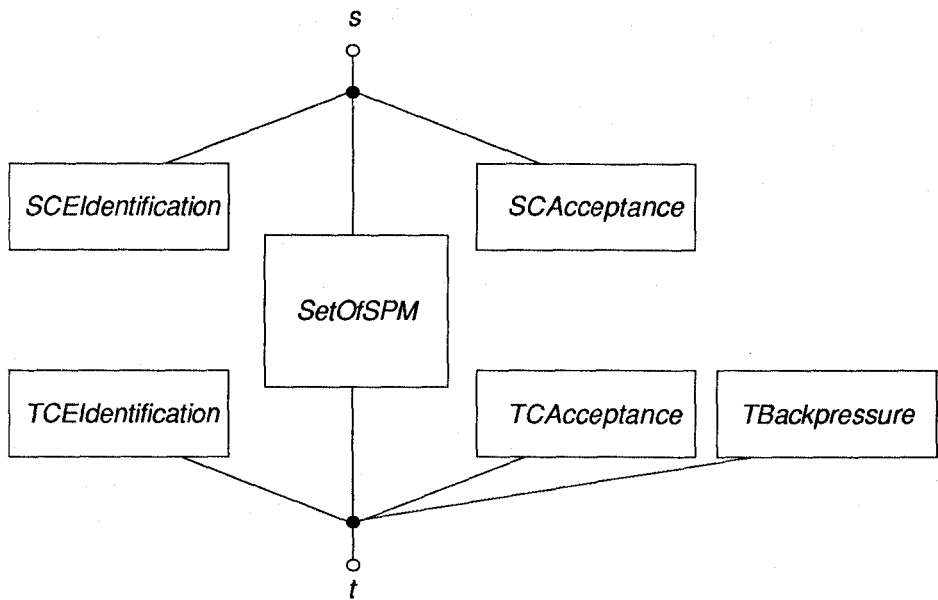


Figure 1 - (Top level) processes representing separate constraints for a Session Protocol entity

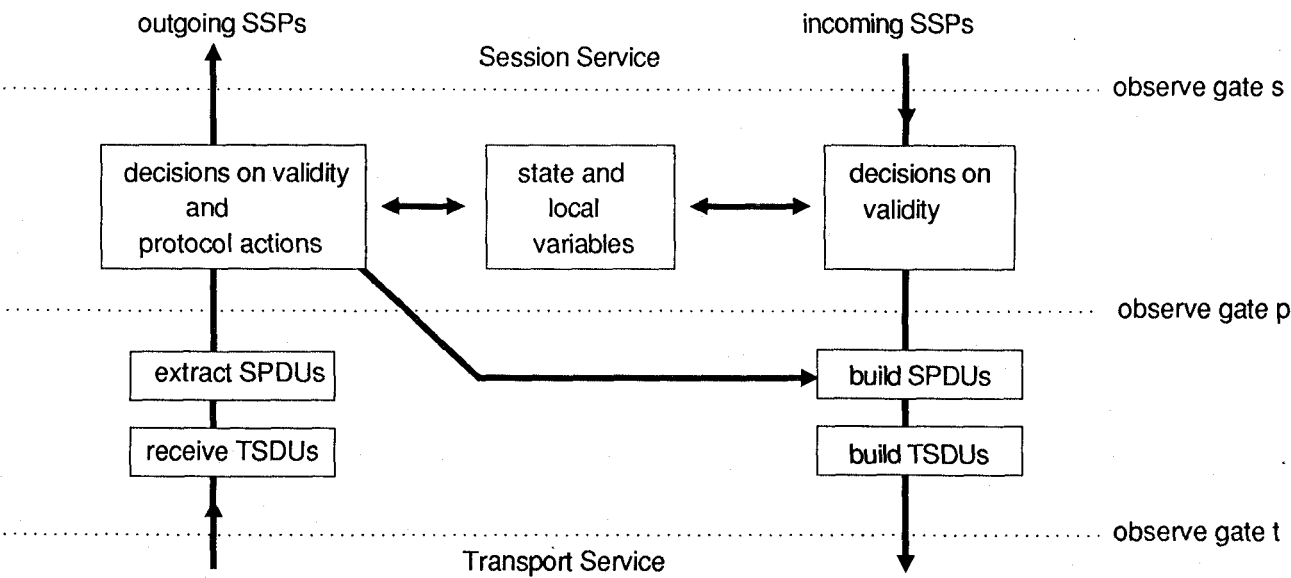


Figure 2 - Actions within a SPM

An internal gate p is introduced to favour separation of concerns based on the concept of SPDU. SPDUs are transferred via the TS by encoding, and possibly concatenating, them into TS data units. The interactions at gate p are independent of encoding and concatenation, however, thus only abstract SPDUs are to be considered at p . The actions of a single SPM are summarized in figure 2.

The behaviour of the SPM is described by specifying the constraints on the behaviour observed at the three gates s , p and t . This is obtained by a parallel composition of a process, termed *STPM*, that roughly corresponds to the state tables protocol machine, together with processes that enforce the local constraints on service primitives at connection endpoints. Again, the latter constraints can be specified with instances of processes imported from the SS and TS formal descriptions (namely *SCEP* and *TCEP*).

The next level of substructuring is found in the decomposition of *STPM* into processes representing the following separate constraints:

- The relationship between SSPs and TSPs that does not involve abstract SPDUs (described by process *SSPTSP*). This constraint is concerned with "direct mappings" between service primitives, where no SPM generated information is needed to coordinate the interworking with the remote SPM. Also the establishment of a TC is attributed to this constraint.
- The relationship between SSPs and abstract SPDUs, including segmentation of SSDUs (described by process *SSPSPDU*).
- The transformation of abstract SPDUs to TSPs, and vice versa, including encoding and concatenation of SPDUs (described by process *SPDUTSP*).
- The constraints on the ordering and contents of abstract SPDUs, corresponding to the state tables description in annex A of ISO 8327 (described by process *SPDUConstraints*).

SSPTSP and *SSPSPDU* synchronize at gate s , *SSPTSP* and *SPDUTSP* synchronize at gate t , and, finally, *SSPSPDU*, *SPDUTSP* and *SPDUConstraints* synchronize at gate p . This is shown in figure 3. The style of the formal description, which now also shows a limited internal structure, can be characterized as a mixture of constraint- and resource-oriented styles.

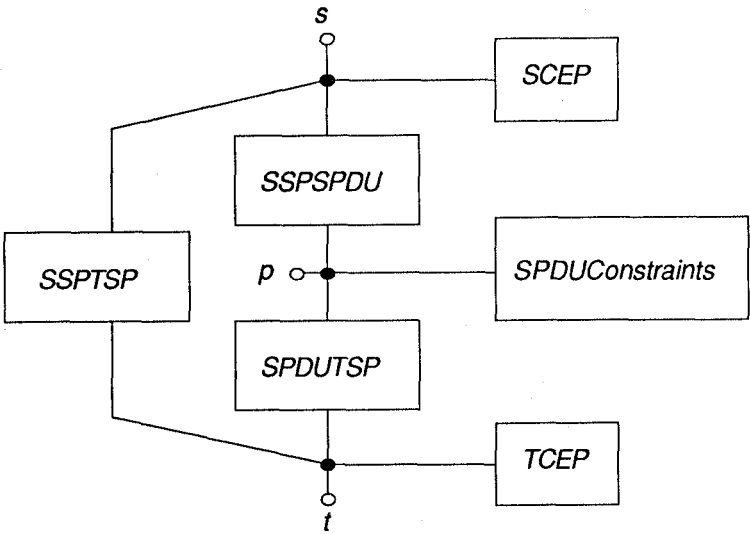


Figure 3 - Conjunction of processes representing the behaviour of a single SPM

Each of the processes that must synchronize at p imposes its own constraints on the exchange of abstract SPDUs. Because of this, an abstract SPDU exchange only occurs if all processes agree on that particular

exchange. The event structure of the abstract SPDU exchange is as follows:

$p ?pd:ASPDU ?fl:TFlow ?d:Dir ?v:Validity ?st:TrState$

with components

- pd of sort *ASPDU*: the abstract SPDU being exchanged;
- fl of sort *TFlow*: the transport flow, which is either normal or expedited;
- d of sort *Dir*: the direction of transfer, which is either sending or receiving;
- v of sort *Validity*: the "status" of the abstract SPDU (only applicable for received SPDUs). According to the state tables of ISO 8327, processing of a received SPDU, and in particular determining its validity, depends upon the result of several actions (see figure 2) and/or the values of certain session variables. Relevant states that involve more than one process can be established through this component when synchronizing at p ; and
- st of sort *TrState*: the "coarse-grained" state of the SPM (only applicable for received SPDUs and for sending the DN SPDU). The SPM state, known to process *SPDUConstraints*, is often also required by other processes upon exchange of a SPDU. However, rather than the fine-grained division of the data transfer phase into SPM states, certain collections of SPM states are relevant for the latter processes. Values of this component are used to represent such collections and can be communicated when synchronizing at p .

NOTES

1 - In order to represent flow control and segmenting, events at gates s and t may represent the exchange of single data octets instead of just executions of "complete" service primitives. Octet-wise exchange of data is only modelled for those service primitives whose user data parameter has no length restrictions (i.e., S-DATA, S-TYPED-DATA and T-DATA). Except for the need of "data octet" events, the characteristics of octet-wise exchange of data are

- two events, a start and an end event, are significant to delimitate an exchange of data;
- the state transition and predicates that are associated with a SSP request are now associated with the corresponding start event. If a colliding or overtaking, that is, "intervenient", event occurs before the end event, then either
 - a) the (unfinished) request is not disrupted (provided the intervenient event is not destructive), but will be resumed after the intervenient event, or
 - b) the (unfinished) request is disrupted, and the corresponding TSP request is cancelled or terminated;
- the state transition and predicates that are associated with a TSP indication are now associated with the corresponding end event. If a destructive or overtaking event occurs before the end event, then the (unfinished) indication is disrupted, and all previous events of the indication are discarded.

2 - The formal description extends beyond the state tables description of ISO 8327 by way of explicitly describing

- the constraints related to the multiplicity of SPMs;
- concatenation, extended concatenation and segmentation;
- the transfer of expedited data;
- the handling of invalid and unrecognizable events.

Figure 4 gives an overview of the processes used for the formal definition of the Session Protocol and their relations (a number of the lowest level processes are omitted). It also indicates the clauses where the definition of these processes can be found ("SS" means: defined in ISO/IEC/TR 9571; "TS" means: defined in ISO/IEC/TR 10023).

The definition of data types precedes the definition of the dynamic behaviour in which these types are used. A number of standard data types are imported from the LOTOS library of data types. Apart from these, and some "ad-hoc" data types, the types can roughly be divided in types for the representation of abstract SPDUs, of encoded SPDUs, and of session variables. Additionally, a number of types are related to the components of an internal event at gate p (besides the abstract SPDU).

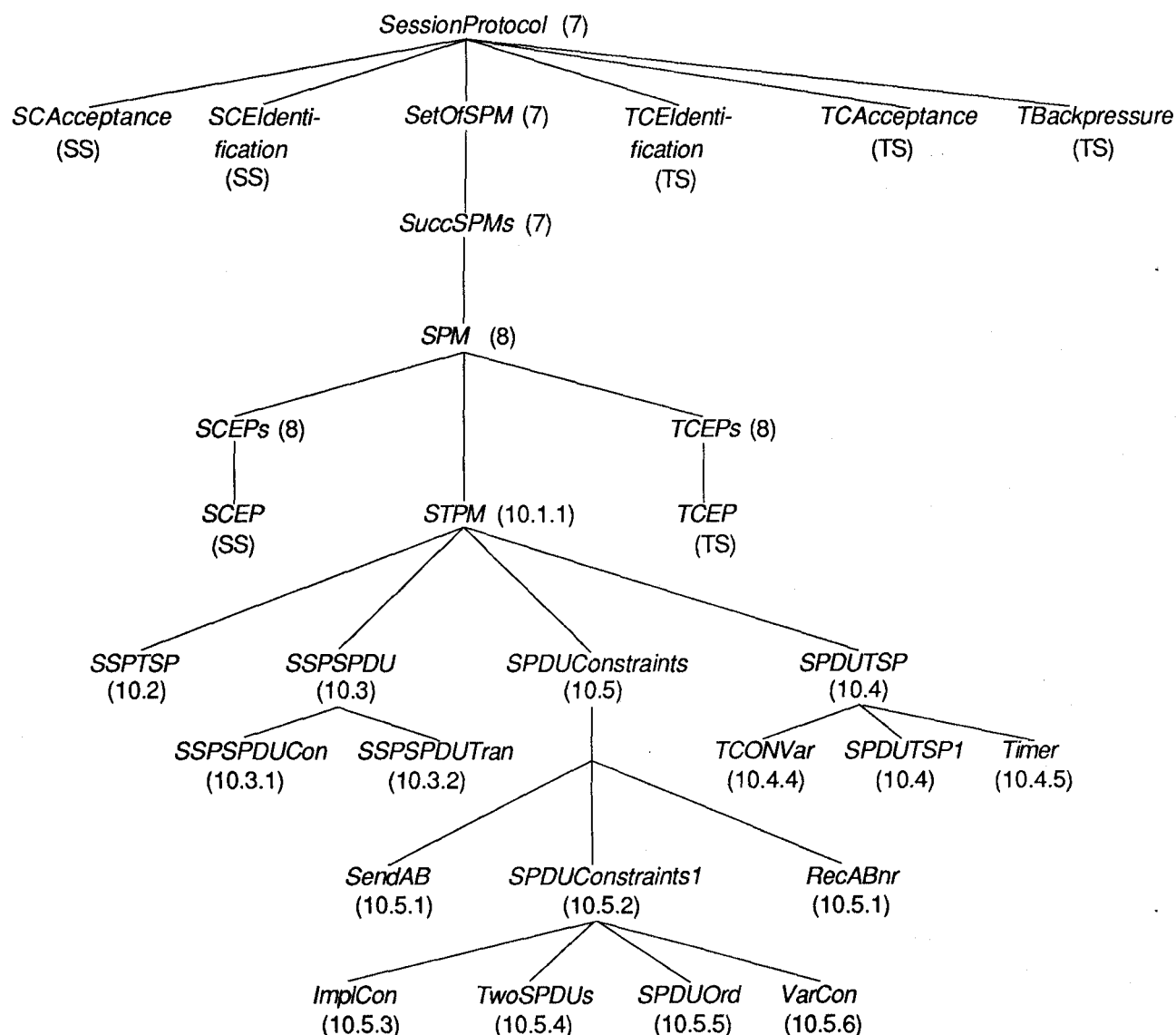


Figure 4 - Processes related by a tree structure: each "father" process contains (or is constructed from) one or more instances of its immediate "descendant" process(es)

7 Overall constraints of the session protocol

A number of standard types are imported from the LOTOS library of data types by means of the *library* construct. The specification is parameterized with a protocol implementation parameter, defined in type *SProtocolImplementationPar*. Its value indicates: 1) whether or not use of the transport expedited service is implemented, 2) what is the maximum TSDU size for both directions of data flow, 3) which functional units have been implemented and 4) whether or not extended concatenation is implemented.

The top level structure shows a full synchronization of process *SetOfSPM* with two behaviour expressions which mutually interleave, viz. *SCEIdentification* synchronized with *SCAcceptance* on the one hand, and *TCEIdentification* synchronized with *TCAcceptance* and *TBackpressure* on the other. *SetOfSPM* defines the dynamic behaviour of a potentially infinite number of independent SPMs, however disregarding any concerns related to the limited capacity of the local system and the TS-provider and the need for unique connection identification. The latter concerns are addressed by the two behaviour expressions mentioned. They formulate the overall constraints that relate to a single service

boundary; the processes representing these constraints are defined in the Session Service formal description, ISO/IEC/TR 9571, and the Transport Service formal description, ISO/IEC/TR 10023, respectively.

SetOfSPM represents concurrency by multiple instances of *SuccSPMs*, where each instance describes a succession of SPM behaviours. A SPM is represented by a distinct instance of *SPM*. The behaviour described by *SPM* is that of using at most one TC for support of a single SC or multiple successive SCs (based on the same TC). Termination of a TC enforces successful termination of the associated instance of *SPM*.

-----*)
specification *SessionProtocol* [s,t] (*spei*:*SPEImplementation*): **noexit**

library *Boolean,Element,Set,String,NatRepresentations,OctetString,NaturalNumber,Bit,FBoolean,Octet,DecNatRepr,BitString,BitNatRepr,DecString,DecDigit*
endlib

type *SProtocolImplementationPar* **is** *SRequirements, TSDUSize*

sorts *SPEImplementation*

opns

SPEImpl: *Bool, SRqms, TSDUSize, Bool* -> *SPEImplementation*

MaxTSize: *SPEImplementation* -> *TSDUSize* *ExtConc*: *SPEImplementation* -> *Bool*

eqns forall *tex,ExtConc:Bool, sfus:SRqms, tsduSize:TSDUSize*

ofsort *TSDUSize* *MaxTSize*(*SPEImpl*(*tex,sfus,tsduSize,ExtConc*)) = *tsduSize*;

ofsort *Bool* *ExtConc*(*SPEImpl*(*tex,sfus,tsduSize,ExtConc*)) = *ExtConc*;

endtype

behaviour

((*SCEIdentification*[s] || *SCAcceptance*[s]) ||
 (*TCEIdentification*[t] || *TCAcceptance*[t] || *TBackpressure*[t])
) || *SetOfSPM*[s,t](*spei*)

where

process *SetOfSPM*[s,t](*spei*:*SPEImplementation*): **noexit** := *SuccSPMs*[s,t](*spei*) || *SetOfSPM*[s,t](*spei*)

where

process *SuccSPMs*[s,t](*spei*:*SPEImplementation*): **noexit** := *SPM*[s,t](*spei*) >> *SuccSPMs*[s,t](*spei*) **endproc**

endproc (* *SetOfSPM* *)

(*-----*)

8 Constraints for a single session protocol machine

Three separate constraints are distinguished with respect to the behaviour of a single SPM, namely constraints that are local to the SS boundary, constraints that are local to the TS boundary, and constraints that relate the behaviour at one boundary to that at the other. These constraints are represented by *SCEPs*, *TCEPs* and *STPM*, respectively.

SCEPs is described as the choice of two instances of *SCEP* that apply to the interaction with the Calling SS-user and with the Called SS-user, respectively, followed by either successful termination or another instance of *SCEPs*. The recursion applies only when the SPM re-uses the TC. Similarly, *TCEPs* is described as the choice of two instances of *TCEP* that apply to the interaction with the Calling TS-user and with the Called TS-user, respectively. The definitions of *SCEP* and *TCEP* are imported from ISO/IEC/TR 9571 and ISO/IEC/TR 10023 respectively.

STPM defines the mapping of SSPs to TSPs during the provision of one SC, or multiple successive SCs, supported by a single TC.

```

-----*)
process SPM[s,t](spei:SPEImplementation): noexit :=
  SCEPs[s] |[s]| STPM[s,t](spei) |[t]| TCEPs[t]
where
process SCEPs[s]: exit := ( SCEP[s](calling) [] SCEP[s](called) ) >> ( exit [] SCEPs[s] ) endproc
process TCEPs[t]: exit := TCEP[t](CallingRole) [] TCEP[t](CalledRole) endproc
endproc (* SPM *)
-----*)

```

9 Session protocol data types

The Session Protocol data types consist of definitions for the construction of a SPDU (see 9.1), some session variables (see 9.2) for the introduction of some additional functions related to SSPs and TSPs (see 9.3), and for the construction of components of the internal event structure (see 9.5). A number of auxiliary definitions of general use are presented in 9.4.

9.1 Session protocol data unit

Two sets of definitions are distinguished for the construction of a SPDU: one for an "abstract" SPDU and the other for an encoded SPDU. This division is consistent with the observation that the elements of procedure related to SPDUs can be described independently of the encoding of SPDUs (i.e., the mapping into TS data units).

9.1.1 Abstract SPDU

The specification of the abstract SPDU type is accomplished by way of a number hierarchical type definitions. First the basic construction of SPDUs is presented (see 9.1.1.1), then a classification of SPDUs (see 9.1.1.2), and subsequently a number of additional functions on SPDUs (see 9.1.1.3 through 9.1.1.5) Concatenation of SPDUs is defined in 9.1.2; (groups of) SPDU parameters are defined in 9.1.3 (in as far as not defined in ISO/IEC/TR 9571).

9.1.1.1 SPDU basic construction

Type *BasicASPDU* defines functions, referred to as "constructor" functions, that yield (abstract) SPDU values. For each class, or "type", of SPDU a corresponding constructor function is defined with values of the parameters of that SPDU as arguments. Additionally, a function *DUM* is introduced for conveniently specifying error reporting. Definitions that relate to SPDU parameters are imported by *BasicASPDU* (most of them indirectly, by importing type *SessionServicePrimitive*).

```

-----*)
type BasicASPDU is SessionServicePrimitive, SPReference, CAltem, TDISPar, Prepare, Encltem
sorts ASPDU
opns
CN: SPReference, CAltem, SFUs, SAddress, SAddress, SData -> ASPDU
AC: SPReference, CAltem, STokens, SFUs, SAddress, SAddress, SData -> ASPDU
RF: SPReference, TDISPar, SFUs, Nat (*Version*) , DatOctStr (*Reason Code*) -> ASPDU
FN: TDISPar, SData -> ASPDU
DN, NF: SData -> ASPDU
AB: TDISPar, Nat (*Reason Code*), DatOctStr (*Reflect Parameter Values*), SData -> ASPDU
GTC, GTA, AA, AIA, ADA: -> ASPDU
DT, TD: Enclt, SData -> ASPDU
CD, EX, CDA: SData -> ASPDU
GT: STokens (*Token Item parameter*) -> ASPDU
PT: STokens, SData -> ASPDU
MIP: SSynType, SSPSN, SData -> ASPDU
MIA: SSPSN, SData -> ASPDU
MAP, AE, MAA, AEA: SSPSN, SData -> ASPDU
RA: STsAss, SSPSN, SData -> ASPDU
RS: STsAss, SResynType, SSPSN, SData -> ASPDU
PR: PrepType -> ASPDU
AS: SActId, SData -> ASPDU

```

AR: SCRef, SActId, SSPSN, SActId, SData -> ASPDU

AI, AD: Nat (*Reason Code*) -> ASPDU

ER: DatOctStr -> ASPDU

endtype

DUM: DatOctStr -> ASPDU

ED: Nat (*Reason Code*), SData -> ASPDU

(*-----*)

9.1.1.2 SPDU classification

Type *ASPDUConstant* defines a classification of SPDUs. Each class of SPDU is represented by a constant with a name similar to the abbreviated name to denote the SPDU in table 39 of ISO 8327. The auxiliary function *h* that maps these constants to natural numbers is defined in order to simplify the definition of equality on SPDU classes.

Type *ASPDUClassifiers* is a functional enrichment of *BasicASPDU*. It defines functions, referred to a "recognizer" functions, that determine whether a given SPDU is of a certain class.

-----*)

type *ASPDUConstant* **is** *NaturalNumber*

sorts *ASPDUConstant*

opns

cn, ac, rf, fn, dn, nf, ab, aa, dt, ex, td, cd, gt, pt, gtc, gta, mip, mia, map, maa, rs, pr, ra, er, ed, as, ar, ad, ai, ada, aia, ae, aea,
dum: -> *ASPDUConstant*

h: *ASPDUConstant* -> *Nat*

*_eq_, _ne_*: *ASPDUConstant, ASPDUConstant* -> *Bool*

eqns forall *x, y*: *ASPDUConstant*

ofsort *Nat*

h(*cn*) = 0;

h(*fn*) = *Succ*(*h*(*rf*));

h(*ab*) = *Succ*(*h*(*nf*));

h(*ex*) = *Succ*(*h*(*dt*));

h(*cda*) = *Succ*(*h*(*cd*));

h(*gtc*) = *Succ*(*h*(*pt*));

h(*mia*) = *Succ*(*h*(*mip*));

h(*rs*) = *Succ*(*h*(*maa*));

h(*er*) = *Succ*(*h*(*ra*));

h(*ar*) = *Succ*(*h*(*as*));

h(*ada*) = *Succ*(*h*(*ai*));

h(*aea*) = *Succ*(*h*(*ae*));

h(*ac*) = *Succ*(0);

h(*dn*) = *Succ*(*h*(*fn*));

h(*aa*) = *Succ*(*h*(*ab*));

h(*td*) = *Succ*(*h*(*ex*));

h(*gt*) = *Succ*(*h*(*cda*));

h(*gta*) = *Succ*(*h*(*gtc*));

h(*map*) = *Succ*(*h*(*mia*));

h(*pr*) = *Succ*(*h*(*rs*));

h(*ed*) = *Succ*(*h*(*er*));

h(*ad*) = *Succ*(*h*(*ar*));

h(*aia*) = *Succ*(*h*(*ada*));

h(*dum*) = *Succ*(*h*(*aea*));

h(*rf*) = *Succ*(*h*(*ac*));

h(*nf*) = *Succ*(*h*(*dn*));

h(*dt*) = *Succ*(*h*(*aa*));

h(*cd*) = *Succ*(*h*(*td*));

h(*pt*) = *Succ*(*h*(*gt*));

h(*mip*) = *Succ*(*h*(*gta*));

h(*maa*) = *Succ*(*h*(*map*));

h(*ra*) = *Succ*(*h*(*pr*));

h(*as*) = *Succ*(*h*(*ed*));

h(*ai*) = *Succ*(*h*(*ad*));

h(*ae*) = *Succ*(*h*(*aia*));

ofsort *Bool*

x eq y = *h*(*x*) *eq* *h*(*y*);

x ne y = *h*(*x*) *ne* *h*(*y*);

endtype

type *ASPDUClassifiers* **is** *BasicASPDU, ASPDUConstant*

opns

IsCN, IsAC, IsRF, IsFN, IsDN, IsNF, IsAB, IsGTC, IsGTA, IsAA, IsDT, IsTD, IsCD, IsEX, IsCDA, IsGT, IsPT, IsMIP,
IsMIA, IsMAP, IsMAA, IsAEA, IsRS, IsRA, IsPR, IsAS, IsAR, IsAI, IsAD, IsAIA, IsADA, IsAE, IsCAT21, IsACK, IsPAB,
IsER, IsED, IsDUM, IsPR_RS, IsPR_RA, IsPR_MAA, IsACT: *ASPDU* -> *Bool*

k: *ASPDU* -> *ASPDUConstant*

eqns forall *spr*: *SReference*, *sreq*: *SFUs*, *cait*: *CAltEm*, *cg*, *cd*: *SAddress*, *d*: *SData*, *tk*: *STokens*,

tdpar: *TDISPar*, *vs*, *reas*: *Nat*, *rcode*, *rpv*, *err*: *DatOctStr*, *sn*: *SSPSN*, *a*: *STsAss*, *et*: *Enclt*, *sntp*: *SSyncType*,

rt: *SResynType*, *pt*: *PrepType*, *aid1*, *aid2*: *SActId*, *scr*: *SCRef*, *pd*: *ASPDU*

ofsort *ASPDUConstant*

k(*CN*(*spr*, *cait*, *sreq*, *cg*, *cd*, *d*)) = *cn*;

k(*AC*(*spr*, *cait*, *tk*, *sreq*, *cg*, *cd*, *d*)) = *ac*;

k(*RF*(*spr*, *tdpar*, *sreq*, *vs*, *rcode*)) = *rf*;

k(*FN*(*tdpar*, *d*)) = *fn*;

k(*DN*(*d*)) = *dn*;

k(*NF*(*d*)) = *nf*;

k(*AB*(*tdpar*, *reas*, *rpv*, *d*)) = *ab*;

k(*GTC*) = *gtc*;

k(*GTA*) = *gta*;

k(*AA*) = *aa*;

k(*AIA*) = *aia*;

k(*ADA*) = *ada*;

k(*DT*(*et*, *d*)) = *dt*;

k(*TD*(*et*, *d*)) = *td*;

k(*CD*(*d*)) = *cd*;

k(*EX*(*d*)) = *ex*;

k(*CDA*(*d*)) = *cda*;

k(*GT*(*tk*)) = *gt*;

```

k(PT(tk,d)) = pt;
k(MAP(sn,d)) = map;
k(AEA(sn,d)) = aea;
k(PR(pt)) = pr;
k(AI(reas)) = ai;
k(ED(reas,d)) = ed;
ofsort Bool
IsCN(pd) = k(pd) eq cn;
IsFN(pd) = k(pd) eq fn;
IsAB(pd) = k(pd) eq ab;
IsAA(pd) = k(pd) eq aa;
IsTD(pd) = k(pd) eq td;
IsMIP(pd) = k(pd) eq mip;
IsMIA(pd) = k(pd) eq mia;
IsAEA(pd) = k(pd) eq aea;
IsPR(pd) = k(pd) eq pr;
IsAS(pd) = k(pd) eq as;
IsAD(pd) = k(pd) eq ad;
IsAE(pd) = k(pd) eq ae;
IsACK(pd) = (IsMIA(pd) or IsMAA(pd) or IsAEA(pd) or IsRA(pd) or IsAIA(pd) or IsADA(pd) or IsCDA(pd));
IsCAT21(pd) = (IsRS(pd) or IsRA(pd) or IsAD(pd) or IsADA(pd) or IsAI(pd) or IsAIA(pd) or IsER(pd) or
  IsED(pd) or IsCD(pd) or IsCDA(pd));
not(IsAB(pd)) => IsPAB(pd) = false;
IsPAB(AB(tdpar,reas,rvp,d)) =
  ((reas eq Succ(Succ(0))) and (Length(rvp) le NatNum(Dec(9)))) or
  ((reas eq NatNum(Dec(3))) and Empty(rvp)) and Empty(d);
IsPR_RS(PR(pt)) = pt eq res;
IsPR_RA(PR(pt)) = pt eq rack;
IsPR_MAA(PR(pt)) = pt eq mack;
IsACT(pd) = IsAR(pd) or IsAS(pd) or IsAI(pd) or IsAD(pd) or IsAE(pd) or IsAIA(pd) or IsADA(pd) or
  sAEA(pd);
endtype
(*)-----

```

9.1.1.3 SPDU parameter selectors

Type *ASPDUPParameterSelectors* defines functions, referred to "extractor" functions, that allow to determine the value of individual SPDU parameters. It imports *SLocalTokensState* from ISO/IEC/TR 9571 and *Compare*, an auxiliary type (see 9.4.2).

```

-----*)
type ASPDUPParameterSelectors is ASPDUClassifiers, SLocalTokensState, Compare
opns
ErrStr,RPV: ASPDU -> DatOctStr
SPRef: ASPDU -> SPReference
Version: ASPDU -> Nat
TDISPar: ASPDU -> TDISPar
Preptype: ASPDU -> PrepType
MTSDUPar: ASPDU -> TSDUSize
FUs: ASPDU -> SFUs
TokenSet,InvTokenSet: ASPDU -> STsAss
Tokens: ASPDU -> STokens
SPSN: ASPDU -> SSPSN
CgAddr,CdAddr: ASPDU -> SAddress
ResynType: ASPDU -> SResynType
Reason: ASPDU -> Nat
MtsRecFlow,MtsSendFlow: ASPDU -> Nat
UserInf: ASPDU -> SData
Encltem: ASPDU -> Enclt
InitialAss: ASPDU -> SLTsState
ProtOptions: ASPDU -> Bool

```

eqns forall *sn*:SSPSN, *cait*:CAItem, *spr*:SPReference, *vs*:Nat, *sdat*:SData, *prt*:PrepType, *rpv*:DatOctStr,
srqms:SFUs, *sa1*,*sa2*:SAddress, *srf*:SCRef, *mt*,*mtr*,*mts*:Nat, *tkass*:STsAss, *rtyp*:SResynType,
tsize:TSDUSize, *actid1*,*actid2*:SActId, *reas*:Nat, *str*:DatOctStr, *tdpar*:TDISPar, *encit*:Enclt,
stok,*CgTokens*,*CdTokens*:STokens, *tk*:SToken, *styp*:SSyncType

ofsort DatOctStr

ErrStr(DUM(str)) = str;

ofsort SSPSN

SPSN(CN(spr,cait,srqms,sa1,sa2,sdat)) = SSPSN(cait);

SPSN(AC(spr,cait,stok,fus,sa1,sa2,sdat)) = SSPSN(cait);

SPSN(MIP(styp,sn,sdat)) = sn; SPSN(MIA(sn,sdat)) = sn;

SPSN(MAP(sn,sdat)) = sn; SPSN(MAA(sn,sdat)) = sn;

SPSN(AEA(sn,sdat)) = sn; SPSN(AE(sn,sdat)) = sn;

SPSN(RS(tkass,rtyp,sn,sdat)) = sn;

SPSN(RA(tkass,sn,sdat)) = sn; SPSN(AR(srf,actid1,sn,actid2,sdat)) = sn;

IsAS(pd) => SPSN(pd) = s(Succ(0)); (* s maps a natural number on a value of sort SSPSN *)

ofsort SPReference

SPRef(CN(spr,cait,srqms,sa1,sa2,sdat)) = spr;

SPRef(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = spr;

ofsort SAddress

CgAddr(CN(spr,cait,srqms,sa1,sa2,sdat)) = sa1;

CdAddr(CN(spr,cait,srqms,sa1,sa2,sdat)) = sa2;

CgAddr(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = sa1;

CdAddr(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = sa2;

ofsort Nat

Version(CN(spr,cait,srqms,sa1,sa2,sdat)) = Version(cait);

Version(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = Version(cait);

ofsort SResynType

ResynType(RS(tkass,rtyp,sn,sdat)) = rtyp;

ofsort TDISPar

TDISPar(RF(spr,tdpar,srqms,vs,str)) = tdpar; TDISPar(FN(tdpar,sdat)) = tdpar;

TDISPar(AB(tdpar,reas,rvp,sdat)) = tdpar;

ofsort Nat

Reason(RF(spr,tdpar,srqms,vs,str)) = ToNatNum(First(str));

Reason(AB(tdpar,reas,rvp,sdat)) = reas; Reason(Al(reas)) = reas;

Reason(AD(reas)) = reas; Reason(ED(reas,sdat)) = reas;

ofsort TSDUSize

MTSDUPar(CN(spr,cait,srqms,sa1,sa2,sdat)) = TSDUSize(cait);

MTSDUPar(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = TSDUSize(cait);

ofsort Nat

MtsSendFlow(CN(spr,cait,srqms,sa1,sa2,sdat)) = inresp(TSDUSize(cait));

MtsRecFlow(CN(spr,cait,srqms,sa1,sa2,sdat)) = respin(TSDUSize(cait));

MtsSendFlow(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = inresp(TSDUSize(cait));

MtsRecFlow(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = respin(TSDUSize(cait));

ofsort SFUs

FUs(CN(spr,cait,srqms,sa1,sa2,sdat)) = srqms; FUs(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = srqms;

FUs(RF(spr,tdpar,srqms,vs,str)) = srqms;

ofsort DatOctStr

RPV(AB(tdpar,reas,str,sdat)) = str; RPV(ER(str)) = str;

ofsort STsAss

TokenSet(CN(spr,cait,srqms,sa1,sa2,sdat)) = STsAss(cait);

TokenSet(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = STsAss(cait);

TokenSet(RS(tkass,rtyp,sn,sdat)) = tkass;

TokenSet(RA(tkass,sn,sdat)) = tkass;

InvTokenSet(CN(spr,cait,srqms,sa1,sa2,sdat)) = STsAss(Second(STsAss(cait)),First(STsAss(cait)),{});

InvTokenSet(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = STsAss(Second(STsAss(cait)),First(STsAss(cait)),{});

InvTokenSet(RS(tkass,rtyp,sn,sdat)) = STsAss(Second(tkass),First(tkass),{});

InvTokenSet(RA(tkass,sn,sdat)) = STsAss(Second(tkass),First(tkass),{});

ofsort STokens

Tokens(AC(spr,cait,stok,srqms,sa1,sa2,sdat)) = stok; Tokens(GT(stok)) = stok;

Tokens(PT(stok,sdat)) = stok;

ofsort Bool

ProtOptions(CN(*spr*,*cait*,*srqms*,*sa1*,*sa2*,*sdat*)) = *Options*(*cait*);

ProtOptions(AC(*spr*,*cait*,*stok*,*srqms*,*sa1*,*sa2*,*sdat*)) = *Options*(*cait*);

ofsort PrepType

PrepType(PR(*prt*)) = *prt*;

ofsort Enclt

Encltem(DT(*encit*,*sdat*)) = *encit*; *Encltem*(TD(*encit*,*sdat*)) = *encit*;

ofsort SData

UserInf(CN(*spr*,*cait*,*srqms*,*sa1*,*sa2*,*sdat*)) = *sdat*;

UserInf(AC(*spr*,*cait*,*stok*,*srqms*,*sa1*,*sa2*,*sdat*)) = *sdat*;

UserInf(RF(*spr*,*tdpar*,*srqms*,*vs*,*str*)) = *sdat*;

UserInf(FN(*tdpar*,*sdat*)) = *sdat*; *UserInf*(DN(*sdat*)) = *sdat*;

UserInf(NF(*sdat*)) = *sdat*; *UserInf*(AB(*tdpar*,*reas*,*rpv*,*sdat*)) = *sdat*;

UserInf(DT(*encit*,*sdat*)) = *sdat*; *UserInf*(TD(*encit*,*sdat*)) = *sdat*;

UserInf(CD(*sdat*)) = *sdat*; *UserInf*(EX(*sdat*)) = *sdat*;

UserInf(CDA(*sdat*)) = *sdat*; *UserInf*(PT(*stok*,*sdat*)) = *sdat*;

UserInf(AR(*srf*,*actid1*,*sn*,*actid2*,*sdat*)) = *sdat*;

UserInf(RS(*tkass*,*rtyp*,*sn*,*sdat*)) = *sdat*;

UserInf(RA(*tkass*,*sn*,*sdat*)) = *sdat*;

UserInf(MIP(*styp*,*sn*,*sdat*)) = *sdat*;

UserInf(MIA(*sn*,*sdat*)) = *sdat*; *UserInf*(MAP(*sn*,*sdat*)) = *sdat*;

UserInf(MAA(*sn*,*sdat*)) = *sdat*; *UserInf*(AEA(*sn*,*sdat*)) = *sdat*;

UserInf(AS(*actid1*,*sdat*)) = *sdat*; *UserInf*(AE(*sn*,*sdat*)) = *sdat*;

UserInf(ED(*reas*,*sdat*)) = *sdat*;

endtype

(*-----*)

9.1.1.4 Equality of SPDUs

ASPDUEquality enriches the ASPDU construction with boolean equality on SPDUs.

-----*)

type ASPDUEquality **is** ASPDUParame~~ter~~Selectors

opns _eq_, _ne_: ASPDU, ASPDU -> Bool

eqns forall *pd1*, *pd2*:ASPDU, *spr1*, *spr2*:SPReference, *sreq1*, *sreq2*:SFUs, *cait1*, *cait2*:CAItem, *cg1*, *cg2*, *cd2*:SAddress, *d1*, *d2*:SData, *tk1*, *tk2*:STokens, *tdpar1*, *tdpar2*:TDISPar, *vs1*, *vs2*, *reas1*, *reas2*:Nat, *rcode1*, *rcode2*, *rpv1*, *rpv2*, *err1*, *err2*:DatOctStr, *sn1*, *sn2*:SSPSN, *a1*, *a2*:STsAss, *et1*, *et2*:Enclt, *sntp1*, *sntp2*:SSyncType, *rt1*, *rt2*:SResynType, *pt1*, *pt2*:PrepType, *aid1*, *aid2*, *aid3*, *aid4*:SAcId, *scr1*, *scr2*:SCRef

ofsort Bool

pd1 ne pd2 = not(*pd1 eq pd2*);

k(pd1) ne k(pd2) => *pd1 eq pd2* = false;

CN(*spr1*,*cait1*,*sreq1*,*cg1*,*cd1*,*d1*) eq CN(*spr2*,*cait2*,*sreq2*,*cg2*,*cd2*,*d2*) = (*spr1 eq spr2*) and (*cait1 eq cait2*) and (*sreq1 eq sreq2*) and (*cg1 eq cg2*) and (*cd1 eq cd2*) and (*d1 eq d2*);

AC(*spr1*,*cait1*,*tk1*,*sreq1*,*cg1*,*cd1*,*d1*) eq AC(*spr2*,*cait2*,*tk2*,*sreq2*,*cg2*,*cd2*,*d2*) = (*spr1 eq spr2*) and (*cait1 eq cait2*) and (*tk1 eq tk2*) and (*sreq1 eq sreq2*) and (*cg1 eq cg2*) and (*cd1 eq cd2*) and (*d1 eq d2*);

RF(*spr1*,*tdpar1*,*sreq1*,*vs1*,*rcode1*) eq RF(*spr2*,*tdpar2*,*sreq2*,*vs2*,*rcode2*) = (*spr1 eq spr2*) and (*tdpar1 eq tdpar2*) and (*sreq1 eq sreq2*) and (*vs1 eq vs2*) and (*rcode1 eq rcode2*);

FN(*tdpar1*,*d1*) eq FN(*tdpar2*,*d2*) = (*tdpar1 eq tdpar2*) and (*d1 eq d2*);

DN(*d1*) eq DN(*d2*) = *d1 eq d2*;

NF(*d1*) eq NF(*d2*) = *d1 eq d2*;

AB(*tdpar1*,*reas1*,*rpv1*,*d1*) eq AB(*tdpar2*,*reas2*,*rpv2*,*d2*) = (*tdpar1 eq tdpar2*) and (*reas1 eq reas2*) and (*rpv1 eq rpv2*) and (*d1 eq d2*);

(*k(pd1) eq k(pd2)* and (*IsGTC(pd1) or IsGTA(pd1) or IsAA(pd1) or IsAIA(pd1) or IsADA(pd1)*)) => *pd1 eq pd2* = true;

DT(*et1*,*d1*) eq DT(*et2*,*d2*) = (*et1 eq et2*) and (*d1 eq d2*);

TD(*et1*,*d1*) eq TD(*et2*,*d2*) = (*et1 eq et2*) and (*d1 eq d2*);

CD(*d1*) eq CD(*d2*) = *d1 eq d2*;

$GT(tk1) \text{ eq } GT(tk2) = tk1 \text{ eq } tk2;$
 $PT(tk1,d1) \text{ eq } PT(tk2,d2) = (tk1 \text{ eq } tk2) \text{ and } (d1 \text{ eq } d2);$
 $MIP(sntp1,sn1,d1) \text{ eq } MIP(sntp2,sn2,d2) = (sntp1 \text{ eq } sntp2) \text{ and } (sn1 \text{ eq } sn2) \text{ and } (d1 \text{ eq } d2);$
 $MIA(sn1,d1) \text{ eq } MIA(sn2,d2) = (sn1 \text{ eq } sn2) \text{ and } (d1 \text{ eq } d2);$
 $MAP(sn1,d1) \text{ eq } MAP(sn2,d2) = (sn1 \text{ eq } sn2) \text{ and } (d1 \text{ eq } d2);$
 $AE(sn1,d1) \text{ eq } AE(sn2,d2) = (sn1 \text{ eq } sn2) \text{ and } (d1 \text{ eq } d2);$
 $MAA(sn1,d1) \text{ eq } MAA(sn2,d2) = (sn1 \text{ eq } sn2) \text{ and } (d1 \text{ eq } d2);$
 $AEA(sn1,d1) \text{ eq } AEA(sn2,d2) = (sn1 \text{ eq } sn2) \text{ and } (d1 \text{ eq } d2);$
 $RA(a1,sn1,d1) \text{ eq } RA(a2,sn2,d2) = (a1 \text{ eq } a2) \text{ and } (sn1 \text{ eq } sn2) \text{ and } (d1 \text{ eq } d2);$
 $RS(a1,rt1,sn1,d1) \text{ eq } RS(a2,rt2,sn2,d2) = (a1 \text{ eq } a2) \text{ and } (rt1 \text{ eq } rt2) \text{ and } (sn1 \text{ eq } sn2) \text{ and } (d1 \text{ eq } d2);$
 $PR(pt1) \text{ eq } PR(pt2) = pt1 \text{ eq } pt2;$
 $AS(aid1,d1) \text{ eq } AS(aid2,d2) = (aid1 \text{ eq } aid2) \text{ and } (d1 \text{ eq } d2);$
 $AR(scr1,aid1,sn1,aid2,d1) \text{ eq } AR(scr2,aid3,sn2,aid4,d2) = (scr1 \text{ eq } scr2) \text{ and } (aid1 \text{ eq } aid3) \text{ and } (sn1 \text{ eq } sn2) \text{ and } (aid2 \text{ eq } aid4) \text{ and } (d1 \text{ eq } d2);$
 $AI(reas1) \text{ eq } AI(reas2) = reas1 \text{ eq } reas2;$
 $AD(reas1) \text{ eq } AD(reas2) = reas1 \text{ eq } reas2;$
 $DUM(err1) \text{ eq } DUM(err2) = err1 \text{ eq } err2;$
endtype
 (*-----*)

9.1.1.5 Other functions on abstract SPDUs

Type *ASPDU* enriches the previous definition with functions to determine whether certain conditions are satisfied by a given SPDU, e.g. concerning the SPDU parameter values. One auxiliary function, *PREP_Needed*, does not have a SPDU as argument, but a SSP; it determines the need to send a PR SPDU.

-----*)

type *ASPDU* **is** *ASPDUEquality, TFlow*

opns

<i>PREP_Needed</i> : SSP -> Bool	<i>_PREP_</i> : ASPDU, SSP -> Bool
<i>DequeuingPDUs</i> : ASPDU -> Bool	<i>CorrectCN</i> : ASPDU -> Bool
<i>Matches</i> : ASPDU, ASPDU -> Bool	<i>ValidSegmenting</i> : Nat, ASPDU -> Bool
<i>CL, C, CF</i> : ASPDU -> Bool	<i>Err_Comb</i> : ASPDU, TFlow, Bool -> Bool
<i>DummyPTorGT</i> : ASPDU -> Bool	

eqns forall *ssp*:SSP, *pd*:ASPDU, *tex*:Bool, *fl*:TFlow, *encit*:EncIt, *mt*:Nat, *spr*, *spr1*, *spr2*:SPReference, *cait*, *cait1*, *cait2*:CAItem, *fus*, *fus1*, *fus2*:SFUs, *sa1*, *sa2*, *sa3*, *sa4*:SAddress, *sdat*, *sdat1*, *sdat2*:SData, *stok*:STokens

ofsort Bool

$PREP_Needed(ssp) = (IsSRSYNreq(ssp) \text{ or } IsSRSYNrsp(ssp) \text{ or } IsSACTlreq(ssp) \text{ or } IsSACTlrsp(ssp) \text{ or } IsSACTDreq(ssp) \text{ or } IsSACTDrsp(ssp) \text{ or } IsSACTersp(ssp) \text{ or } IsSSYNMarsp(ssp));$
 $not (PREP_Needed(ssp)) \Rightarrow pd \text{ PREP } ssp = false;$
 $(IsSRSYNreq(ssp) \text{ or } IsSACTlreq(ssp) \text{ or } IsSACTDreq(ssp)) \Rightarrow pd \text{ PREP } ssp = pd \text{ eq } PR(res);$
 $(IsSRSYNrsp(ssp) \text{ or } IsSACTlrsp(ssp) \text{ or } IsSACTDrsp(ssp)) \Rightarrow pd \text{ PREP } ssp = pd \text{ eq } PR(rack);$
 $(IsSACTersp(ssp) \text{ or } IsSSYNMarsp(ssp)) \Rightarrow pd \text{ PREP } ssp = pd \text{ eq } PR(mack);$
 $DequeuingPDUs(pd) = (IsAC(pd) \text{ or } IsAEA(pd) \text{ or } IsMAA(pd) \text{ or } IsRS(pd) \text{ or } IsRA(pd) \text{ or } IsAIA(pd) \text{ or } IsADA(pd) \text{ or } IsAR(pd) \text{ or } IsAS(pd) \text{ or } IsGTA(pd));$
 $DummyPTorGT(pd) = (IsPT(pd) \text{ or } IsGT(pd)) \text{ and } (\{ \} \text{ eq } Tokens(pd));$
 $not (IsDT(pd) \text{ or } IsTD(pd)) \Rightarrow CL(pd) = false;$
 $CL(DT(encit, sdat)) = (encit \text{ eq } absent) \text{ or } (encit \text{ eq } end);$
 $CL(TD(encit, sdat)) = (encit \text{ eq } absent) \text{ or } (encit \text{ eq } end);$
 (* CL: complete SSDU or last segment of a segmented SSDU. *)
 $not (IsDT(pd) \text{ or } IsTD(pd)) \Rightarrow CF(pd) = false;$
 $not (IsDT(pd) \text{ or } IsTD(pd)) \Rightarrow C(pd) = false;$
 $C(DT(encit, sdat)) = (encit \text{ eq } absent);$
 $C(TD(encit, sdat)) = (encit \text{ eq } absent);$
 (* CF: complete SSDU or first segment of a segmented SSDU; C: complete SSDU. *)
 $not (IsDT(pd) \text{ or } IsTD(pd)) \Rightarrow ValidSegmenting(mt, pd) = false;$
 $(pd \text{ eq } DT(encit, sdat)) \text{ or } (pd \text{ eq } TD(encit, sdat)) \Rightarrow ValidSegmenting(mt, pd) =$
 $((encit \text{ ne } absent) \text{ iff } (mt \text{ ne } 0)) \text{ and } ((encit \text{ ne } end) \text{ implies } (sdat \text{ ne } (<> \text{ of } SData)))$;

```

(* Err_Comb: SPDU received on the wrong transport flow. *)
Err_Comb(pd,fl,tex) = ((IsEX(pd) or IsPR(pd)) and (fl eq normal)) or
  ((IsAA(pd) or IsAB(pd)) and (tex implies (fl eq normal)))
(* CorrectCN checks whether an incoming CN SPDU satisfies clause 7.1.1. of ISO 8327. *)
not(IsCN(pd)) => CorrectCN(pd) = false;
CorrectCN(CN(spr,cait,fus,sa1,sa2,sdat)) =
  (((not(ACT IsIn fus) and ((MA IsIn fus) or (SY IsIn fus) or (RESYN IsIn fus))) implies
    (SSPSNSSynNumber(cait) ne absent)) or
    ((ACT IsIn fus) and (SSPSN(cait) ne absent)) implies
    ((MA IsIn fus) or (SY IsIn fus) or (RESYN IsIn fus)))));
(* Matches yields true if its arguments constitute a valid combination of a CN and an AC SPDU according to
  clause 7.2.1 of ISO 8327. *)
not(IsCN(pd1) and IsAC(pd2)) => Matches(pd1,pd2) = false;
Matches(CN(spr1,cait1,fus1,sa1,sa2,sdat1),AC(spr2,cait2,stok,fus2,sa3,sa4,sdat2)) =
  ((sa3 ne absent) implies (sa3 eq sa1)) and ((sa4 ne absent) implies (sa4 eq sa2)) and
  ( (SSPSN(cait2) ne absent) iff
    (not(ACT IsIn Ints(fus1,fus2)) and ((MA IsIn Ints(fus1,fus2)) or (SY IsIn Ints(fus1,fus2)) or
    (RESYN IsIn Ints(fus1,fus2)))) )
  ) and
  (stok eq Ints(stok,RqrTokens(STsAss(cait1)))) and
  not((FD IsIn fus2) and (HD IsIn fus2)) and
  ((FD IsIn fus2) implies (FD IsIn fus1)) and ((HD IsIn fus2) implies (HD IsIn fus1)) and
  (ChoiceTokens(STsAss(cait2)) eq {}) and
  ( ((mi IsIn RqrTokens(STsAss(cait2))) or (mi IsIn AcrTokens(STsAss(cait2))))
    implies (SY IsIn Ints(fus1,fus2)) ) and
  ( ((ma IsIn RqrTokens(STsAss(cait2))) or (ma IsIn AcrTokens(STsAss(cait2))))
    implies ((MA IsIn Ints(fus1,fus2)) or (ACT IsIn Ints(fus1,fus2))) ) and
  ( ((dk IsIn RqrTokens(STsAss(cait2))) or (dk IsIn AcrTokens(STsAss(cait2))))
    implies (HD IsIn Ints(fus1,fus2)) ) and
  ( ((nr IsIn RqrTokens(STsAss(cait2))) or (nr IsIn AcrTokens(STsAss(cait2))))
    implies (NR IsIn Ints(fus1,fus2)) ) and
  ( (tk IsIn RqrTokens(STsAss(cait2))) implies
    ( (tk IsIn RqrTokens(tkass1)) or (tk IsIn ChoiceTokens(STsAss(cait1))) ) ) and
  ( (tk IsIn AcrTokens(STsAss(cait2))) implies
    ( (tk IsIn AcrTokens(tkass1)) or (tk IsIn ChoiceTokens(tkass1)) ) );

endtype
(*-----*)

```

9.1.2 Concatenation of SPDUs

The basic construction of concatenated SPDUs is defined by type *BasicConcASPDUs* that consists of an actualization of the standard type *String*. The empty string of concatenated SPDUs is represented by constant *NULL*.

```

-----*)
type BasicConcASPDUs is String actualized by NaturalNumber, ASPDU, Boolean using
sortnames  ASPDU for Element      Bool for FBool      ASPDUs for String
opnnames   ASPDUs for String      NULL for <>          Number for Length
endtype
(*-----*)

```

Type *ConcASPDUs* enriches this definition with functions:

- *Empty*: yields true if the argument string is equal to *NULL*;
- *PRAinQ*: yields true if the argument string contains a PR (RESYNCHRONIZE ACK) SPDU;
- *First,...,Fourth*: yield the first,...,fourth SPDU in the given string;
- *IsXConcat*: yields true if the argument string is an extended concatenated sequence of SPDUs;
- *IsValidConc*: yields true if the argument string is a valid concatenated sequence of SPDUs;

- *IsValConc*: yields true if the argument string could be part of a valid concatenated sequence of SPDUs;
- *M_S_w_GT*: yields true if only a dummy GT SPDU can be concatenated to the argument string in order to form a valid concatenated sequence of SPDUs;
- *CAT1,CAT0*: yield true if their argument string belong in certain categories;
- *Concatenate*: appends a SPDU to a string of SPDUs; and
- *ConcatPDUs*: adds a new SPDU to a concatenated sequence in such a way that the result still forms a valid concatenation according to clause 6 of ISO 8327.

*)

type ConcASPDUs *Is BasicConcASPDUs*

opns

Empty,PRAinQ: ASPDUs -> Bool

Rest: ASPDUs -> ASPDUs

First,Second,Third,Fourth: ASPDUs -> ASPDU

IsXConcat,IsValidConc,IsValConc,M_S_w_GT: ASPDUs -> Bool

CAT1,CAT0: ASPDU -> Bool

ConcatPDUs,Concatenate: ASPDU,ASPDUs -> ASPDUs

eqns forall *ps*:ASPDUs, *pd,pd1,pd2,pd3*:ASPDU

ofsort ASPDUs

Rest(NULL) = NULL;

Rest(Concatenate(*pd,ps*)) = *ps*;

ofsort ASPDU

First(NULL) = DUM(empty) ;

Second(NULL) = DUM(empty);

Third(NULL) = DUM(empty);

Fourth(NULL) = DUM(empty);

First(Concatenate(*pd,ps*)) = *pd*;

Second(*ps*) = *First*(*Rest*(*ps*));

Third(*ps*) = *Second*(*Rest*(*ps*));

Fourth(*ps*) = *Third*(*Rest*(*ps*));

ofsort Bool

Empty(*ps*) = (*ps* eq NULL);

PRAinQ(NULL) = false;

PRAinQ(Concatenate(*pd,ps*)) = ((*pd* eq PREP(rack)) or *PRAinQ*(*ps*));

M_S_w_GT(ASPDUs(*pd*)) = ((*IsDT*(*pd*) and not(*CL*(*pd*))) or *IsRS*(*pd*) or *IsAD*(*pd*) or *IsAI*(*pd*) or *IsCD*(*pd*));

IsValConc(*pd1*+ASPDUs(*pd2*)) => *M_S_w_GT*(*pd1*+ASPDUs(*pd2*)) =

(((*IsAS*(*pd1*) or *IsAR*(*pd1*)) and (*IsAE*(*pd2*) or *IsMAP*(*pd2*))) or

((*IsAS*(*pd1*) or *IsAR*(*pd1*)) and (begin eq Encltem(*pd2*))));

(*IsValConc*(*ps*) and (Number(*ps*) eq NatNum(Dec(3)))) => *M_S_w_GT*(*ps*) = (not(*IsMIP*(*Second*(*ps*))));

(* *M_S_w_GT* yields true if its argument can only be concatenated to a GT SPDU not containing a Token Item parameter (i.e., a dummy GT SPDU). *)

(not(*IsValConc*(*ps*))) => *M_S_w_GT*(*ps*) = false;

(not(*IsValidConc*(*ps*))) => *IsXConcat*(*ps*) = false;

(*IsValidConc*(*ps*) and (Number(*ps*) gt NatNum(Dec(2)))) => *IsXConcat*(*ps*) = true;

(*IsValidConc*(*ps*) and (Number(*ps*) lt NatNum(Dec(2)))) => *IsXConcat*(*ps*) = false;

IsValidConc(*pd1*+ASPDUs(*pd2*)) => *IsXConcat*(*pd1*+ASPDUs(*pd2*)) =

(*IsGT*(*pd1*) and (*IsMIA*(*pd2*) or *IsMAA*(*pd2*) or *IsAEA*(*pd2*)));

IsValConc(ASPDUs(*pd1*)) = true;

IsValConc(NULL) = false;

(Number(*ps*) gt NatNum(Dec(3))) => *IsValConc*(*ps*) = false;

IsValConc(*pd1*+ASPDUs(*pd2*)) =

(((*IsAS*(*pd1*) or *IsAR*(*pd1*)) and (*IsMIP*(*pd2*) or *IsAE*(*pd2*) or *IsMAP*(*pd2*) or *CF*(*pd2*))) or

((*IsMIP*(*pd1*) or *IsMIA*(*pd1*) or *IsMAP*(*pd1*) or *IsMAA*(*pd1*) or *IsAE*(*pd*) or *IsAEA*(*pd*)) and *CL*(*pd2*)));

IsValConc(*pd1*+(*pd2*+ASPDUs(*pd3*))) =

((*IsAS*(*pd1*) or *IsAR*(*pd1*)) and (*IsMIP*(*pd2*) or *IsMAP*(*pd2*) or *IsAE*(*pd2*)) and *C*(*pd3*));

IsValidConc(ASPDUs(*pd*)) = *CAT1*(*pd*);

CAT1(*pd*) = (*IsCN*(*pd*) or *IsAC*(*pd*) or *IsRF*(*pd*) or *IsFN*(*pd*) or *IsDN*(*pd*) or *IsNF*(*pd*) or *IsAB*(*pd*) or *IsAA*(*pd*) or *IsGTC*(*pd*) or *IsGTA*(*pd*) or *IsTD*(*pd*) or *IsEX*(*pd*) or *IsPR*(*pd*));

CAT0(*pd*) = (*IsGT*(*pd*) or *IsPT*(*pd*));

$IsValidConc(pd1+ASPDUs(pd2)) = (\text{not}(CAT1(pd2) \text{ or } CAT0(pd2)) \text{ and } (((IsDT(pd2) \text{ and } \text{not}(CL(pd2))) \text{ or } IsRS(pd2) \text{ or } IsAD(pd2) \text{ or } IsAI(pd2) \text{ or } IsCD(pd2)) \text{ implies } (IsGT(pd1) \text{ and } DummyPTorGT(pd1))) \text{ and } ((IsRA(pd2) \text{ or } IsADA(pd2) \text{ or } IsAIA(pd2) \text{ or } IsCDA(pd2)) \text{ implies } IsPT(pd1)) \text{ and } ((IsMIA(pd2) \text{ or } IsMAA(pd2) \text{ or } IsAEA(pd2)) \text{ implies } (IsPT(pd1) \text{ or } IsGT(pd1))) \text{ and } ((CL(pd2) \text{ or } IsMIP(pd2) \text{ or } IsMAP(pd2) \text{ or } IsAS(pd2) \text{ or } IsAR(pd2) \text{ or } IsAE(pd2)) \text{ implies } IsGT(pd1))));$
 $(Number(ps) \text{ gt } NatNum(Dec(2))) \Rightarrow IsValidConc(ps) = IsGT(First(ps)) \text{ and } IsValidConc(Rest(ps)) \text{ and } (M_S_w_GT(Rest(ps)) \text{ implies } DummyPTorGT(First(ps)));$

ofsort ASPDUs

$(IsAS(pd1) \text{ or } IsAR(pd1)) \Rightarrow ConcatPDUs(pd, ASPDUs(pd1)) = Reverse(pd+ASPDUs(pd1));$
 $(IsDT(pd1) \text{ and } (IsMIP(pd) \text{ or } IsMIA(pd) \text{ or } IsMAP(pd) \text{ or } IsMAA(pd) \text{ or } IsAE(pd) \text{ or } IsAEA(pd))) \Rightarrow ConcatPDUs(pd, ASPDUs(pd1)) = pd+ASPDUs(pd1);$
 $(IsDT(pd1) \text{ and } \text{not } (IsMIP(pd) \text{ or } IsMIA(pd) \text{ or } IsMAP(pd) \text{ or } IsMAA(pd) \text{ or } IsAE(pd) \text{ or } IsAEA(pd))) \Rightarrow ConcatPDUs(pd, ASPDUs(pd1)) = NULL;$
 $ConcatPDUs(pd, NULL) = ASPDUs(pd);$
 $(IsMIP(pd1) \text{ or } IsMIA(pd1) \text{ or } IsMAP(pd1) \text{ or } IsMAA(pd1) \text{ or } IsAE(pd1) \text{ or } IsAEA(pd1)) \Rightarrow ConcatPDUs(pd1, ASPDUs(pd1)) = NULL;$
 $((IsAR(pd1) \text{ or } IsAS(pd1)) \text{ and } IsDT(pd2) \text{ and } (IsMIP(pd) \text{ or } IsMIA(pd) \text{ or } IsMAP(pd) \text{ or } IsMAA(pd) \text{ or } IsAE(pd) \text{ or } IsAEA(pd))) \Rightarrow ConcatPDUs(pd, pd1+ASPDUs(pd2)) = pd1+(pd+ASPDUs(pd2));$
 $(\text{not } ((IsAR(pd1) \text{ or } IsAS(pd1)) \text{ and } IsDT(pd2) \text{ and } (IsMIP(pd) \text{ or } IsMIA(pd) \text{ or } IsMAP(pd) \text{ or } IsMAA(pd) \text{ or } IsAE(pd) \text{ or } IsAEA(pd)))) \Rightarrow ConcatPDUs(pd, pd1+ASPDUs(pd2)) = NULL$

endtype

(*-----*)

9.1.3 SPDU parameters**9.1.3.1 Connection Identifier parameter group**

The Connection Identifier parameter group, contained in CN, AC and RF SPDUs, is defined by type *SPReference*. It consists of an actualization of the auxiliary type *Threetuple* (see 9.4.6).

-----*)
type SPReference is Threetuple actualizedby *DatOctStr1, Boolean using*
sortnames *Bool for FBool* *DatOctStr for Element1*
DatOctStr for Element2 *DatOctStr for Element3* *SPReference for Threetuple*
opnnames *SPReference for Threetuple* *CgUserReference for First*
CommonReference for Second *AdditionalRef for Third*
endtype
 (*-----*)

9.1.3.2 Connect/Accept Item parameter group

The Connect/Accept Item parameter group, contained in CN an AC SPDUs, is defined by type *CAItem*. It consists of an actualization of the auxiliary type *FiveTuple* (see 9.4.6). *SSynchronizationNumber* and *STokensAssignment* are imported from ISO/IEC/TR 9571.

-----*)
type CAItem is Fivetuple actualizedby *TSDUSize, SSynchronizationNumber, STokensAssignment using*
sortnames *Bool for FBool* *Bool for Element1* *TSDUSize for Element2*
Nat for Element3 *SSPSN for Element4* *STsAss for Element5* *CAItem for Fivetuple*
opnnames *Options for First* *CAItem for Fivetuple* *TSDUSize for Second*
Version for Third *SSPSN for Fourth* *STsAss for Fifth*
endtype
 (*-----*)

9.1.3.3 TSDU Maximum Size parameter

The TSDU Maximum Size parameter, contained in CN and AC SPDUs, is defined by type *TSDUSize*, an enrichment of type *TSDUSize0*. The latter type consists of an actualization of auxiliary type *TwoTuple* (see 9.4.6). Function *inresp* is used to determine the TSDU Maximum Size value for the direction from initiator to responder, *respin* is used to determine the value for the opposite direction. Value 0 indicates that segmenting is not allowed for the direction of concern. The constant *absent*, introduced by type *TSDUSize*, indicates that segmenting is not allowed for either direction.

-----*)

type *TSDUSize* **is** *TSDUSize0*

opns *absent*: -> *TSDUSize* *_eq_*: *TSDUSize*,*TSDUSize* -> *Bool*

eqns ofsort *Nat* *inresp*(*absent*) = 0; *respin*(*absent*) = 0

endtype

type *TSDUSize0* **is** *Twotuple* **actualizedby** *Boolean*,*NaturalNumber* **using**

sortnames

<i>Bool</i> for <i>FBool</i>	<i>Nat</i> for <i>Element1</i>	<i>Nat</i> for <i>Element2</i>	<i>TSDUSize</i> for <i>Twotuple</i>
opnnames	<i>TSDUSize</i> for <i>Twotuple</i>	<i>inresp</i> for <i>First</i>	<i>respin</i> for <i>Second</i>

endtype

(*-----)

9.1.3.4 Transport Disconnect parameter

The Transport Disconnect parameter, contained in RF, FN and AB SPDUs, is defined by type *TDISPar*. It consists of a renaming of auxiliary type *Doublet*, imported from ISO/IEC/TR 9571. Constant *nr* indicates that the sender of the SPDU requests termination of the TC; constant *r* indicates that re-use of the TC is requested.

-----*)

type *TDISPar* **is** *Doublet* **renamedby**

sortnames *TDISPar* **for** *Doublet*

opnnames *nr* **for** *constant1* *r* **for** *constant2*

endtype

(*-----)

9.1.3.5 Prepare Type parameter

The Prepare Type parameter, contained in PR SPDUs, is defined by type *Prepare*. It consists of a renaming of type *Triplet*, imported from ISO/IEC/TR 9571. The three constants *res*, *rack* and *mack*, respectively, correspond with the values "RESYNCHRONIZE", "RESYNCHRONIZE ACK" and "MAJOR SYNC ACK" defined in table 10 of ISO 8327.

-----*)

type *Prepare* **is** *Triplet* **renamedby**

sortnames *PrepType* **for** *Triplet*

opnnames *res* **for** *constant1* *rack* **for** *constant2* *mack* **for** *constant3*

endtype

(*-----)

9.1.3.6 Enclosure Item parameter

The Enclosure Item parameter, contained in DT and TD SPDUs, is defined by type *EnclItem*. It consists of a renaming of type *Quartet*, imported from ISO/IEC/TR 9571. Constant *begin* is used to indicate that the SPDU is the first segment of a segmented SSDU, *middle* that the SPDU is neither the first nor the last

segment, *end* that the SPDU is the last segment of a segmented SSDU, and *absent* that segmenting is not allowed.

```

-----*)
type EnclItem is Quartet renamedby
sortnames Enclt for Quartet
opnnames
begin for constant1      middle for constant2      end for constant3      absent for constant4
endtype
(*)-----

```

9.1.4 Encoded SPDU

9.1.4.1 SI field

Valid values of the SI field are defined by type *PDUSIs*. It imports type *PDUSIs0* that actualizes the standard type *Set*. Constant *ValidSIs* represents the set containing all valid SI values.

```

-----*)
type PDUSIs is PDUSIs0
opns ValidSIs: -> Siset
eqns
ValidSIs = Insert(Dec(0), Insert(Dec(1), Insert(Dec(2), Insert(Dec(5), Insert(Dec(7), Insert(Dec(8),
Insert(Dec(9), Insert(1+Dec(0), Insert(1+Dec(2), Insert(1+Dec(3), Insert(1+Dec(4), Insert(2+Dec(1),
Insert(2+Dec(2), Insert(2+Dec(5), Insert(2+Dec(6), Insert(2+Dec(9), Insert(3+Dec(3), Insert(3+Dec(4),
Insert(4+Dec(1), Insert(4+Dec(2), Insert(4+Dec(5), Insert(4+Dec(8), Insert(4+Dec(9), Insert(5+Dec(0),
Insert(5+Dec(3), Insert(5+Dec(7), Insert(5+Dec(8), Insert(6+Dec(1),
Insert(6+Dec(2),{))))))))))))))))))))))))))
endtype
type PDUSIs0 is Set actualizedby DecNatRepr, NaturalNumber using
sortnames DecString for Element      Bool for FBool      Siset for Set
endtype
(*)-----

```

9.1.4.2 Auxiliary functions for SPDU valid encoding checks

Type *Err3Tuple* defines values to represent intermediate tests on the valid encoding of SPDUs. It imports type *Err3Tuple0*, an actualization of the auxiliary type *ThreeTuple* (see 9.4.6). The values are 3-tuples, where the second and third component are octet strings representing, respectively, the part of an encoding not yet tested and the part that is already tested, and the first component is a boolean indicating the result of the test so far. The function *eq* tests equality of two such 3-tuples (all three components must be equal).

```

-----*)
type Err3Tuple is Err3Tuple0
opns _eq_ : Err3Tuple, Err3Tuple -> Bool
eqns forall s1,s2,s3,s4:DatOctStr, b1,b2:Bool ofsort Bool
Err3Tuple(b1,s1,s2) eq Err3Tuple(b2,s3,s4) = ( (b1 eq b2) and (s1 eq s3) and (s2 eq s4) );
endtype
type Err3Tuple0 is Threetuple actualizedby DatOctStr1 using
sortnames      Bool for FBool      Bool for Element1      Err3Tuple for ThreeTuple
DatOctStr for Element2      DatOctStr for Element3
opnnames      Err3Tuple for ThreeTuple
endtype
(*)-----

```

9.1.4.3 Valid encoding checks

Type *ErrCoding* defines functions that check whether a given octet string constitutes a valid encoding of a SPDU. *ErrCodStr* yields an empty octet string if the argument octet string represents a correctly encoded SPDU, otherwise the erroneous part of the argument string is returned. The first octet of a valid encoding should represent a valid SI value (an "unknown" value is returned by *ErrCodStr*). If a valid SI value is present, the remaining part of the octet string should be an encoding of the SPDU class identified by the SI value; this is checked with *ErrCodXX* (XX being the denotation of the SPDU class, see 9.1.1.2). The valid encodings of the parameters (parameter groups) from which the SPDU encoding is constructed are again tested by separate functions, one for each parameter (group). The definition of the latter functions is based on a limited number of auxiliary functions.

-----*)

type *ErrCoding* **is** *DatOctStr2, PDUSIs, Compare, Err3Tuple, BitString*

opns

ErrCodStr: *DatOctStr* -> *DatOctStr*

ErrCodCN, ErrCodRF, ErrCodFN, ErrCodDNorNF, ErrCodABorAI, ErrCodEX, ErrCodTD, ErrCodGT, ErrCodCDorCDA, ErrCodPT, ErrCodMIA, ErrCodMAAorAEA, ErrCodRS, ErrCodRA, ErrCodPR, ErrCodAS, ErrCodAR, ErrCodAD, ErrCodAEorMAPorMIP, ErrCodAC, ErrCodNF: *DatOctStr, DatOctStr* -> *DatOctStr*

CodCN1, CodCN2, CodCN3: *DatOctStr, DatOctStr, DatOctStr* -> *DatOctStr*

CodCNLen: *DatOctStr, DatOctStr* -> *DatOctStr*

CorrConId: *DatOctStr, Octet* -> *Err3Tuple*

CorrConId1: *DatOctStr, DatOctStr, DatOctStr, Octet* -> *Err3Tuple*

CorrConId2, CorrConId3: *DatOctStr, DatOctStr, DatOctStr* -> *Err3Tuple*

CorrCAIt, CorrRest: *DatOctStr* -> *Err3Tuple*

CorrCAIt1, CorrCAIt2, CorrCAIt3, CorrCAIt4, CorrCAIt5: *DatOctStr, DatOctStr, DatOctStr* -> *Err3Tuple*

CorrRest2, CorrRest3, CorrRest4: *DatOctStr, DatOctStr* -> *Err3Tuple*

CorrParML: *DecString, DecString, DatOctStr, DatOctStr* -> *Err3Tuple*

CorrPar, CorrParm, MCorrPar, MCorrParm: *DecString, DecString, DatOctStr* -> *Err3Tuple*

CheckBits: *DecString, DatOctStr* -> *DatOctStr*

No11: *BitString* -> *Bool*

SPSNOctets: *DatOctStr* -> *DatOctStr*

SPSNOct2: *DatOctStr, DatOctStr* -> *DatOctStr*

eqns forall *o, o1, o2, o3: Octet, ds, pgi, pi, len, li: DecString, b: Bool,*

st, s1, s2, s3, s4, s5: DatOctStr, bstr, bstr1, bstr2: BitString, b1, b2: Bit, dv1, dv2, dv3, dv4: DecString

ofsort *DatOctStr*

(*o1 eq ds* and not (*ds IsIn ValidSIs*)) => *ErrCodStr(o1+s1) = Octet(o1)*;

(* The SI code is not in the list of known SI codes so only this octet is returned as being the erroneous part of the coded ASPDU. *)

(*o1 eq (1+Dec(3))*) => *ErrCodStr(o1+s1) = ErrCodCN(s1, Octet(o1))*;

(*o1 eq (1+Dec(4))*) => *ErrCodStr(o1+s1) = ErrCodAC(s1, Octet(o1))*;

(*o1 eq (2+Dec(5))*) => *ErrCodStr(o1+s1) = ErrCodABorAI(s1, Octet(o1))*;

((*o1 eq (4+Dec(1))*) or (*o1 eq (4+Dec(9))*)) => *ErrCodStr(o1+s1) = ErrCodAEorMAPorMIP(s1, Octet(o1))*;

(*o1 eq (1+Dec(2))*) => *ErrCodStr(o1+s1) = ErrCodRF(s1, Octet(o1))*;

(*o1 eq Dec(9)*) => *ErrCodStr(o1+s1) = ErrCodFN(s1, Octet(o1))*;

(*o1 eq (1+Dec(0))* or (*o1 eq Dec(8)*)) => *ErrCodStr(o1+s1) = ErrCodDNorNF(s1, Octet(o1))*;

(*o1 eq Dec(8)*) => *ErrCodStr(o1+s1) = ErrCodNF(s1, Octet(o1))*;

(*o1 eq (3+Dec(3))*) => *ErrCodStr(o1+s1) = ErrCodTD(s1, Octet(o1))*;

(*o1 eq Dec(1)*) => *ErrCodStr(o1+s1) = ErrCodGT(s1, Octet(o1))*;

(*o1 eq Dec(5)*) => *ErrCodStr(o1+s1) = ErrCodEX(s1, Octet(o1))*;

((*o1 eq (6+Dec(1))*) or (*o1 eq (6+Dec(2))*)) => *ErrCodStr(o1+s1) = ErrCodCDorCDA(s1, Octet(o1))*;

(*o1 eq Dec(2)*) => *ErrCodStr(o1+s1) = ErrCodPT(s1, Octet(o1))*;

(*o1 eq (5+Dec(0))*) => *ErrCodStr(o1+s1) = ErrCodMIA(s1, Octet(o1))*;

(*o1 eq (4+Dec(2))*) => *ErrCodStr(o1+s1) = ErrCodMAAorAEA(s1, Octet(o1))*;

(*o1 eq (5+Dec(3))*) => *ErrCodStr(o1+s1) = ErrCodRS(s1, Octet(o1))*;

(*o1 eq (3+Dec(4))*) => *ErrCodStr(o1+s1) = ErrCodRA(s1, Octet(o1))*;

(*o1 eq Dec(7)*) => *ErrCodStr(o1+s1) = ErrCodPR(s1, Octet(o1))*;

```

(o1 eq (4+Dec(5))) => ErrCodStr(o1+s1) = ErrCodAS(s1,Octet(o1));
(o1 eq (2+Dec(9))) => ErrCodStr(o1+s1) = ErrCodAR(s1,Octet(o1));
(o1 eq (5+Dec(7))) => ErrCodStr(o1+s1) = ErrCodAD(s1,Octet(o1));
((o1 eq (2+Dec(6))) or (o1 eq (2+Dec(1))) or (o1 eq (2+Dec(2))) or (o1 eq (5+Dec(8))) and (o2 eq Dec(0)))
=> ErrCodStr(o1+(o2+s1)) = ErrCodStr(s1);
((o1 eq (2+Dec(6))) or (o1 eq (2+Dec(1))) or (o1 eq (2+Dec(2))) or (o1 eq (5+Dec(8))) and (o2 ne Dec(0)))
=> ErrCodStr(o1+(o2+s1)) = o1+Octet(o2);

```

(* CN SPDU valid encoding checks: Each parameter (group) is analysed separately to determine whether the encoding is according to the rules given in clause 8 of ISO 8327. The following is checked: whether the LI (length indicator) fields have the correct values, whether the parameters (parameter groups) are in the correct order, whether maximum length restrictions have not been violated and whether the individual parameter values have been correctly encoded. *)

```

Empty(s1) => ErrCodCN(s1,s2) = s2;
(* The length indicator is absent. The first octet is returned as the erroneous string. *)
(o1 eq Dec(0)) => ErrCodCN(o1+s3,s2) = ErrCodStr(s3);
(* The length indicator contains the value zero, so there are no parameters present. *)

```

```

((o1 lt (2+(5+Dec(5)))) and (Length(s3) lt o1)) => ErrCodCN(o1+s3,s2) = s2++s1;
(* The length of the remaining string is smaller than indicated by the LI value; the whole octet string is
returned as being erroneous. *)

```

```

((o1 lt (2+(5+Dec(5)))) and (Length(s3) eq o1)) => ErrCodCN(o1+s3++s4,s2) = CodCN1(s3,s2--o1,s4);
(o1 eq (2+(5+Dec(5)))) => ErrCodCN(o1+s3,s2) = CodCNLen(s3,s2--o1);

```

(* Two cases are distinguished in the following: 1) the LI value indicates a length smaller than 255 octets, in which case the LI field is encoded using one octet, and 2) the first octet represents the value 255, in which case the encoding of the LI field should consist of three octets. The actual LI value is then encoded in the remaining two octets and it should be greater than or equal to 255, otherwise the encoding of LI is considered to be erroneous. Furthermore, the length of any correctly encoded CN SPDU is implicitly constrained by a maximum value equal to 719. This is also checked. *)

```

(Length(s1) lt NatNum(Dec(2))) => CodCNLen(s1,s2) = s2++s1;
((NatNum(BStr(o1)++BStr(o2)) gt (7+(1+Dec(9)))) or (NatNum(BStr(o1)++BStr(o2)) lt (2+(5+Dec(5)))) ) =>
CodCNLen(o1+(o2+s3),s2) = s2++(o1+Octet(o2));
((Length(s3) lt NatNum(BStr(o1)++BStr(o2))) and (NatNum(BStr(o1)++BStr(o2)) le (7+(1+Dec(9)))) and
(NatNum(BStr(o1)++BStr(o2)) ge (2+(5+Dec(5)))) ) =>
CodCNLen(o1+(o2+s3),s2) = s2++s1;
(* The LI field is correctly encoded but the remainder of the string is too short. *)

```

```

((Length(s3) eq NatNum(BStr(o1)++BStr(o2))) and (NatNum(BStr(o1)++BStr(o2)) le (7+(1+Dec(9)))) and
(NatNum(BStr(o1)++BStr(o2)) ge (2+(5+Dec(5)))) ) =>
CodCNLen(o1+(o2+s3++s4),s2) = CodCN1(s3,s2++(o1+Octet(o2)),s4);
(* LI and length restrictions are satisfied: next, the parameter groups are checked, starting with the
Connection Identifier parameter group. *)

```

```

First(CorrConld(s1,First(s2))) => CodCN1(s1,s2,s3) =
CodCN2(Third(CorrConld(s1,First(s2))),s2++Second(CorrConld(s1,First(s2))),s3);
(* CorrConld returns a boolean value and two octet strings. The boolean is true if no error was "detected" in
the encoding. In that case, the second string is the part found to be correct so far and the third string is the
part still to be checked. *)

```

```

not(First(CorrConld(s1,First(s2)))) => CodCN1(s1,s2,s3) = s2++Second(CorrConld(s1,First(s2)));
First(CorrCAIt(s1)) => CodCN2(s1,s2,s3) = CodCN3(Third(CorrCAIt(s1)),s2++Second(CorrCAIt(s1)),s3);
(* The Connect/Accept Item parameter group is not mandatory. If a parameter (group) is not mandatory, an
empty octet string is considered to be a valid encoding. *)

```

```

not(First(CorrCAIt(s1))) => CodCN2(s1,s2,s3) = s2++Second(CorrCAIt(s1));
First(CorrRest(s1)) => CodCN3(s1,s2,s3) = ErrCodStr(s3);

```

not(First(CorrRest(s1))) => CodCN3(s1,s2,s3) = s2++Second(CorrRest(s1));
 (* CorrRest takes care of the remaining parameters of the CN SPDU. *)

ofsort Err3Tuple

Empty(s1) => CorrConld(s1,o) = Err3Tuple(true,s1,s1);
 (* No parameters (parameter groups) are present if s1 is empty. Only the octet representing the SI value is present. This octet is denoted by o. *)

(o1 ne Dec(1)) => CorrConld(o1+s2,o) = Err3Tuple(true,empty,o1+s2);
 (* The encoding of the Connection Identifier parameter group is absent. Note that this parameter group is not mandatory. *)

((o1 eq Dec(1)) and Empty(s2)) => CorrConld(o1+s2,o) = Err3Tuple(false,Octet(o1),s2);
((o1 eq Dec(1)) and (o2 gt (1+(3+Dec(2))))) =>
CorrConld(o1+(o2+s2),o) = Err3Tuple(false,o1+Octet(o2),s2);
 (* The value contained in the LI field is too large. *)

((o1 eq Dec(1)) and (o2 le (1+(3+Dec(2)))) and (Length(s2) lt o2)) =>
CorrConld(o1+(o2+s2),o) = Err3Tuple(false,o1+(o2+s2),empty);
 (* The LI field contains a valid value but the remaining string is too short. *)

((o1 eq Dec(1)) and (o2 le (1+(3+Dec(2)))) and (Length(s2) eq o2)) =>
CorrConld(o1+(o2+s2++s3),o) = CorrConld1(s2,o1+Octet(o2),s3,o);

(* Individual parameters are checked below. This check is performed by the function CorrParm whose arguments represent the PI field, the maximum length of the parameter value, and the parameter encoding to be checked. For example, the PI value 10 identifies the Calling SS-user Reference parameter of the CN SPDU, and PI value 9 identifies the Called SS-user Reference parameter of an AC or RF SPDU. The maximum length of these parameters is 64 octets. CorrParm is used to check the correctness of variable, bounded, lengths of non-mandatory parameters. *)

(First(CorrParm(1+Dec(0),6+Dec(4),s1)) and (o eq (1+Dec(3)))) => CorrConld1(s1,s2,s3,o) =
CorrConld2(Third(CorrParm(1+Dec(0),6+Dec(4),s1)),
s2++Second(CorrParm(1+Dec(0),6+Dec(4),s1)), s3);
(First(CorrParm(Dec(9),6+Dec(4),s1)) and (o ne (1+Dec(3)))) => CorrConld1(s1,s2,s3,o) =
CorrConld2(Third(CorrParm(Dec(9),6+Dec(4),s1)), s2++Second(CorrParm(Dec(9),6+Dec(4),s1)), s3);

(not(First(CorrParm(1+Dec(0),6+Dec(4),s1))) and (o eq (1+Dec(3)))) =>
CorrConld1(s1,s2,s3,o) = Err3Tuple(false,s2++Second(CorrParm(1+Dec(0),6+Dec(4),s1)),s3);
(not(First(CorrParm(Dec(9),6+Dec(4),s1))) and (o ne (1+Dec(3)))) =>
CorrConld1(s1,s2,s3,o) = Err3Tuple(false,s2++Second(CorrParm(Dec(9),6+Dec(4),s1)),s3);

First(CorrParm(1+Dec(1),6+Dec(4),s1)) => CorrConld2(s1,s2,s3) =
CorrConld3(Third(CorrParm(1+Dec(1),6+Dec(4),s1)),
s2++Second(CorrParm(1+Dec(1),6+Dec(4),s1)), s3);
not(First(CorrParm(1+Dec(1),6+Dec(4),s1))) => CorrConld2(s1,s2,s3) =
Err3Tuple(false,s2++Second(CorrParm(1+Dec(1),6+Dec(4),s1)),s3);
(First(CorrParm(1+Dec(2),Dec(4),s1)) and Empty(Third(CorrParm(1+Dec(2),Dec(4),s1)))) =>
CorrConld3(s1,s2,s3) = Err3Tuple(true,s2++Second(CorrParm(1+Dec(2),Dec(4),s1)),s3);
 (* The Connection Identifier parameter group is found to be correctly encoded if there are no octets left for checking and the length indicated in the LI field corresponds to the length of the octet string representing the encoding. *)

(First(CorrParm(1+Dec(2),Dec(4),s1)) and not(Empty(Third(CorrParm(1+Dec(2),Dec(4),s1))))) =>
CorrConld3(s1,s2,s3) = Err3Tuple(false, s2++Second(CorrParm(1+Dec(2),Dec(4),s1))++
Octet(First(Third(CorrParm(1+Dec(2),Dec(4),s1)))), s3);
not(First(CorrParm(1+Dec(2),Dec(4),s1))) =>
CorrConld3(s1,s2,s3) = Err3Tuple(false,s2++Second(CorrParm(1+Dec(2),Dec(4),s1)),s3);

(* CorrCAIt, below, checks the encoding of the Connect/Accept Item parameter group. This check is analogous to that presented above. Additional auxiliary functions are defined similar to CorrParm. These are: MCorrParm (mandatory, bounded-variable length), MCorrPar (mandatory, fixed-length) and CorrPar (non-mandatory, fixed-length). Note that in certain parameter value encodings some bits are "reserved" or set to certain specific values. This implies that additional checks are involved which are taken care of by function CheckBits. *)

```
Empty(s1) => CorrCAIt(s1) = Err3Tuple(true,s1,s1);
(o1 ne Dec(5)) => CorrCAIt(o1+s2) = Err3Tuple(true,empty,o1+s2);
( (o1 eq Dec(5)) and Empty(s2) ) => CorrCAIt(o1+s2) = Err3Tuple(false,Octet(o1),s2);
( (o1 eq Dec(5)) and (o2 gt (1+Dec(3))) ) =>
    CorrCAIt(o1+(o2+s2)) = Err3Tuple(false,o1+Octet(o2),s2);
( (o1 eq Dec(5)) and (o2 le (1+Dec(3))) and (Length(s2) lt o2) ) =>
    CorrCAIt(o1+(o2+s2)) = Err3Tuple(false,o1+(o2+s2),empty);
( (o1 eq Dec(5)) and (o2 le (1+Dec(3))) and (Length(s2) eq o2) ) =>
    CorrCAIt(o1+(o2+s2++s3)) = CorrCAIt1(s2,o1+Octet(o2),s3);
First(MCorrPar(1+Dec(9),Dec(1),s1)) => CorrCAIt1(s1,s2,s3) =
    CorrCAIt2(Third(MCorrPar(1+Dec(9),Dec(1),s1)),s2++Second(MCorrPar(1+Dec(9),Dec(1),s1)),s3);
not(First(MCorrPar(1+Dec(9),Dec(1),s1))) =>
    CorrCAIt1(s1,s2,s3) = Err3Tuple(false,s2++Second(MCorrPar(1+Dec(9),Dec(1),s1)),s3);
First(CorrPar(2+Dec(1),Dec(4),s1)) => CorrCAIt2(s1,s2,s3) =
    CorrCAIt3(Third(CorrPar(2+Dec(1),Dec(4),s1)),s2++Second(CorrPar(2+Dec(1),Dec(4),s1)),s3);
not(First(CorrPar(2+Dec(1),Dec(4),s1))) =>
    CorrCAIt2(s1,s2,s3) = Err3Tuple(false,s2++Second(CorrPar(2+Dec(1),Dec(4),s1)),s3);
First(MCorrPar(2+Dec(2),Dec(1),s1)) => CorrCAIt3(s1,s2,s3) =
    CorrCAIt4(Third(MCorrPar(2+Dec(2),Dec(1),s1)),s2++Second(MCorrPar(2+Dec(2),Dec(1),s1)),s3);
not(First(MCorrPar(2+Dec(2),Dec(1),s1))) =>
    CorrCAIt3(s1,s2,s3) = Err3Tuple(false,s2++Second(MCorrPar(2+Dec(2),Dec(1),s1)),s3);
First(CorrParm(2+Dec(3),Dec(6),s1)) => CorrCAIt4(s1,s2,s3) =
    CorrCAIt5(Third(CorrParm(2+Dec(3),Dec(6),s1)),s2++Second(CorrParm(2+Dec(3),Dec(6),s1)),s3);
not(First(CorrParm(2+Dec(3),Dec(6),s1))) =>
    CorrCAIt4(s1,s2,s3) = Err3Tuple(false,s2++Second(CorrParm(2+Dec(3),Dec(6),s1)),s3);
( First(CorrParm(2+Dec(6),Dec(1),s1)) and Empty(Third(CorrParm(2+Dec(6),Dec(1),s1))) ) =>
    CorrCAIt5(s1,s2,s3) = Err3Tuple(true,s2++Second(CorrParm(2+Dec(6),Dec(1),s1)),s3);
( First(CorrParm(2+Dec(6),Dec(1),s1)) and not(Empty(Third(CorrParm(2+Dec(6),Dec(1),s1)))) ) =>
    CorrCAIt5(s1,s2,s3) = Err3Tuple(false,s2++Second(CorrParm(2+Dec(6),Dec(1),s1))+
        Octet(First(Third(CorrParm(2+Dec(6),Dec(1),s1))))),s3);
not(First(CorrParm(2+Dec(6),Dec(1),s1))) =>
    CorrCAIt5(s1,s2,s3) = Err3Tuple(false,s2++Second(CorrParm(2+Dec(6),Dec(1),s1)),s3);
```

(* CorrRest checks whether the Session Requirements parameter, the Session Address parameters and the SS-user Data parameter have been correctly encoded. *)

```
First(CorrPar(2+Dec(0),Dec(2),s1)) => CorrRest(s1) =
    CorrRest2(Third(CorrPar(2+Dec(0),Dec(2),s1)),Second(CorrPar(2+Dec(0),Dec(2),s1)));
not(First(CorrPar(2+Dec(0),Dec(2),s1))) => CorrRest(s1) =
    Err3Tuple(false,Second(CorrPar(2+Dec(0),Dec(2),s1)),Third(CorrPar(2+Dec(0),Dec(2),s1)));
First(CorrParm(5+Dec(1),1+Dec(6),s1)) => CorrRest2(s1,s4) =
    CorrRest3(Third(CorrParm(5+Dec(1),1+Dec(6),s1)),Second(CorrParm(5+Dec(1),1+Dec(6),s1))+s4);
not(First(CorrParm(5+Dec(1),1+Dec(6),s1))) => CorrRest2(s1,s4) =
    Err3Tuple (false, Second(CorrParm(5+Dec(1),1+Dec(6),s1))+s4,
        Third(CorrParm(5+Dec(1),1+Dec(6),s1)));
First(CorrParm(5+Dec(2),1+Dec(6),s1)) =>
    CorrRest3(s1,s4) = CorrRest4(s3,Second(CorrParm(5+Dec(2),1+Dec(6),s1))+s4);
not(First(CorrParm(5+Dec(2),1+Dec(6),s1))) => CorrRest3(s1,s4) =
    Err3Tuple (false, Second(CorrParm(5+Dec(2),1+Dec(6),s1))+s4,
        Third(CorrParm(5+Dec(2),1+Dec(6),s1)));
( First(CorrParm(1+(9+Dec(3)),5+(1+Dec(2)),s1)) and
    Empty(Third(CorrParm(1+(9+Dec(3)),5+(1+Dec(2)),s1))) ) => CorrRest4(s1,s4) =
    Err3Tuple(true,Second(CorrParm(1+(9+Dec(3)),5+(1+Dec(2)),s1))+s4,empty);
```

```
( First(CorrParm(1+(9+Dec(3)),5+(1+Dec(2)),s1)) and
  not(Empty(Third(CorrParm(1+(9+Dec(3)),5+(1+Dec(2)),s1)))) ) => CorrRest4(s1,s4) =
  Err3Tuple (false, s2++Octet(First(Third(CorrParm(1+(9+Dec(3)),5+(1+Dec(2)),s1))))++s4,
    Third(CorrParm(1+(9+Dec(3)),5+(1+Dec(2)),s1)));
not(First(CorrParm(1+(9+Dec(3)),5+(1+Dec(2)),s1)) => CorrRest4(s1,s4) =
  Err3Tuple(false,Second(CorrParm(1+(9+Dec(3)),5+(1+Dec(2)),s1))++s4,
    Third(CorrParm(1+(9+Dec(3)),5+(1+Dec(2)),s1)));
```

(* AC SPDU valid encoding checks: *)

ofsort DatOctStr

```
Empty(s1) => ErrCodAC(s1,s2) = s2;
(o1 eq Dec(0)) => ErrCodAC(o1+s3,s2) = ErrCodStr(s3);
( (o1 lt (2+(5+Dec(5)))) and (Length(s3) lt o1) ) => ErrCodAC(o1+s3,s2) = s2++s1;
( (o1 lt (2+(5+Dec(5)))) and (Length(s3) eq o1) ) => ErrCodAC(o1+s3++s4,s2) = CodAC1(s3,s2--o1,s4);
(o1 eq (2+(5+Dec(5)))) => ErrCodAC(o1+s3,s2) = CodACLen(s3,s2--o1);
(Length(s1) lt NatNum(Dec(2))) => CodACLen(s1,s2) = s2++s1;
( (NatNum(BStr(o1))++BStr(o2)) gt (7+(2+Dec(2)))) or (NatNum(BStr(o1))++BStr(o2)) lt (2+(5+Dec(5)))) ) =>
  CodACLen(o1+(o2+s3),s2) = s2++(o1+Octet(o2));
( (Length(s3) lt NatNum(BStr(o1))++BStr(o2))) and (NatNum(BStr(o1))++BStr(o2)) le (7+(2+Dec(2)))) and
  (NatNum(BStr(o1))++BStr(o2)) ge (2+(5+Dec(5)))) ) =>
  CodACLen(o1+(o2+s3),s2) = s2++s1;
( (Length(s3) eq NatNum(BStr(o1))++BStr(o2))) and (NatNum(BStr(o1))++BStr(o2)) le (7+(2+Dec(2)))) and
  (NatNum(BStr(o1))++BStr(o2)) ge (2+(5+Dec(5)))) ) =>
  CodACLen(o1+(o2+s3++s4),s2) = CodAC1(s3,s2++(o1+Octet(o2)),s4);
First(CorrConld(s1,First(s2))) => CodAC1(s1,s2,s3) =
  CodAC2(Third(CorrConld(s1,First(s2))),s2++Second(CorrConld(s1,First(s2))),s3);
not(First(CorrConld(s1,First(s2)))) => CodAC1(s1,s2,s3) = s2++Second(CorrConld(s1,First(s2)));
First(CorrCAIt(s1)) => CodAC2(s1,s2,s3) = CodAC3(Third(CorrCAIt(s1)),s2++Second(CorrCAIt(s1)),s3);
not(First(CorrCAIt(s1))) => CodAC2(s1,s2,s3) = s2++Second(CorrCAIt(s1));
First(CorrTokenItem(s1)) =>
  CodAC3(s1,s2,s3) = CodAC4(Third(CorrTokenItem(s1)),s2++Second(CorrTokenItem(s1)),s3);
not(First(CorrTokenItem(s1))) => CodAC3(s1,s2,s3) = s2++Second(CorrTokenItem(s1));
First(CorrRest(s1)) => CodAC4(s1,s2,s3) = ErrCodStr(s3);
not(First(CorrRest(s1))) => CodAC4(s1,s2,s3) = s2++Second(CorrRest(s1));
```

ofsort Err3Tuple

```
CorrTokenItem(s1) =
Err3Tuple( First(CorrPar(1+Dec(6),Dec(1),s1)), Second(CorrPar(1+Dec(6),Dec(1),s1)),
  Third(CorrPar(1+Dec(6),Dec(1),s1)) );
```

(* RF SPDU valid encoding checks: *)

ofsort DatOctStr

```
Empty(s1) => ErrCodRF(s1,s2) = s2;
(o1 eq Dec(0)) => ErrCodRF(o1+s3,s2) = ErrCodStr(s3);
( (o1 lt (2+(5+Dec(5)))) and (Length(s3) lt o1) ) => ErrCodRF(o1+s3,s2) = s2++s1;
( (o1 lt (2+(5+Dec(5)))) and (Length(s3) eq o1) ) => ErrCodRF(o1+s3++s4,s2) = CodRF1(s3,s2--o1,s4);
(o1 eq (2+(5+Dec(5)))) => ErrCodRF(o1+s3,s2) = CodRFLen(s3,s2--o1);
(Length(s1) lt NatNum(Dec(2))) => CodRFLen(s1,s2) = s2++s1;
( (NatNum(BStr(o1))++BStr(o2)) gt (6+(6+Dec(5)))) or (NatNum(BStr(o1))++BStr(o2)) lt (2+(5+Dec(5)))) ) =>
  CodRFLen(o1+(o2+s3),s2) = s2++(o1+Octet(o2));
( (Length(s3) lt NatNum(BStr(o1))++BStr(o2))) and (NatNum(BStr(o1))++BStr(o2)) le (6+(6+Dec(5)))) and
  (NatNum(BStr(o1))++BStr(o2)) ge (2+(5+Dec(5)))) ) =>
  CodRFLen(o1+(o2+s3),s2) = s2++s1;
( (Length(s3) eq NatNum(BStr(o1))++BStr(o2))) and (NatNum(BStr(o1))++BStr(o2)) le (6+(6+Dec(5)))) and
  (NatNum(BStr(o1))++BStr(o2)) ge (2+(5+Dec(5)))) ) =>
  CodRFLen(o1+(o2+s3++s4),s2) = CodRF1(s3,s2++(o1+Octet(o2)),s4);
First(CorrConld(s1,First(s2))) => CodRF1(s1,s2,s3) =
  CodRF2(Third(CorrConld(s1,First(s2))),s2++Second(CorrConld(s1,First(s2))),s3);
not(First(CorrConld(s1,First(s2)))) => CodRF1(s1,s2,s3) = s2++Second(CorrConld(s1,First(s2)));
First(CorrTransDisc(s1)) => CodRF2(s1,s2,s3) =
  CodRF3(Third(CorrTransDisc(s1)),s2++Second(CorrTransDisc(s1)),s3);
```

```

not(First(CorrTransDisc(s1))) => CodRF2(s1,s2,s3) = s2++Second(CorrTransDisc(s1));
First(CorrSesReq(s1)) =>
    CodRF3(s1,s2,s3) = CodRF4(Third(CorrSesReqs1)),s2++Second(CorrSesReq(s1)),s3);
not(First(CorrSesReq(s1))) => CodRF3(s1,s2,s3) = s2++Second(CorrSesReq(s1));
First(CorrVersNum(s1)) =>
    CodRF4(s1,s2,s3) = CodRF5(Third(CorrVersNum(s1)),s2++Second(CorrVersNum(s1)),s3);
not(First(CorrVersNum(s1))) => CodRF4(s1,s2,s3) = s2++Second(CorrVersNum(s1));
First(CorrReasCode(s1)) => CodRF5(s1,s2,s3) = ErrCodStr(s3);
not(First(CorrReasCode(s1))) => CodRF5(s1,s2,s3) = s2++Second(CorrReasCode(s1));

```

ofsort Err3Tuple

```

CorrTransDisc(s1) = CorrPar(1+Dec(7),Dec(1),s1);
CorrSesReq(s1) = CorrPar(2+Dec(0),Dec(2),s1);
CorrVersNum(s1) = CorrPar(2+Dec(2),Dec(1),s1);
CorrReasCode(s1) = CorrParm(5+Dec(0),5+(1+Dec(3)),s1);

```

(* FN SPDU valid encoding checks: *)

ofsort DatOctStr

```

Empty(s1) => ErrCodFN(s1,s2) = s2;
(o1 eq Dec(0)) => ErrCodRF(o1+s3,s2) = ErrCodStr(s3);
( (o1 lt (2+(5+Dec(5)))) and (Length(s3) lt o1) ) => ErrCodFN(o1+s3,s2) = s2++s1;
( (o1 lt (2+(5+Dec(5)))) and (Length(s3) eq o1) ) => ErrCodFN(o1+s3++s4,s2) = CodFN1(s3,s2--o1,s4);
(o1 eq (2+(5+Dec(5)))) => ErrCodFN(o1+s3,s2) = CodFNLen(s3,s2--o1);
(Length(s1) lt NatNum(Dec(2))) => CodFNLen(s1,s2) = s2++s1;
( (NatNum(BStr(o1)++BStr(o2)) gt (5+(2+Dec(3)))) or (NatNum(BStr(o1)++BStr(o2)) lt (2+(5+Dec(5)))) ) =>
    CodFNLen(o1+(o2+s3),s2) = s2++(o1+Octet(o2));
( (Length(s3) lt NatNum(BStr(o1)++BStr(o2))) and (NatNum(BStr(o1)++BStr(o2)) le (5+(2+Dec(3)))) and
  (NatNum(BStr(o1)++BStr(o2)) ge (2+(5+Dec(5)))) ) =>
    CodFNLen(o1+(o2+s3),s2) = s2++s1;
( (Length(s3) eq NatNum(BStr(o1)++BStr(o2))) and (NatNum(BStr(o1)++BStr(o2)) le (5+(2+Dec(3)))) and
  (NatNum(BStr(o1)++BStr(o2)) ge (2+(5+Dec(5)))) ) =>
    CodFNLen(o1+(o2+s3++s4),s2) = CodFN1(s3,s2++(o1+Octet(o2)),s4);
First(CorrTransDisc(s1,First(s2))) => CodFN1(s1,s2,s3) =
    CodFN2(Third(CorrTransDisc(s1,First(s2))),s2++Second(CorrTransDisc(s1,First(s2))),s3);
not(First(CorrTransDisc(s1,First(s2)))) => CodFN1(s1,s2,s3) = s2++Second(CorrTransDisc(s1,First(s2)));
First(CorrParm(1+(9+Dec(3)),5+(1+Dec(3)),s1)) => CodFN2(s1,s2,s3) = ErrCodStr(s3);
not(First(CorrParm(1+(9+Dec(3)),5+(1+Dec(3)),s1))) =>
    CodFN2(s1,s2,s3) = s2++Second(CorrParm(1+(9+Dec(3)),5+(1+Dec(3)),s1));

```

(* DN and NF SPDU valid encoding checks: *)

```

Empty(s1) => ErrCodDNorNF(s1,s2) = s2;
(o1 eq Dec(0)) => ErrCodDNorNF(o1+s3,s2) = ErrCodStr(s3);
( (o1 lt (2+(5+Dec(5)))) and (Length(s3) lt o1) ) => ErrCodDNorNF(o1+s3,s2) = s2++s1;
( (o1 lt (2+(5+Dec(5)))) and (Length(s3) eq o1) ) =>
    ErrCodDNorNF(o1+s3++s4,s2) = CodDNorNF1(s3,s2--o1,s4);
(o1 eq (2+(5+Dec(5)))) => ErrCodDNorNF(o1+s3,s2) = CodDNorNFLen(s3,s2--o1);
(Length(s1) lt NatNum(Dec(2))) => CodDNorNFLen(s1,s2) = s2++s1;
( (NatNum(BStr(o1)++BStr(o2)) gt (5+(1+Dec(6)))) or (NatNum(BStr(o1)++BStr(o2)) lt (2+(5+Dec(5)))) ) =>
    CodDNorNFLen(o1+(o2+s3),s2) = s2++(o1+Octet(o2));
( (Length(s3) lt NatNum(BStr(o1)++BStr(o2))) and (NatNum(BStr(o1)++BStr(o2)) le (5+(1+Dec(6)))) and
  (NatNum(BStr(o1)++BStr(o2)) ge (2+(5+Dec(5)))) ) =>
    CodDNorNFLen(o1+(o2+s3),s2) = s2++s1;
( (Length(s3) eq NatNum(BStr(o1)++BStr(o2))) and (NatNum(BStr(o1)++BStr(o2)) le (5+(1+Dec(6)))) and
  (NatNum(BStr(o1)++BStr(o2)) ge (2+(5+Dec(5)))) ) =>
    CodDNorNFLen(o1+(o2+s3++s4),s2) = CodDNorNF1(s3,s2++(o1+Octet(o2)),s4);
First(CorrParm(1+(9+Dec(3)),5+(1+Dec(3)),s1)) => CodDNorNF1(s1,s2,s3) = ErrCodStr(s3);
not(First(CorrParm(1+(9+Dec(3)),5+(1+Dec(3)),s1))) =>
    CodDNorNF1(s1,s2,s3) = s2++Second(CorrParm(1+(9+Dec(3)),5+(1+Dec(3)),s1));

```