JTC 1

TECHNICAL REPORT

ISO/IEC TR 9571

First edition
1989-09-15

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

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

Reference number
ISO/IEC/TR 9571 : 1989 (E)

Contents

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 Technical reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.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 9571, 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 decriptions 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 8326. It forms therefore a companion document to ISO 8326. It takes account of the Defect Reports incorporated in the standard (annex A of ISO 8326), 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 service

1 Scope

This Technical Report contains a formal description of the OSI Basic Connection Oriented Session Service defined in ISO 8326. 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 applicable for use in formal descriptions in LOTOS of the OSI Connection Oriented Session Protocol defined in ISO 8327, namely ISO/IEC TR 9572, and of the OSI Connection Oriented Presentation Protocol defined in ISO 8823.

The formal description is not limited to a single session connection, but also describes the service instances that result from multiple session connections either in parallel or in sequence. It therefore also formalizes aspects of multiplicity which are not presented in ISO 8326 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 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/TR 8509: 1987, *Information processing systems - Open Systems Interconnection - Service conventions.*

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

ISO 8823: 1988, *Information processing systems - Open Systems Interconnection - Connection oriented presentation protocol specification.*

ISO/IEC/TR 9572: 1989, *Information processing systems - Open Systems Interconnection - LOTOS description of the session protocol.*

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 8326 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 8326.

The following additional abbreviations are employed in this Techical Report:

SC	session connection
SCEP	session connection endpoint
SCEI	session connection endpoint identifier
SSP	session service primitive

5 Conventions

Clauses 6 through 13 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.

The conventions defined in ISO/TR 8509 are adopted.

[1] To be published.

(* start of LOTOS text ----------

6 Introduction to the formal description

The formal description relates to the dynamic behaviour observable at the SS boundary. The whole service boundary is formally represented by a single gate *s*. Events at *s* consist of three values, of sort *SAddress*, *SCEI* and *SSP*, respectively. The first value identifies the SSAP where the event occurs. The second value identifies the SCEI within that SSAP where the event occurs. The third value represents the SSP executed in the event.

NOTE - An event is an atomic form of interaction. SSPs are thus represented as atomic interactions, which is consistent with ISO 8326. However, for reasons of consistency with the formal description of the session protocol, where flow control and segmenting require a finer granularity of those SSPs that can carry an unlimited amount of user data, this representation may have to be changed.

The possibility of multiple concurrent and consecutive SCs is represented by the formal description. Since this multiplicity is not restricted other than by nondeterministic decisions to not service certain requests and by the requirement of unambiguous use of SCEIs, the behaviour described is that of a non-terminating SS-provider.

A constraint-oriented style is adopted for the specification: different "types" of constraints in the service definition are specified in separate processes; these processes are appropriately composed, either synchronized or unsynchronized, in the specification to represent the total set of constraints. The decomposition in separate constraints is done at several levels.

The top level decomposition shows a structuring of the global constraints of the session service into the following separate constraints:

a) The dynamic behaviour of all potential SCs (described by process *SConnections*). This constraint is explicitly not concerned with the constraints below, i.e. it assumes no limits on the SS-provider capacity and does not care about administrative concerns for connection identification.

b) The correct allocation of SCEIs (described by process *SCIdentification*). It seems to be an explicit choice of ISO 7498 not to specify any requirement on local identification of connection endpoints, other than the obvious one that connection endpoint identifiers locally provide the means of distinguishing connections at the same service access point. This constraint applies precisely to this requirement.

c) The acceptance vs. refusal of new SCs by the SS-provider (described by process *SCAcceptance*). In spite of the "independence" of concurrent connections, finiteness of resources implies that interdependencies may exist. Such interdependencies concern the availability of resources to new connections in the same end-system, i.e. the ability of the service provider to accept an additional connection.

d) The backpressure flow control exerted by the SS-provider (described by process *SBackpressure*). This constraint relates to the fact that existing connections may be temporary blocked by the service provider, e.g. for reasons of resource shortage.

Constraint a) above is further decomposed into constraints related to the dynamic behaviour of a single SC (described by process *SConnection*). Each of these, in turn, is decomposed into "local" and "end-to-end" constraints (described by process *SCEP* and process *SCEPAssociation*,

respectively). Local constraints are constraints which are local to a SCEP, whereas end-to-end constraints concern the end-to-end relations which result from the exchange via the SS-provider.

Figure 1 gives an overview of the processes used for the formal definition of the Session Service and their relations (some of the lowest level processes are omitted). It also indicates the clauses where the definition of these processes can be found.

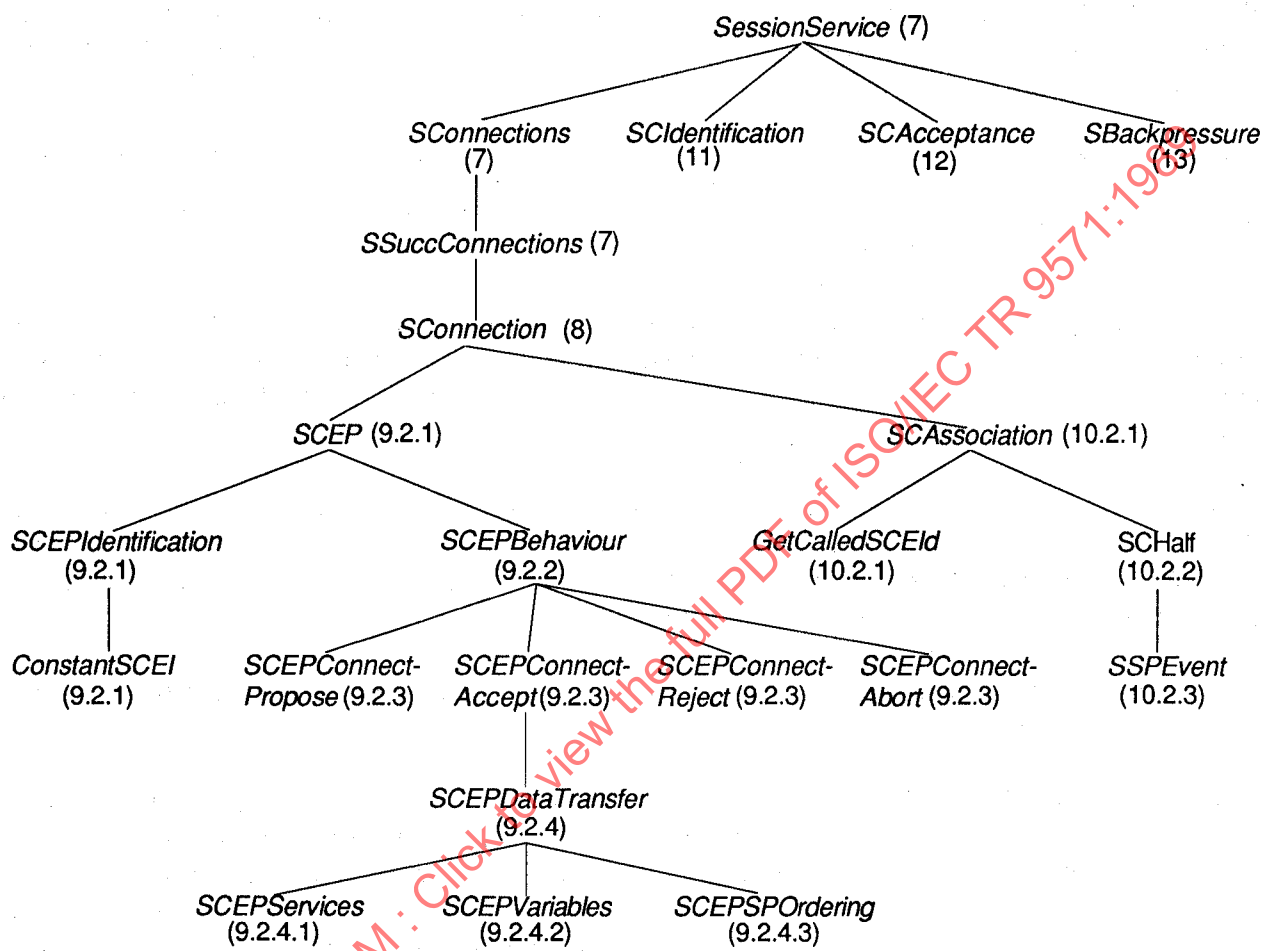

Figure 1 - Processes related by a tree structure: each "father" process contains (or is constructed from) one or more instances of its immediate "descendant" process(es)

The definition of data types precedes the definition of the dynamic behaviour in which these data types are used. Apart from standard data types, which are imported from the LOTOS library of data types, and some "ad-hoc" data types, two groupings of data types are identified, namely interface data types and association data types. Interface data types are used in the specification of local constraints, and are to be shared with the formal descriptions that may interwork with the present one, namely those of the Session Protocol and of the Presentation Protocol. Association data types are used in the specification of the end-to-end constraints; they abstract from the underlying protocol and are therefore probably not re-usable in other formal descriptions.

7 Global constraints of the session service

Standard data types for the construction of a boolean, a natural number, a generic set, a string of octets, and representations of a natural number (including the decimal representation) are imported from the LOTOS library of data types by means of the *library* construct.

The specification of the behaviour of the SS-provider consists of the conjunction of four separate constraints, as explained in clause 6. *SConnections* is described as being able to support a potentially infinite number of independent SCs. Concurrency is here represented by multiple instances of *SSuccConnections*, where each instance describes a succession of SCs. An SC is represented by a distinct instance of *SConnection*. Each instance of *SConnection* can terminate. The composites *SConnections* and *SSuccConnections* can never terminate because the possibility always exists that a new instance of *SConnection* is invoked. Thus, at any time, any number of SCs can be active. The SS-provider's nondeterminacy in limiting this potentially infinite concurrency is specified as a separate constraint, viz. by *SCAcceptance* (see clause 12). *SCIdentification* prescribes unique identification of SCEPs (see clause 11). *SBackpressure* enforces the constraint relating the SS-provider's backpressure (see clause 13).

```
----------------------------------------------------------------------------------------------------------------------*)
specification SessionService [s]: noexit

library FBoolean, Boolean, NaturalNumber, Element, Set, OctetString, NatRepresentations endlib

behaviour
SConnections [s] || SCIdentification [s] || SCAcceptance [s] || SBackpressure [s]
where

process SConnections [s]: noexit := SSuccConnections [s] ||| SConnections [s]
where
process SSuccConnections [s]: noexit := SConnection [s] >> SSuccConnections [s] endproc
endproc (* SConnections *)
(*----------------------------------------------------------------------------------------------------------------------
```

8 Provision of a session connection

Two separate constraints are distinguished with respect to the behaviour of a single SC, namely local constraints and end-to-end constraints. Local constraints apply to the possible behaviour at each SCEP taking only into account the history of SSPs executed at that SCEP. End-to-end constraints, on the other hand, relate the possible behaviour at each SCEP to the history of SSPs at the other SCEP (that is, the other end of the SC).

Local constraints are represented by two parallel instances of *SCEP* (see 9.2) that apply to the interaction with the Calling SS-user and with the Called SS-user, respectively. The type *SSUserRole* presents two constants that distinguish between these roles (see 9.1.5.1 for the definition of *Doublet*). End-to-end constraints are represented by an instance of *SCEPAssociation* (see 10.2).

The end of the SC lifetime is represented by termination of the two instances of *SCEP*; they force the instance of *SCEPAssociation*, a non-terminating process, to be disabled.

```
----------------------------------------------------------------------------------------------------------------------*)
type SSUserRole is Doublet renamedby
sortnames   SSUserRole for Doublet
opnnames    calling for constant1          called for constant2
endtype

process SConnection [s]: exit :=
( SCEP [s] (calling) ||| SCEP [s] (called) ) || ( SCAssociation [s] [> exit )
endproc
(*----------------------------------------------------------------------------------------------------------------------
```

9 Local constraints at a session connection endpoint

9.1 Interface data types

The interface data types consist of definitions for the construction of a Session (Service Access Point) address (see 9.1.1), a SCEI (see 9.1.2), and a SSP (see 9.1.3 and 9.1.4). A number of auxiliary definitions of general use are presented in 9.1.5.

Almost all type definitions include the definition of two infix boolean functions, *eq* and *ne*, for testing the equality and unequality, respectively, of two values of some sort.

9.1.1 Session address

No structure of the (Calling, Called or Responding) Session Address parameter is defined by ISO 8326. The following definition allows to represent an infinite number of Session Address values.

```
----------------------------------------------------------------------------------------------------------------------*)
type SessionAddress is Boolean
sorts SAddress
opns
someAddress: -> SAddress
AnotherAddress: SAddress -> SAddress
_eq_, _ne_: SAddress, SAddress -> Bool
eqns forall a,a1:SAddress ofsort Bool
a1 ne a = not (a1 eq a);                          someAddress eq someAddress = true;
someAddress eq AnotherAddress (a) = false;        AnotherAddress (a) eq someAddress = false;
AnotherAddress (a1) eq AnotherAddress (a) = a1 eq a;
endtype
(*----------------------------------------------------------------------------------------------------------------------
```

9.1.2 Session connection endpoint identifier

No structure of SCEIs is defined by ISO 8326. The following definition allows to represent an infinite number of SCEI values.

```
----------------------------------------------------------------------------------------------------------------------*)
type SCEndpointIdentifier is Boolean
sorts SCEI
opns
someCEI: -> SCEI
AnotherCEI: SCEI -> SCEI
_eq_, _ne_: SCEI, SCEI -> Bool
eqns forall e,e1:SCEI ofsort Bool
e1 ne e = not (e1 eq e);                          someCEI eq someCEI = true;
someCEI eq AnotherCEI (e) = false;                AnotherCEI (e) eq someCEI = false;
AnotherCEI (e1) eq AnotherCEI (e) = e1 eq e;
endtype
(*----------------------------------------------------------------------------------------------------------------------
```

9.1.3 Session service primitive

The specification of the SSP type is accomplished by way of a number of hierarchical type definitions. First, the basic construction of SSP values is presented (see 9.1.3.1), then a classification of SSPs (see 9.1.3.2), and subsequently a number of additional functions on SSPs (see 9.1.3.3 through 9.1.3.5). The individual SSP parameters are defined in 9.1.4.

9.1.3.1 Session service primitive basic construction

The construction of SSP values in type *BasicSSP* is a direct formulation of table 5 through 7 of ISO 8326. The functions that yield SSP values are referred to as "constructor" functions. This definition imports the definitions that relate to SSP parameters (see 9.1.4).

```
----------------------------------------------------------------------------------------------------------------------------*)
type BasicSSP is
SCIdentifier, SessionAddress, SQuality, SConnectResult, SRequirements, SSynchronizationNumber,
STokensAssignment, SData, STokens, SSyncMinorType, SResynchronizeType, SPExceptionReason,
SUExceptionReason, SActivityIdentifier, SReleaseResult, SPAbortReason
sorts SSP
opns
SCONreq, SCONind: SCRef, SAddress, SAddress, SQOS, SFUs, SSPSN, STsAss, SData -> SSP
SCONrsp, SCONcnf: SCRef, SAddress, SConResult, SQOS, SFUs, SSPSN, STsAss, SData -> SSP
SDTreq, SDTind, SEXreq, SEXind, STDreq, STDind, SCDreq, SCDind, SCDrsp, SCDcnf, SSYNMarsp,
      SSYNMacnf,SUABreq, SUABind, SACTErsp, SACTEcnf, SRELreq, SRELind: SData -> SSP
SGTreq, SGTind: STokens -> SSP
SPTreq, SPTind: STokens, SData -> SSP
SCGreq, SCGind, SACTIrsp, SACTIcnf, SACTDrsp, SACTDcnf: -> SSP
SSYNmreq, SSYNmind: SSyncType, SSPSN, SData -> SSP
SSYNmrsp, SSYNmcnf, SSYNMareq, SSYNMaind, SACTEreq, SACTEind: SSPSN, SData -> SSP
SRSYNreq, SRSYNind: SResynType, SSPSN, STsAss, SData -> SSP
SRSYNrsp, SRSYNcnf: SSPSN, STsAss, SData -> SSP
SPERind: SPExcReason -> SSP
SUERreq, SUERind: SUExcReason, SData -> SSP
SACTSreq, SACTSind: SActId, SData -> SSP
SACTRreq, SACTRind: SActId, SActId, SSPSN, SCRef, SData -> SSP
SACTIreq, SACTIind, SACTDreq, SACTDind: SUExcReason -> SSP
SRELrsp, SRELcnf: SRelResult, SData -> SSP
SPABind: SPAbReason -> SSP
endtype
(*----------------------------------------------------------------------------------------------------------------------------
```

9.1.3.2 Session service primitive classification

Type *SSPConstant* defines a classification of SSPs. Each class, or "type", of SSP is represented by a constant with a name similar to the name used in table 5 through 7 of ISO 8326 to represent the SSP. The auxiliary function *h* that maps these constants to natural numbers is defined in order to simplify the definition of equality on SSP classes (and on SSP values, see 9.1.3.4). The boolean function *IsConstantReq* determines whether its argument is a constant indicating a request or response SSP. Similarly, *IsConstantInd* characterizes indication or confirm SSPs. *Even* and *Odd* are auxiliary functions used in the definition of *IsConstantReq* and *IsConstantInd*. *Succ* is defined by the standard type *NaturalNumber* and yields the successor of a given natural number.

Type *SSPBasicClassifiers* is a functional enrichment of type *BasicSSP*. It defines a function *k* that yields the constant corresponding to its argument SSP (thus relating abbreviated SSP names to their full names; the abbreviated names correspond with those defined in annex A of ISO 8326, except *SSYNMa* which corresponds with SSYNM). Furthermore, "recognizer" functions are defined that test whether their argument SSP is of a certain class. There are functions to test whether a SSP is a request, indication, response, or confirm of a particular service. In addition, there are functions that test whether a SSP is either a request or indication, or either a response or confirm of a particular service. *IsReq* and *IsInd* recognize whether the argument SSP is respectively a request or response and an indication or confirm; *IsProvGenerated* recognizes whether the argument SSP is generated by the SS-provider.

```
----------------------------------------------------------------------------------------------------------------------*)
type SSPConstant is NaturalNumber
sorts SSPConstant
opns
S-CONNECTrequest, S-CONNECTindication, S-CONNECTresponse, S-CONNECTconfirm,
      S-DATArequest, S-DATAindication, S-EXPEDITED-DATArequest, S-EXPEDITED-DATAindication,
      S-TYPED-DATArequest, S-TYPED-DATAindication, S-CAPABILITY-DATArequest,
      S-CAPABILITY-DATAindication, S-CAPABILITY-DATAresponse, S-CAPABILITY-DATAconfirm,
      S-TOKEN-GIVErequest, S-TOKEN-GIVEindication, S-TOKEN-PLEASErequest,
      S-TOKEN-PLEASEindication, S-CONTROL-GIVErequest, S-CONTROL-GIVEindication,
      S-SYNC-MINORrequest, S-SYNC-MINORindication, S-SYNC-MINORresponse,
      S-SYNC-MINORconfirm, S-SYNC-MAJORrequest, S-SYNC-MAJORindication,
      S-SYNC-MAJORresponse, S-SYNC-MAJORconfirm, S-RESYNCHRONIZErequest,
      S-RESYNCHRONIZEindication, S-RESYNCHRONIZEresponse, S-RESYNCHRONIZEconfirm,
      S-P-EXCEPTION-REPORTindication, S-U-EXCEPTION-REPORTrequest,
      S-U-EXCEPTION-REPORT-indication, S-ACTIVITY-STARTrequest, S-ACTIVITY-STARTindication,
      S-ACTIVITY-RESUMErequest, S-ACTIVITY-RESUMEindication, S-ACTIVITY-INTERRUPTrequest,
      S-ACTIVITY-INTERRUPTindication, S-ACTIVITY-INTERRUPTresponse,
      S-ACTIVITY-INTERRUPTconfirm, S-ACTIVITY-DISCARDrequest, S-ACTIVITY-DISCARDindication,
      S-ACTIVITY-DISCARDresponse, S-ACTIVITY-DISCARDconfirm, S-ACTIVITY-ENDrequest,
      S-ACTIVITY-ENDindication, S-ACTIVITY-ENDresponse, S-ACTIVITY-ENDconfirm,
      S-RELEASErequest, S-RELEASEindication, S-RELEASEresponse, S-RELEASEconfirm,
      S-U-ABORTrequest, S-U-ABORTindication, S-P-ABORTindication: -> SSPConstant
h: SSPConstant -> Nat                                   Even, Odd: Nat -> Bool
IsConstantReq, IsConstantInd: SSPConstant -> Bool
_eq_, _ne_: SSPConstant, SSPConstant -> Bool
eqns forall c,c1:SSPConstant, n:Nat
ofsort Nat
h(S-CONNECTrequest) = 0;
h(S-CONNECTindication) = Succ (h(S-CONNECTrequest));
h(S-CONNECTresponse) = Succ (h(S-CONNECTindication));
h(S-CONNECTconfirm) = Succ (h(S-CONNECTresponse));
h(S-DATArequest) = Succ (h(S-CONNECTconfirm));
h(S-DATAindication) = Succ (h(S-DATArequest));
h(S-EXPEDITED-DATArequest) = Succ (h(S-DATAindication));
h(S-EXPEDITED-DATAindication) = Succ (h(S-EXPEDITED-DATArequest));
h(S-TYPED-DATArequest) = Succ (h(S-EXPEDITED-DATAindication));
h(S-TYPED-DATAindication) = Succ (h(S-TYPED-DATArequest));
h(S-CAPABILITY-DATArequest) = Succ (h(S-TYPED-DATAindication));
h(S-CAPABILITY-DATAindication) = Succ (h(S-CAPABILITY-DATArequest));
h(S-CAPABILITY-DATAresponse) = Succ (h(S-CAPABILITY-DATAindication));
h(S-CAPABILITY-DATAconfirm) = Succ (h(S-CAPABILITY-DATAresponse));
h(S-TOKEN-GIVErequest) = Succ (h(S-CAPABILITY-DATAconfirm));
h(S-TOKEN-GIVEindication) = Succ (h(S-TOKEN-GIVErequest));
h(S-TOKEN-PLEASErequest) = Succ (h(S-TOKEN-GIVEindication));
h(S-TOKEN-PLEASEindication) = Succ (h(S-TOKEN-PLEASErequest));
h(S-CONTROL-GIVErequest) = Succ (h(S-TOKEN-PLEASEindication));
h(S-CONTROL-GIVEindication) = Succ (h(S-CONTROL-GIVErequest));
h(S-SYNC-MINORrequest) = Succ (h(S-CONTROL-GIVEindication));
h(S-SYNC-MINORindication) = Succ (h(S-SYNC-MINORrequest));
h(S-SYNC-MINORresponse) = Succ (h(S-SYNC-MINORindication));
h(S-SYNC-MINORconfirm) = Succ (h(S-SYNC-MINORresponse));
h(S-SYNC-MAJORrequest) = Succ (h(S-SYNC-MINORconfirm));
h(S-SYNC-MAJORindication) = Succ (h(S-SYNC-MAJORrequest));
h(S-SYNC-MAJORresponse) = Succ (h(S-SYNC-MAJORindication));
h(S-SYNC-MAJORconfirm) = Succ (h(S-SYNC-MAJORresponse));
h(S-RESYNCHRONIZErequest) = Succ (h(S-SYNC-MAJORconfirm));
h(S-RESYNCHRONIZEindication) = Succ (h(S-RESYNCHRONIZErequest));
h(S-RESYNCHRONIZEresponse) = Succ (h(S-RESYNCHRONIZEindication));
h(S-RESYNCHRONIZEconfirm) = Succ (h(S-RESYNCHRONIZEresponse));
h(S-P-EXCEPTION-REPORTindication) = Succ (Succ (h(S-RESYNCHRONIZEconfirm)));
```

```
h(S-U-EXCEPTION-REPORTrequest) = Succ (h(S-P-EXCEPTION-REPORTindication));
h(S-U-EXCEPTION-REPORTindication) = Succ (h(S-U-EXCEPTION-REPORTrequest));
h(S-ACTIVITY-STARTrequest) = Succ (h(S-U-EXCEPTION-REPORTindication));
h(S-ACTIVITY-STARTindication) = Succ (h(S-ACTIVITY-STARTrequest));
h(S-ACTIVITY-RESUMErequest) = Succ (h(S-ACTIVITY-STARTindication));
h(S-ACTIVITY-RESUMEindication) = Succ (h(S-ACTIVITY-RESUMErequest));
h(S-ACTIVITY-INTERRUPTrequest) = Succ (h(S-ACTIVITY-RESUMEindication));
h(S-ACTIVITY-INTERRUPTindication) = Succ (h(S-ACTIVITY-INTERRUPTrequest));
h(S-ACTIVITY-INTERRUPTresponse) = Succ (h(S-ACTIVITY-INTERRUPTindication));
h(S-ACTIVITY-INTERRUPTconfirm) = Succ (h(S-ACTIVITY-INTERRUPTresponse));
h(S-ACTIVITY-DISCARDrequest) = Succ (h(S-ACTIVITY-INTERRUPTconfirm));
h(S-ACTIVITY-DISCARDindication) = Succ (h(S-ACTIVITY-DISCARDrequest));
h(S-ACTIVITY-DISCARDresponse) = Succ (h(S-ACTIVITY-DISCARDindication));
h(S-ACTIVITY-DISCARDconfirm) = Succ (h(S-ACTIVITY-DISCARDresponse));
h(S-ACTIVITY-ENDrequest) = Succ (h(S-ACTIVITY-DISCARDconfirm));
h(S-ACTIVITY-ENDindication) = Succ (h(S-ACTIVITY-ENDrequest));
h(S-ACTIVITY-ENDresponse) = Succ (h(S-ACTIVITY-ENDindication));
h(S-ACTIVITY-ENDconfirm) = Succ (h(S-ACTIVITY-ENDresponse));
h(S-RELEASErequest) = Succ (h(S-ACTIVITY-ENDconfirm));
h(S-RELEASEindication) = Succ (h(S-RELEASErequest));
h(S-RELEASEresponse) = Succ (h(S-RELEASEindication));
h(S-RELEASEconfirm) = Succ (h(S-RELEASEresponse));
h(S-U-ABORTrequest) = Succ (h(S-RELEASEconfirm));
h(S-U-ABORTindication) = Succ (h(S-U-ABORTrequest));
h(S-P-ABORTindication) = Succ (Succ (h(S-U-ABORTindication)));
ofsort Bool
Even (0) = true;                Even (Succ (0)) = false;        Even (Succ (Succ (n))) = Even (n);
Odd (n) = not (Even (n));       IsConstantReq (c) = Even(h(c)); IsConstantInd (c) = Odd (h(c));
c1 eq c = h(c1) eq h(c);        c1 ne c = not (c1 eq c);
endtype

type SSPBasicClassifiers is BasicSSP, SSPConstant
opns
k: SSP -> SSPConstant
IsSCONreq, IsSCONind, IsSCONrsp, IsSCONcnf, IsSCON, IsSCONAK, IsSDTreq, IsSDTind, IsSDT,
      IsSEXreq, IsSEXind, IsSEX, IsSTDreq, IsSTDind, IsSTD, IsSCDreq, IsSCDind, IsSCDrsp, IsSCDcnf,
      IsSCD, IsSCDAK, IsSGTreq, IsSGTind, IsSGT, IsSPTreq, IsSPTind, IsSPT, IsSCGreq, IsSCGind,
      IsSCG, IsSSYNmreq, IsSSYNmind, IsSSYNmrsp, IsSSYNmcnf, IsSSYNm, IsSSYNmAK,
      IsSSYNMareq, IsSSYNMaind, IsSSYNMarsp, IsSSYNMacnf, IsSSYNMa, IsSSYNMaAK,
      IsSRSYNreq, IsSRSYNind, IsSRSYNrsp, IsSRSYNcnf, IsSRSYN, IsSRSYNAK, IsSPERind,
      IsSUERreq, IsSUERind, IsSUER, IsSACTSreq, IsSACTSind, IsSACTS, IsSACTRreq, IsSACTRind,
      IsSACTR, IsSACTIreq, IsSACTIind, IsSACTIrsp, IsSACTIcnf, IsSACTI, IsSACTIAK, IsSACTDreq,
      IsSACTDind, IsSACTDrsp, IsSACTDcnf, IsSACTD, IsSACTDAK, IsSACTEreq, IsSACTEind,
      IsSACTErsp, IsSACTEcnf, IsSACTE, IsSACTEAK, IsSRELreq, IsSRELind, IsSRELrsp, IsSRELcnf,
      IsSREL, IsSRELAK, IsSUABreq, IsSUABind, IsSUAB, IsSPABind: SSP -> Bool
IsReq, IsInd, IsProvGenerated: SSP -> Bool
eqns forall r:SCRef, cg,cd,rg:SAddress, q:SQOS, cr:SConResult, rs:SFUs, sn:SSPSN, a:STsAss,
      d:SData, ts:STokens, st:SSyncType, rt:SResynType, per:SPExcReason, uer,er:SUExcReason,
      ai,aio:SActId, rr:SRelResult, ar:SPAbReason, p:SSP
ofsort SSPConstant
k(SCONreq(r,cg,cd,q,rs,sn,a,d)) = S-CONNECTrequest;
k(SCONind(r,cg,cd,q,rs,sn,a,d)) = S-CONNECTindication;
k(SCONrsp(r,rg,cr,q,rs,sn,a,d)) = S-CONNECTresponse;
k(SCONcnf(r,rg,cr,q,rs,sn,a,d)) = S-CONNECTconfirm;
k(SDTreq(d)) = S-DATArequest;                   k(SDTind(d)) = S-DATAindication;
k(SEXreq(d)) = S-EXPEDITED-DATArequest;         k(SEXind(d)) = S-EXPEDITED-DATAindication;
k(STDreq(d)) = S-TYPED-DATArequest;             k(SDTind(d)) = S-TYPED-DATAindication;
k(SCDreq(d)) = S-CAPABILITY-DATArequest;        k(SCDind(d)) = S-CAPABILITY-DATAindication;
k(SCDrsp(d)) = S-CAPABILITY-DATAresponse;       k(SCDcnf(d)) = S-CAPABILITY-DATAconfirm;
k(SGTreq(ts)) = S-TOKEN-GIVErequest;            k(SGTind(ts)) = S-TOKEN-GIVEindication;
k(SPTreq(ts,d)) = S-TOKEN-PLEASErequest;        k(SPTind(ts,d)) = S-TOKEN-PLEASEindication;
```

k(SCGreq) = S-CONTROL-GIVErequest; *k(SCGind) = S-CONTROL-GIVEindication;*
k(SSYNmreq(st,sn,d)) = S-SYNC-MINORrequest; *k(SSYNmind(st,sn,d)) = S-SYNC-MINORindication;*
k(SSYNmrsp(sn,d)) = S-SYNC-MINORresponse; *k(SSYNmcnf(sn,d)) = S-SYNC-MINORconfirm;*
k(SSYNMareq(sn,d)) = S-SYNC-MAJORrequest; *k(SSYNMaind(sn,d)) = S-SYNC-MAJORindication;*
k SSYNMarsp(d)) = S-SYNC-MAJORresponse; *k(SSYNMacnf(d)) = S-SYNC-MAJORconfirm;*
k(SRSYNreq(rt,sn,a,d)) = S-RESYNCHRONIZErequest;
k(SRSYNind(rt,sn,a,d)) = S-RESYNCHRONIZEindication;
k(SRSYNrsp(sn,a,d)) = S-RESYNCHRONIZEresponse;
k(SRSYNcnf(sn,a,d)) = S-RESYNCHRONIZEconfirm;
k(SPERind(per)) = S-P-EXCEPTION-REPORTindication;
k(SUERreq(uer,d)) = S-U-EXCEPTION-REPORTrequest;
k(SUERind(uer,d)) = S-U-EXCEPTION-REPORTindication;
k(SACTSreq(ai,d)) = S-ACTIVITY-STARTrequest; *k(SACTSind(ai,d)) = S-ACTIVITY-STARTindication;*
k(SACTRreq(ai,aio,sn,r,d)) = S-ACTIVITY-RESUMErequest;
k(SACTRind(ai,aio,sn,r,d)) = S-ACTIVITY-RESUMEindication;
k(SACTIreq(er)) = S-ACTIVITY-INTERRUPTrequest;
k(SACTIind(er)) = S-ACTIVITY-INTERRUPTindication;
k(SACTIrsp) = S-ACTIVITY-INTERRUPTresponse; *k(SACTIcnf) = S-ACTIVITY-INTERRUPTconfirm;*
k(SACTDreq(er)) = S-ACTIVITY-DISCARDrequest; *k(SACTDind(er)) = S-ACTIVITY-DISCARDindication;*
k(SACTDrsp) = S-ACTIVITY-DISCARDresponse; *k(SACTDcnf) = S-ACTIVITY-DISCARDconfirm;*
k(SACTEreq(sn,d)) = S-ACTIVITY-ENDrequest; *k(SACTEind(sn,d)) = S-ACTIVITY-ENDindication;*
k(SACTErsp(d)) = S-ACTIVITY-ENDresponse; *k(SACTEcnf(d)) = S-ACTIVITY-ENDconfirm;*
k(SRELreq(d)) = S-RELEASErequest; *k(SRELind(d)) = S-RELEASEindication;*
k(SRELrsp(rr,d)) = S-RELEASEresponse; *k(SRELcnf(rr,d)) = S-RELEASEconfirm;*
k(SUABreq(d)) = S-U-ABORTrequest; *k(SUABind(d)) = S-U-ABORTindication;*
k(SPABind(ar)) = S-P-ABORTindication;
***ofsort** Bool*
IsSCONreq(p) = k(p) eq S-CONNECTrequest; *IsSCONind(p) = k(p) eq S-CONNECTindication;*
IsSCONrsp(p) = k(p) eq S-CONNECTresponse; *IsSCONcnf(p) = k(p) eq S-CONNECTconfirm;*
IsSDTreq(p) = k(p) eq S-DATArequest; *IsSDTind(p) = k(p) eq S-DATAindication;*
IsSEXreq(p) = k(p) eq S-EXPEDITED-DATArequest;
IsSEXind(p) = k(p) eq S-EXPEDITED-DATAindication;
IsSTDreq(p) = k(p) eq S-TYPED-DATArequest; *IsSTDind(p) = k(p) eq S-TYPED-DATAindication;*
IsSCDreq(p) = k(p) eq S-CAPABILITY-DATArequest;
IsSCDind(p) = k(p) eq S-CAPABILTY-DATAindication;
IsSCDrsp(p) = k(p) eq S-CAPABILITY-DATAresponse;
IsSCDcnf(p) = k(p) eq S-CAPABILITY-DATAconfirm;
IsSGTreq(p) = k(p) eq S-TOKEN-GIVErequest; *IsSGTind(p) = k(p) eq S-TOKEN-GIVEindication;*
IsSPTreq(p) = k(p) eq S-TOKEN-PLEASErequest; *IsSPTind(p) = k(p) eq S-TOKEN-PLEASEindication;*
IsSCGreq(p) = k(p) eq S-CONTROL-GIVErequest; *IsSCGind(p) = k(p) eq S-CONTROL-GIVEindication;*
IsSSYNmreq(p) = k(p) eq S-SYNC-MINORrequest; *IsSSYNmind(p) = k(p) eq S-SYNC-MINORindication;*
IsSSYNmrsp(p) = k(p) eq S-SYNC-MINORresponse; *IsSSYNmcnf(p) = k(p) eq S-SYNC-MINORconfirm;*
IsSSYNMareq(p) = k(p) eq S-SYNC-MAJORrequest;
IsSSYNMaind(p) = k(p) eq S-SYNC-MAJORindication;
IsSSYNMarsp(p) = k(p) eq S-SYNC-MAJORresponse;
IsSSYNMacnf(p) = k(p) eq S-SYNC-MAJORconfirm;
IsSRSYNreq(p) = k(p) eq S-RESYNCHRONIZErequest;
IsSRSYNind(p) = k(p) eq S-RESYNCHRONIZEindication;
IsSRSYNrsp(p) = k(p) eq S-RESYNCHRONIZEresponse;
IsSRSYNcnf(p) = k(p) eq S-RESYNCHRONIZEconfirm;
IsSPERind(p) = k(p) eq S-P-EXCEPTION-REPORTindication;
IsSUERreq(p) = k(p) eq S-U-EXCEPTION-REPORTrequest;
IsSUERind(p) = k(p) eq S-U-EXCEPTION-REPORTindication;
IsSACTSreq(p) = k(p) eq S-ACTIVITY-STARTrequest;
IsSACTSind(p) = k(p) eq S-ACTIVITY-STARTindication;
IsSACTRreq(p) = k(p) eq S-ACTIVITY-RESUMErequest;
IsSACTRind(p) = k(p) eq S-ACTIVITY-RESUMEindication;
IsSACTIreq(p) = k(p) eq S-ACTIVITY-INTERRUPTrequest;
IsSACTIind(p) = k(p) eq S-ACTIVITY-INTERRUPTindication;
IsSACTIrsp(p) = k(p) eq S-ACTIVITY-INTERRUPTresponse;
IsSACTIcnf(p) = k(p) eq S-ACTIVITY-INTERRUPTconfirm;

```
IsSACTDreq(p) = k(p) eq S-ACTIVITY-DISCARDrequest;
IsSACTDind(p) = k(p) eq S-ACTIVITY-DISCARDindication;
IsSACTDrsp(p) = k(p) eq S-ACTIVITY-DISCARDresponse;
IsSACTDcnf(p) = k(p) eq S-ACTIVITY-DISCARDconfirm;
IsSACTEreq(p) = k(p) eq S-ACTIVITY-ENDrequest;
IsSACTEind(p) = k(p) eq S-ACTIVITY-ENDindication;
IsSACTErsp(p) = k(p) eq S-ACTIVITY-ENDresponse;
IsSACTEcnf(p) = k(p) eq S-ACTIVITY-ENDconfirm;
IsSRELreq(p) = k(p) eq S-RELEASErequest;          IsSRELind(p) = k(p) eq S-RELEASEindication;
IsSRELrsp(p) = k(p) eq S-RELEASEresponse;         IsSRELcnf(p) = k(p) eq S-RELEASEconfirm;
IsSUABreq(p) = k(p) eq S-U-ABORTrequest;          IsSUABind(p) = k(p) eq S-U-ABORTindication;
IsSPABind(p) = k(p) eq S-P-ABORTindication;
IsSCON(p) = IsSCONreq(p) or IsSCONind(p);         IsSCONAK(p) = IsSCONrsp(p) or IsSCONcnf(p);
IsSDT(p) = IsSDTreq(p) or IsSDTind(p);            IsSEX(p) = IsSEXreq(p) or IsSEXind(p);
IsSTD(p) = IsSTDreq(p) or IsSTDind(p);            IsSCD(p) = IsSCDreq(p) or IsSCDind(p);
IsSCDAK(p) = IsSCDrsp(p) or IsSCDcnf(p);          IsSGT(p) = IsSGTreq(p) or IsSGTind(p);
IsSPT(p) = IsSPTreq(p) or IsSPTind(p);            IsSCG(p) = IsSCGreq(p) or IsSCGind(p);
IsSSYNm(p) = IsSSYNmreq(p) or IsSSYNmind(p);
IsSSYNmAK(p) = IsSSYNmrsp(p) or IsSSYNmcnf(p);
IsSSYNMa(p) = IsSSYNMareq(p) or IsSSYNMaind(p);
IsSSYNMaAK(p) = IsSSYNMarsp(p) or IsSSYNMacnf(p);
IsSRSYN(p) = IsSRSYNreq(p) or IsSRSYNind(p);      IsSRSYNAK(p) = IsSRSYNrsp(p) or IsSRSYNcnf(p);
IsSUER(p) = IsSUERreq(p) or IsSUERind(p);         IsSACTS(p) = IsSACTSreq(p) or IsSACTSind(p);
IsSACTR(p) = IsSACTRreq(p) or IsSACTRind(p);      IsSACTI(p) = IsSACTIreq(p) or IsSACTIind(p);
IsSACTIAK(p) = IsSACTIrsp(p) or IsSACTIcnf(p);    IsSACTD(p) = IsSACTDreq(p) or IsSACTDind(p);
IsSACTDAK(p) = IsSACTDrsp(p) or IsSACTDcnf(p);    IsSACTE(p) = IsSACTEreq(p) or IsSACTEind(p);
IsSACTEAK(p) = IsSACTErsp(p) or IsSACTEcnf(p);    IsSREL(p) = IsSRELreq(p) or IsSRELind(p);
IsSRELAK(p) = IsSRELrsp(p) or IsSRELcnf(p);       IsSUAB(p) = IsSUABreq(p) or IsSUABind(p);
IsReq(p) = IsConstantReq(k(p));                   IsInd(p) = IsConstantInd(k(p));
IsProvGenerated(p) = IsSPABind(p) or IsSPERind(p);
endtype
(*-------------------------------------------------------------------------------------------------------------------------------------------
```

9.1.3.3 Session service primitive parameter selectors

Type *SSPParameterSelectors* defines a further enrichment with functions that allow to determine the value of individual SSP parameters. Boolean functions are defined to test whether a given parameter value is carried in a given SSP. "Extractor" functions are defined to extract a parameter value from their argument SSP; extraction is only defined for those argument SSPs which are defined to carry the parameter to be extracted.

```
type SSPParameterSelectors is SSPBasicClassifiers
opns
_IsCRefOf_: SCRef, SSP -> Bool
_IsCallingOf_, _IsCalledOf_, _IsRespondingOf_: SAddress, SSP -> Bool
_IsQOSOf_: SQOS, SSP -> Bool                      _IsConResultOf_: SConResult, SSP -> Bool
_IsRqmsOf_: SFUs, SSP -> Bool                     _IsSPSNOf_: SSPSN, SSP -> Bool
_IsTsAssOf_: STsAss, SSP -> Bool                  _IsDataOf_: SData, SSP -> Bool
_IsTokensOf_: STokens, SSP -> Bool                _IsSyncTypeOf_: SSyncType, SSP -> Bool
_IsResynTypeOf_: SResynType, SSP -> Bool          _IsPExcReasonOf_: SPExcReason, SSP -> Bool
_IsUExcReasonOf_: SUExcReason, SSP -> Bool        _IsActIdOf_, _IsOldActIdOf_: SActId, SSP -> Bool
_IsRelResultOf_: SRelResult, SSP -> Bool          _IsPAbReasonOf_: SPAbReason, SSP -> Bool
CRef: SSP -> SCRef                                Called, Calling, Responding: SSP -> SAddress
QOS: SSP -> SQOS                                  ConResult: SSP -> SConResult
Rqms: SSP -> SFUs                                 SPSN: SSP -> SSPSN
TsAss : SSP -> STsAss                             Data: SSP -> SData
Tokens: SSP -> STokens                            SyncType: SSP -> SSyncType
ResynType: SSP -> SResynType                      PExcReason: SSP -> SPExcReason
UExcReason: SSP -> SUExcReason                    ActId, OldActId: SSP -> SActId
```

```
RelResult: SSP -> SRelResult                    PAbReason: SSP -> SPAbReason
eqns forall r,r1:SCRef, sa,cg,cd,rg:SAddress, q,q1:SQOS, cr,cr1:SConResult, rs,rs1:SFUs,
      sn,sn1:SSPSN, a,a1:STsAss, d,d1:SData, ts,ts1:STokens, st,st1:SSyncType, rt,rt1:SResynType,
      per,per1:SPExcReason, uer,uer1,er,er1:SUExcReason, ai,ai1,aio,aio1:SActId, rr,rr1:SRelResult,
      ar,ar1:SPAbReason, p,p1:SSP
ofsort Bool
r1 IsCRefOf SCONreq(r,cg,cd,q,rs,sn,a,d) = r1 eq r;   r1 IsCRefOf SCONind(r,cg,cd,q,rs,sn,a,d) = r1 eq r;
r1 IsCRefOf SCONrsp(r,rg,cr,q,rs,sn,a,d) = r1 eq r;   r1 IsCRefOf SCONcnf(r,rg,cr,q,rs,sn,a,d) = r1 eq r;
r1 IsCRefOf SACTRreq(a1,aio,sn,r,d) = r1 eq r;        r1 IsCRefOf SACTRind(a1,aio,sn,r,d) = r1 eq r;
not (IsSCON(p) or IsSCONAK(p) or IsSACTR(p)) => r IsCRefOf p = false;
sa IsCallingOf SCONreq(r,cg,cd,q,rs,sn,a,d) = sa eq cg;
sa IsCallingOf SCONind(r,cg,cd,q,rs,sn,a,d) = sa eq cg;
not (IsSCON(p)) => sa IsCallingOf p = false;
sa IsCalledOf SCONreq(r,cg,cd,q,rs,sn,a,d) = sa eq cd;
sa IsCalledOf SCONind(r,cg,cd,q,rs,sn,a,d) = sa eq cd;
not (IsSCON(p)) => sa IsCalledOf p = false;
sa IsRespondingOf SCONrsp(r,rg,cr,q,rs,sn,a,d) = sa eq rg;
sa IsRespondingOf SCONcnf(r,rg,cr,q,rs,sn,a,d) = sa eq rg;
not (IsSCONAK(p)) => sa IsCalledOf p = false;
q1 IsQOSOf SCONreq(r,cg,cd,q,rs,sn,a,d) = q1 eq q;
q1 IsQOSOf SCONind(r,cg,cd,q,rs,sn,a,d) = q1 eq q;
q1 IsQOSOf SCONrsp(r,rg,cr,q,rs,sn,a,d) = q1 eq q;
q1 IsQOSOf SCONcnf(r,rg,cr,q,rs,sn,a,d) = q1 eq q;
not (IsSCON(p) or IsSCONAK(p)) => q IsQOSOf p = false;
cr1 IsConResultOf SCONrsp(r,rg,cr,q,rs,sn,a,d) = cr1 eq cr;
cr1 IsConResultOf SCONcnf(r,rg,cr,q,rs,sn,a,d) = cr1 eq cr;
not (IsSCONAK(p)) => cr IsConResultOf p = false;
rs1 IsRqmsOf SCONreq(r,cg,cd,q,rs,sn,a,d) = rs1 eq rs;
rs1 IsRqmsOf SCONind(r,cg,cd,q,rs,sn,a,d) = rs1 eq rs;
rs1 IsRqmsOf SCONrsp(r,rg,cr,q,rs,sn,a,d) = rs1 eq rs;
rs1 IsRqmsOf SCONcnf(r,rg,cr,q,rs,sn,a,d) = rs1 eq rs;
not (IsSCON(p) or IsSCONAK(p)) => rs IsRqmsOf p = false;
sn1 IsSPSNOf SCONreq(r,cg,cd,q,rs,sn,ta,d) = sn1 eq sn;
sn1 IsSPSNOf SCONind(r,cg,cd,q,rs,sn,ta,d) = sn1 eq sn;
sn1 IsSPSNOf SCONrsp(r,rg,cr,q,rs,sn,ta,d) = sn1 eq sn;
sn1 IsSPSNOf SCONcnf(r,rg,cr,q,rs,sn,ta,d) = sn1 eq sn;
sn1 IsSPSNOf SSYNmreq(st,sn,d) = sn1 eq sn;       sn1 IsSPSNOf SSYNmind(st,sn,d) = sn1 eq sn;
sn1 IsSPSNOf SSYNmrsp(sn,d) = sn1 eq sn;          sn1 IsSPSNOf SSYNmcnf(sn,d) = sn1 eq sn;
sn1 IsSPSNOf SSYNMareq(sn,d) = sn1 eq sn;         sn1 IsSPSNOf SSYNMaind(sn,d) = sn1 eq sn;
sn1 IsSPSNOf SRSYNreq(rt,sn,a,d) = sn1 eq sn;     sn1 IsSPSNOf SRSYNind(rt,sn,a,d) = sn1 eq sn;
sn1 IsSPSNOf SRSYNrsp(sn,a,d) = sn1 eq sn;        sn1 IsSPSNOf SRSYNcnf(sn,a,d) = sn1 eq sn;
sn1 IsSPSNOf SACTRreq(ai,aio,sn,r,d) = sn1 eq sn; sn1 IsSPSNOf SACTRind(ai,aio,sn,r,d) = sn1 eq sn;
sn1 IsSPSNOf SACTEreq(sn,d) = sn1 eq sn;          sn1 IsSPSNOf SACTEind(sn,d) = sn1 eq sn
not (IsSCON(p) or IsSCONAK(p) or IsSSYNm(p) or IsSSYNmAK(p) or IsSSYNMa(p) or IsSRSYN(p) or
      IsSRSYNAK(p) or IsSACTR(p) or IsSACTE(p)) => sn IsSPSNOf p = false;
a1 IsTsAssOf SCONreq(r,cg,cd,q,rs,sn,a,d) = a1 eq a;
a1 IsTsAssOf SCONind(r,cg,cd,q,rs,sn,a,d) = a1 eq a;
a1 IsTsAssOf SCONrsp(r,rg,cr,q,rs,sn,a,d) = a1 eq a;
a1 IsTsAssOf SCONcnf(r,rg,cr,q,rs,sn,a,d) = a1 eq a;
a1 IsTsAssOf SRSYNreq(rt,sn,a,d) = a1 eq a;       a1 IsTsAssOf SRSYNind(rt,sn,a,d) = a1 eq a;
a1 IsTsAssOf SRSYNrsp(sn,a,d) = a1 eq a;          a1 IsTsAssOf SRSYNcnf(sn,a,d) = a1 eq a;
not (IsSCON(p) or IsSCONAK(p) or IsSRSYN(p) or IsSRSYNAK(p)) => a IsTsAssOf p = false;
d1 IsDataOf SCONreq(r,cg,cd,q,rs,sn,a,d) = d1 eq d;  d1 IsDataOf SCONind(r,cg,cd,q,rs,sn,a,d) = d1 eq d;
d1 IsDataOf SCONrsp(r,rg,cr,q,rs,sn,a,d) = d1 eq d;  d1 IsDataOf SCONcnf(r,rg,cr,q,rs,sn,a,d) = d1 eq d;
d1 IsDataOf SDTreq(d) = d1 eq d;                  d1 IsDataOf SDTind(d) = d1 eq d;
d1 IsDataOf SEXreq(d) = d1 eq d;                  d1 IsDataOf SEXind(d) = d1 eq d;
d1 IsDataOf STDreq(d) = d1 eq d;                  d1 IsDataOf STDind(d) = d1 eq d;
d1 IsDataOf SCDreq(d) = d1 eq d;                  d1 IsDataOf SCDind(d) = d1 eq d;
d1 IsDataOf SCDrsp(d) = d1 eq d;                  d1 IsDataOf SCDcnf(d) = d1 eq d;
d1 IsDataOf SPTreq(ts,d) = d1 eq d;               d1 IsDataOf SPTind(ts,d) = d1 eq d;
d1 IsDataOf SSYNmreq(st,sn,d) = d1 eq d;          d1 IsDataOf SSYNmind(st,sn,d) = d1 eq d;
```

```
d1 IsDataOf SSYNmrsp(sn,d) = d1 eq d;                d1 IsDataOf SSYNmcnf(sn,d) = d1 eq d;
d1 IsDataOf SSYNMareq(sn,d) = d1 eq d;               d1 IsDataOf SSYNMaind(sn,d) = d1 eq d;
d1 IsDataOf SSYNMarsp(d) = d1 eq d;                  d1 IsDataOf SSYNMacnf(d) = d1 eq d;
d1 IsDataOf SRSYNreq(rt,sn,a,d) = d1 eq d;           d1 IsDataOf SRSYNind(rt,sn,a,d) = d1 eq d;
d1 IsDataOf SRSYNrsp(sn,a,d) = d1 eq d;              d1 IsDataOf SRSYNcnf(sn,a,d) = d1 eq d;
d1 IsDataOf SUERreq(uer,d) = d1 eq d;                d1 IsDataOf SUERind(uer,d) = d1 eq d;
d1 IsDataOf SACTSreq(ai,d) = d1 eq d;                d1 IsDataOf SACTSind(ai,d) = d1 eq d;
d1 IsDataOf SACTRreq(ai,aio,sn,r,d) = d1 eq d;       d1 IsDataOf SACTRind(ai,aio,sn,r,d) = d1 eq d;
d1 IsDataOf SACTEreq(sn,d) = d1 eq d;                d1 IsDataOf SACTEind(sn,d) = d1 eq d;
d1 IsDataOf SACTErsp(d) = d1 eq d;                   d1 IsDataOf SACTEcnf(d) = d1 eq d;
d1 IsDataOf SRELreq(d) = d1 eq d;                    d1 IsDataOf SRELind(d) = d1 eq d;
d1 IsDataOf SRELrsp(rr,d) = d1 eq d;                 d1 IsDataOf SRELcnf(rr,d) = d1 eq d;
d1 IsDataOf SUABreq(d) = d1 eq d;                    d1 IsDataOf SUABind(d) = d1 eq d;
IsSGT(p) or IsSCG(p) or IsSPERind(p) or IsSACTI(p) or IsSACTIAK(p) or IsSACTD(p) or IsSACTDAK(p) or
      IsSPABind(p) => d IsDataOf p = false;
ts1 IsTokensOf SGTreq(ts) = ts1 eq ts;               ts1 IsTokensOf SGTind(ts) = ts1 eq ts;
ts1 IsTokensOf SPTreq(ts,d) = ts1 eq ts;             ts1 IsTokensOf SPTind(ts,d) = ts1 eq ts;
not (IsSGT(p) or IsSPT(p)) => ts IsTokensOf p = false;
st1 IsSyncTypeOf SSYNmreq(st,sn,d) = st1 eq st;      st1 IsSyncTypeOf SSYNmind(st,sn,d) = st1 eq st;
not (IsSSYNm(p)) => st IsSyncTypeOf p =false;
rt1 IsResynTypeOf SRSYNreq(rt,sn,a,d) = rt1 eq rt;   rt1 IsResynTypeOf SRSYNind(rt,sn,a,d) = rt1 eq rt;
not (IsSRSYN(p)) => rt IsResynTypeOf p =false;
per1 IsPExcReasonOf SPERind(per) = per1 eq per;
not (IsSPERind(p)) => per IsPExcReasonOf p =false;
uer1 IsUExcReasonOf SUERreq(uer,d) = uer1 eq uer;
uer1 IsUExcReasonOf SUERind(uer,d) = uer1 eq uer;
er1 IsUExcReasonOf SACTIreq(er) = er1 eq er;         er1 IsUExcReasonOf SACTIind(er) = er1 eq er;
er1 IsUExcReasonOf SACTDreq(er) = er1 eq er;         er1 IsUExcReasonOf SACTDind(er) = er1 eq er;
not (IsSUER(p) or IsSACTI(p) or IsSACTD(p)) => er IsUExcReasonOf p = false;
ai1 IsActIdOf SACTSreq(ai,d) = ai1 eq ai;            ai1 IsActIdOf SACTSind(ai,d) = ai1 eq ai;
ai1 IsActIdOf SACTRreq(ai,aio,sn,r,d) = ai1 eq ai;   ai1 IsActIdOf SACTRind(ai,aio,sn,r,d) = ai1 eq ai;
not (IsSACTS(p) or IsSACTR(p)) => ai IsActIdOf p = false;
aio1 IsOldActIdOf SACTRreq(ai,aio,sn,r,d) = aio1 eq aio;
aio1 IsOldActIdOf SACTRind(ai,aio,sn,r,d) = aio1 eq aio;
not (IsSACTR(p)) => aio IsOldActIdOf p = false;
rr1 IsRelResultOf SRELrsp(rr,d) = rr1 eq rr;         rr1 IsRelResultOf SRELcnf(rr,d) = rr1 eq rr;
not (IsSRELAK(p)) => rr IsRelResultOf p = false;
ar1 IsPAbReasonOf SPABind(ar) = ar1 eq ar;           not (IsSPABind(p)) => ar IsPAbReasonOf p = false;
ofsort SCRef                 r IsCRefOf p => CRef(p) = r;
ofsort SAddress
cg IsCallingOf p => Calling(p) = cg;                 cd IsCalledOf p => Called(p) = cd;
rg IsRespondingOf p => Responding(p) = rg;
ofsort SQOS                  q IsQOSOf p => QOS(p) = q;
ofsort SConResult            cr IsConResultOf p => ConResult(p) = cr;
ofsort SRqms                 rs IsRqmsOf p => Rqms(p) = rs;
ofsort SSPSN                 sn IsSPSNOf p => SPSN(p) = sn;
ofsort STsAss                a IsTsAssOf p => TsAss(p) = a;
ofsort SData                 d IsDataOf p => Data(p) = d;
ofsort STokens               ts IsTokensOf p => Tokens(p) = ts;
ofsort SSyncType             st IsSyncTypeOf p => SyncType(p) = st;
ofsort SResynType            rt IsResynTypeOf p => ResynType(p) = rt;
ofsort SPExcReason           per IsPExcReasonOf p => PExcReason(p) = per;
ofsort SUExcReason           er IsUExcReasonOf p => UExcReason(p) = er;
ofsort SActId
ai IsActIdOf p => ActId(p) = ai;                     aio IsOldActIdOf p => OldActId(p) = aio;
ofsort SRelResult            rr IsRelResultOf p => RelResult(p) = rr;
ofsort SPAbReason            ar IsPAbReasonOf p => PAbReason(p) = ar;
endtype
(*-------------------------------------------------------------------------------------------------------------------------------
```

9.1.3.4 Equality of session service primitives

SSPEquality enriches the SSP construction with boolean equality on SSPs. This is defined as the conjunction of SSP class equality, and pairwise equality of SSP parameters. *IncludesParsOf* is an auxiliary function that checks whether a given SSP carries (includes) the same parameter values as another given SSP, independent of the SSP classes.

```
----------------------------------------------------------------------------------------------------------------------------*)
type SSPEquality is SSPParameterSelectors
opns
_eq_, _ne_, _IncludesParsOf_: SSP, SSP -> Bool
eqns forall p,p1:SSP ofsort Bool
IsSCG(p) or IsSACTIAK(p) or IsSACTDAK(p) => p1 IncludesParsOf p = true;
IsSDT(p) or IsSEX(p) or IsSTD(p) or IsSCD(p) or IsSCDAK(p) or IsSSYNMaAK(p) or
      IsSACTEAK(p) or IsSREL(p) or IsSUAB(p) => p1 IncludesParsOf p = (Data(p) IsDataOf p1);
IsSGT(p) => p1 IncludesParsOf p = (Tokens(p) IsTokensOf p1);
IsSPER(p) => p1 IncludesParsOf p = (PExcReason(p) IsPExcReasonOf p1);
IsSACTI(p) or IsSACTD(p) => p1 IncludesParsOf p = (UExcReason(p) IsUExcReasonOf p1);
IsSPAB(p) => p1 IncludesParsOf p = (PAbReason(p) IsPAbReasonOf p1);
IsSPT(p) => p1 IncludesParsOf p = (Tokens(p) IsTokensOf p1) and (Data(p) IsDataOf p1);
IsSSYNmAK(p) or IsSSYNMa(p) or IsSACTE(p) =>
      p1 IncludesParsOf p = (SPSN(p) IsSPSNOf p1) and (Data(p) IsDataOf p1);
IsSUER(p) => p1 IncludesParsOf p = (UExcReason(p) IsUExcReasonOf p1) and (Data(p) IsDataOf p1);
IsSACTS(p) => p1 IncludesParsOf p = (ActId(p) IsActIdOf p1) and (Data(p) IsDataOf p1);
IsSRELAK(p) => p1 IncludesParsOf p = (RelResult(p) IsRelResultOf p1) and (Data(p) IsDataOf p1);
IsSSYNm(p) => p1 IncludesParsOf p =
      (SyncType(p) IsSyncTypeOf p1) and (SPSN(p) IsSPSNOf p1) and (Data(p) IsDataOf p1);
IsSRSYNAK(p) => p1 IncludesParsOf p = (SPSN(p) IsSPSNOf p1)) and
      (TsAss(p) IsTsAssOf p1) and (Data(p) IsDataOf p1);
IsSRSYN(p) => p1 IncludesParsOf p = (ResynType(p) IsResynTypeOf p1) and
      (SPSN(p) IsSPSNOf p1) and (TsAss(p) IsTsAssOf p1) and (Data(p) IsDataOf p1);
IsACTR(p) => p1 IncludesParsOf p = (ActId(p) IsActIdOf p1) and
      (OldActId(p) IsOldActIdOf p1) and (SPSN(p) IsSPSNOf p1) and
      (CRef(p) Is CRefOf p1) and (Data(p) IsDataOf p1);
IsSCON(p) => p1 IncludesParsOf p = (CRef(p) IsCRefOf p1) and
      (Calling(p) IsCallingOf p1) and (Called(p) IsCalledOf p1) and
      (QOS(p) IsQOSOf p1) and (Rqms(p) IsRqmsOf p1) and (SPSN(p) IsSPSNOf p1) and
      (TsAss(p) IsTsAssOf p1) and (Data(p) IsDataOf p1);
IsSCONAK(p) => p1 IncludesParsOf p = (CRef(p) IsCRefOf p1) and
      (Called(p) IsCalledOf p1) and (ConResult(p) IsConResultOf p1) and
      (QOS(p) IsQOSOf p1) and (Rqms(p) IsRqmsOf p1) and (SPSN(p) IsSPSNOf p1) and
      (TsAss(p) IsTsAssOf p1) and (Data(p) IsDataOf p1);
p eq p1 = (k(p) eq k(p1)) and (p IncludesParsOf p1);
p ne p1 = not (p eq p1);
endtype
(*----------------------------------------------------------------------------------------------------------------------------
```


9.1.3.5 Other functions on session service primitives

The final functional enrichment for the construction of SSPs is presented by type *SessionServicePrimitive*. It defines boolean functions for testing restrictions on SSP parameter values which are useful to represent negotiation (see 9.2.3) and nondeterminacy of the SS-provider (see 10.2).

IsIndicationOf relates the parameter values in an indication or confirm to those in a corresponding request or response, respectively. It yields true if the parameter values of its argument SSPs are equal or satisfy the negotiation requirements stated in subclause 10.2 of ISO 8326. Similarly, *IsConResponseOf* relates the parameter values in a S-CONNECT response to those in a corresponding S-CONNECT indication. The remaining functions, with a single argument SSP, test whether boundary values for parameters are not

exceeded and mutual dependencies between parameter values are not violated in the SSP. The constants used in the definitions of these functions are defined by type *NatConstant* (see 9.1.5.2).

```
----------------------------------------------------------------------------------------------------*)
type SessionServicePrimitive is SSPEquality, NatConstant
opns
_IsIndicationOf_, _IsConResponseOf_: SSP, SSP -> Bool
IsValidSCONreq, IsValidSCONrsp, IsValidSDT, IsValidSEX, IsValidSTD, IsValidSCD, IsValidSCDAK,
IsValidSGT, IsValidSPT, IsValidSSYNm, IsValidSSYNmAK, IsValidSSYNMa, IsValidSSYNMaAK,
IsValidSRSYN, IsValidSRSYNAK, IsValidSUER, IsValidSACTS, IsValidSACTR, IsValidSACTE,
IsValidSACTEAK, IsValidSREL, IsValidSRELAK, IsValidSUAB: SSP -> Bool
eqns forall r,r1:SCRef, cg,cg1,cd,cd1,rg:SAddress, q,q1:SQOS, cr:SConResult, rs,rs1:SFUs,
     sn,sn1:SSPSN, a,a1:STsAss, d,d1:SData, ts:STokens, st:SSyncType, rt:SResynType, ai,aio:SActId,
     rr:SRelResult, p,p1:SSP
ofsort Bool
SCONind(r1,cg1,cd1,q1,rs1,sn1,a1,d1) IsIndicationOf SCONreq(r,cg,cd,q,rs,sn,a,d) =
     (r1 eq r) and (cg1 eq cg) and (cd1 eq cd) and (q1 le q) and (rs1 eq rs) and (sn1 eq sn) and
     (a1 eq a) and (d1 eq d);
not (IsSCONreq(p)) =>
     p1 IsIndicationOf p = IsReq(p) and (h(k(p1)) eq Succ(h(k(p)))) and (p1 IncludesParsOf p);
IsSCONreq(p) and not(IsSCONind(p1) => p1 IsIndicationOf p = false

SCONrsp(r1,rg,cr,q1,rs1,sn1,a1,d1) IsConResponseOf SCONind(r,cg,cd,q,rs,sn,a,d) =
     ((cr eq agree) implies (q1 le q)) and
     (((HD IsIn rs) and (FD NotIn rs)) implies (HD IsIn rs1)) and
     (((HD NotIn rs) and (FD IsIn rs)) implies (FD IsIn rs1)) and
     ((CD IsIn (rs1 Ints rs)) implies (ACT IsIn (rs1 Ints rs))) and
     ((EXCEP IsIn (rs Ints rs1)) implies (HD IsIn (rs Ints rs1))) and
     ((RqrTokens(a1) Union AcrTokens(a1)) eq ChoiceTokens(a));
not (IsSCONrsp(p1) and IsSCONind(p)) => p1 IsConResponseOf p = false;

IsValidSCONreq (SCONreq (r,cg,cd,q,rs,sn,a,d)) =
     ( (Length(CgUserRef(r)) le 64 ) and (Length(CgUserRef(r)) gt 0) and
     ( Length(CdUserRef(r)) le 64 ) and (Length(CdUserRef(r)) gt 0) and
     ( Length(CommonRef(r)) le 64 ) and (Length(CommonRef(r)) gt 0) and
     ( Length(AdditionalRef(r)) le 4 ) and
     ( not ((SY IsIn rs) or (MA IsIn rs) or (RESYN IsIn rs)) implies (sn eq absent) ) and
     ( (((SY IsIn rs) or (MA IsIn rs) or (RESYN IsIn rs)) and (ACT NotIn rs)) implies (sn ne absent) ) and
     ( (sn le s(999999)) or (sn eq absent) ) and
     ( (HD IsIn rs) implies (dk IsIn Tokens(a)) ) and ( (NR IsIn rs) implies (tr IsIn Tokens(a)) ) and
     ( (SY IsIn rs) implies (mi IsIn Tokens(a)) ) and
     ( ((MA IsIn rs) or (ACT IsIn rs)) implies (ma IsIn Tokens(a)) ) and
     ( (RqrTokens(a) Ints AcrTokens(a)) eq {} ) and ( (RqrTokens(a) Ints ChoiceTokens(a)) eq {} ) and
     ( (AcrTokens(a) Ints ChoiceTokens(a)) eq {} ) and
     ( Length(d) le 512 ) );
not (IsSCONreq(p)) => IsValidSCONreq(p) = false;

IsValidSCONrsp (SCONrsp (r,rg,cr,q,rs,sn,a,d)) =
     ( (Length(CgUserRef(r)) le 64 ) and (Length(CgUserRef(r)) gt 0) and
     ( Length(CdUserRef(r)) le 64 ) and (Length(CdUserRef(r)) gt 0) and
     ( Length(CommonRef(r)) le 64 ) and (Length(CommonRef(r)) gt 0) and
     ( Length(AdditionalRef(r)) le 4 ) and
     ( not (IsPReject(cr)) ) and
     ( not ((HD IsIn rs) and (FD IsIn rs)) ) and
     ( (((SY NotIn rs) and (MA NotIn rs) and (RESYN NotIn rs)) or (ACT IsIn rs))
          implies (sn eq absent) ) and
     ( (((SY IsIn rs) or (MA IsIn rs) or (RESYN IsIn rs)) and (ACT NotIn rs)) implies (sn ne absent) ) and
     ( (sn le s(999999)) or (sn eq absent) ) and
     ( (RqrTokens(a) Ints AcrTokens(a)) eq {} ) and ( ChoiceTokens(a)) eq {} ) and
     ( Length(d) le 512 ) );
```

```
not (IsSCONrsp(p)) => IsValidSCONrsp(p) = false;

IsValidSDT(p) = IsSDT(p) and (Length(Data(p)) gt 0);
IsValidSEX(p) = IsSEX(p) and (Length(Data(p)) gt 0) and (Length(Data(p)) le 14);
IsValidSTD(p) = IsSTD(p) and (Length(Data(p)) gt 0);
IsValidSCD(p) = IsSCD(p) and (Length(Data(p)) le 512);
IsValidSCDAK(p) = IsSCDAK(p) and (Length(Data(p)) le 512);
IsValidSGT(p) = IsSGT(p) and (Tokens(p) ne {});
IsValidSPT(p) = IsSPT(p) and (Length(Data(p)) le 512) and (Tokens(p) ne {});
IsValidSSYNm(p) = IsSSYNm(p) and (Length(Data(p)) le 512) and (SSPN(p) le s(999998));
IsValidSSYNmAK(p) = IsSSYNmAK(p) and (Length(Data(p)) le 512) and (SSPN(p) le s(999998));
IsValidSSYNMa(p) = IsSSYNMa(p) and (Length(Data(p)) le 512) and (SSPN(p) le s(999998));
IsValidSSYNMaAK(p) = IsSSYNMaAK(p) and (Length(Data(p)) le 512) and (SSPN(p) le s(999998));
IsValidSRSYN(p) = IsSRSYN(p) and
      ( (ResynType(p) eq a) implies (SPSN(p) eq absent) ) and
      ( (SPSN(p) eq absent) or (SPSN(p) le s(999999)) ) and
      ( (RqrTokens(TsAss(p)) Ints AcrTokens(TsAss(p))) eq {} ) and
      ( (RqrTokens(TsAss(p)) Ints ChoiceTokens(TsAss(p))) eq {} ) and
      ( (AcrTokens(TsAss(p)) Ints ChoiceTokens(TsAss(p))) eq {} ) and
      (Length(Data(p)) le 512);
IsValidSRSYNAK(p) = IsSRSYNAK(p) and
      ( (SPSN(p) le s(999999)) ) and
      ( (RqrTokens(TsAss(p)) Ints AcrTokens(TsAss(p))) eq {} ) and ( ChoiceTokens(TsAss(p)) eq {} ) and
      ( Length(Data(p)) le 512 );
IsValidSUER(p) = IsSUER(p) and (Length(Data(p)) le 512);
IsValidSACTS(p) = IsSACTS(p) and (Length(Data(p)) le 512) and (Length(ActId(p)) gt 0) and
      (Length(ActId(p)) le 6);
IsValidSACTR(p) = IsSACTR(p) and
      (Length(ActId(p)) gt 0) and (Length(ActId(p)) le 6) and
      (Length(OldActId(p)) gt 0) and (Length(OldActId(p)) le 6) and
      (SPSN(p) le s(999998)) and (Length(CgUserRef(CRef(p))) le 64) and
      (Length(CgUserRef(CRef(p))) gt 0) and (Length(CdUserRef(CRef(p))) le 64) and
      (Length(CdUserRef(CRef(p))) gt 0) and (Length(CommonRef(CRef(p))) le 64) and
      (Length(CommonRef(CRef(p))) gt 0) and
      (Length(AdditionalRef(CRef(p))) le 4) and (Length(Data(p)) le 512);
IsValidSACTE(p) = IsSACTE(p) and (Length(Data(p)) le 512) and (SPSN(p) le s(999998));
IsValidSACTEAK(p) = IsSACTEAK(p) and (Length(Data(p)) le 512);
IsValidSREL(p) = IsSREL(p) and (Length(Data(p)) le 512);
IsValidSRELAK(p) = IsSRELAK(p) and (Length(Data(p)) le 512);
IsValidSUAB(p) = IsSUAB(p) and (Length(Data(p)) le 9);
endtype
(*-----------------------------------------------------------------------------------------------------------------------
```

9.1.4 Session service primitive parameters

Values of the Calling, Called and Responding Session Address parameter are of sort *SAddress* (see 9.1.1). The other SSP parameters are defined by the data type definitions in the following subclauses. In a number of definitions a mapping function *h* is used in a similar way as in 9.1.3.1 to simplify the specification of boolean operations for comparing two values of some sort.

9.1.4.1 (Old) Session Connection Identifier parameter

The Session Connection Identifier and Old Session Connection Identifier parameters are defined by type *SCIdentifier*. Values of these parameters are represented as sequences of four component octet strings. The length restrictions imposed on these octet strings are defined in 9.1.3.5.

```
----------------------------------------------------------------------------------------------------------------------*)
type SCIdentifier is SCReferenceElement
sorts SCRef
opns
ConnectionRef: SCRefEl, SCRefEl, SCRefEl, SCRefEl -> SCRef
CgUserRef, CdUserRef, CommonRef, AdditionalRef: SCRef -> SCRefEl
_eq_, _ne_: SCRef, SCRef -> Bool
eqns forall cgr,cgr1,cdr,cdr1,cr,cr1,ar,ar1:SCRefEl, r,r1:SCRef
ofsort SCRefEl
CgUserRef (ConnectionRef (cgr,cdr,cr,ar)) = cgr;      CdUserRef (ConnectionRef (cgr,cdr,cr,ar)) = cdr;
CommonRef (ConnectionRef (cgr,cdr,cr,ar)) = cr;       AdditionalRef (ConnectionRef (cgr,cdr,cr,ar)) = ar;
ofsort Bool
ConnectionRef (cgr1,cdr1,cr1,ar1) eq ConnectionRef (cgr,cdr,cr,ar) =
      (cgr1 eq cgr) and (cdr1 eq cdr) and (cr1 eq cr) and (ar1 eq ar);
r1 ne r = not (r1 eq r);
endtype

type SCReferenceElement is OctetString renamedby sortnames SCRefEl for OctetString endtype
(*----------------------------------------------------------------------------------------------------------------------
```

9.1.4.2 Result parameter for the Session Connection service

The Result parameter which is employed in the Session Connection service is defined by type *SConnectResult*. Parameter values are represented by constants that correspond with the values defined in subclause 12.1.2 of ISO 8326 (*agree* represents the value "accept", since ***accept*** is a keyword in LOTOS). Three "recognizer" functions are defined for testing whether a given Result value indicates acceptence, rejection, and rejection by the SS-provider respectively.

```
----------------------------------------------------------------------------------------------------------------------*)
type SConnectResult is Boolean, NaturalNumber
sorts SConResult
opns
agree, ureject1, ureject2, ureject3, preject1, preject2, preject3, preject4: ->SConResult
IsAccept, IsReject, IsPReject: SConResult -> Bool
h: SConResult -> Nat
_eq_, _ne_: SConResult, SConResult -> Bool
eqns forall r,r1:SConResult
ofsort Bool
IsAccept (r) = r eq agree;            IsReject (r) = r ne agree;
IsPReject (r) = (r eq preject1) or (r eq preject2) or (r eq preject3) or (r eq preject4);
r1 eq r = h(r1) eq h(r);              r1 ne r = not (r1 eq r);
ofsort Nat
h(agree) = 0;                         h(ureject1) = Succ(h(agree));     h(ureject2) = Succ(h(ureject1));
h(ureject3) = Succ(h(ureject2));      h(preject1) = Succ(h(ureject3));  h(preject2) = Succ(h(preject1));
h(preject3) = Succ(h(preject2));      h(preject4) = Succ(h(preject3));
endtype
(*----------------------------------------------------------------------------------------------------------------------
```

9.1.4.3 Quality of Service parameter

The structure of the QOS parameter is defined in clause 10 of ISO 8326. In that clause, the QOS component parameters are classified as either Performance, Extended Control, SC Protection, SC Priority, or Optimized Dialogue Transfer parameters. This structure is reflected by type *SCQuality*. Two ordered constants are defined for both the Extended Control and Optimized Dialogue Transfer component parameter values, corresponding with the values "feature desired" and "feature not desired" in ISO 8326 (see 9.1.5.1 for the definition of *Doublet*). The QOS component parameters for the classes Performance, SC Protection, and SC Priority are not modified by the Session Protocol, but are mapped onto the corresponding parameters of the Transport Service. The associated definitions of the formal description of the Transport Service, ISO/IEC/TR 10023, are renamed for use in this specification.

```
----------------------------------------------------------------------------------------------------------------------*)
type SCQuality is
SCPerformance, SCExtendedControl, SCProtection, SCPriority, SCOptimizedDialogueTransfer
sorts SQOS
opns
ConnectionQOS: SCPerformance, SCExtControl, SCProtection, SCPriority, SCOptTransfer -> SQOS
Performance: SQOS -> SCPerformance                ExtControl: SQOS -> SCExtControl
Protection: SQOS -> SCProtection                  Priority: SQOS -> SCPriority
OptTransfer: SQOS -> SCOptTransfer                _eq_, _ne_, _le_: SQOS, SQOS -> Bool
eqns forall pf,pf1:SCPerformance, ec,ec1:SCExtControl, pt,pt1:SCProtection, pr,pr1:SCPriority,
      ot,ot1:SCOptTransfer, q,q1:SQOS
ofsort SCPerformance     Performance(ConnectionQOS (pf,ec,pt,pr,ot)) = pf;
ofsort SCExtControl      ExtControl (ConnectionQOS (pf,ec,pt,pr,ot)) = ec;
ofsort SCProtection      Protection (ConnectionQOS (pf,ec,pt,pr,ot)) = pt;
ofsort SCPriority        Priority (ConnectionQOS (pf,ec,pt,pr,ot)) = pr;
ofsort SCOptTransfer     OptTransfer (ConnectionQOS (pf,ec,pt,pr,ot)) = ot;
ofsort Bool
ConnectionQOS(pf1,ec1,pt1,pr1,ot1) le ConnectionQOS(pf,ec,pt,pr,ot)) =
      (pf1 le pf) and (ec1 le ec) and (pt1 le pt) and (pr1 le pr) and (ot1 le ot);
q1 eq q = (q1 le q) and (q le q1);                q1 ne q = not (q1 eq q);
endtype

type SCExtendedControl is SCQualityComponent renamedby
sortnames SCExtControl for SCQualityComp
endtype

type SCOptimizedDialogueTransfer is SCQualityComponent renamedby
sortnames SCOptTransfer for SCQualityComp
endtype

type SCQualityComponent is SCQualityComponent0
opns _le_: SCQualityComp, SCQualityComp -> Bool
eqns forall c:SCQualityComp ofsort Bool
c le c = true;                    notDesired le desired = true;        desired le notDesired = false;
endtype

type SCQualityComponent0 is Doublet renamedby
sortnames SCQualityComp for Doublet
opnnames   desired for constant1          notDesired for constant2
endtype

type SCPerformance is TCPerformance renamedby sortnames SCPerformance for TCPerformance
endtype
type SCProtection is TCProtection renamedby sortnames SCProtection for TCProtection endtype
type SCPriority is TCPriority renamedby sortnames SCPriority for TCPriority endtype
(*----------------------------------------------------------------------------------------------------------------------
```

9.1.4.4 Session Requirements parameter

The Session Requirements parameter is represented by an actualization of the generic type *Set*. Type *SFunctionalUnit* defines the element values of the set; these are constants, each denoting an abbreviated functional unit name in correspondence with subclause A.5.1 of ISO 8326.

```
----------------------------------------------------------------------------------------------------------------------------*)
type SRequirements is Set actualizedby SFunctionalUnit, Boolean using
sortnames   SFUs for Set                    SFU for Element                 Bool for FBool
endtype

type SFunctionalUnit is NaturalNumber
sorts SFU
opns
FD, HD, EXCEP, TD, NR, SY, MA, RESYN, EX, ACT, CD: -> SFU
h: SFU -> Nat                               _eq_, _ne_: SFU, SFU -> Bool
eqns forall f,f1:SFU
ofsort Nat
h(FD) = 0;                                  h(HD) = Succ(h(FD));            h(EXCEP) = Succ(h(HD));
h(TD) = Succ(h(EXCEP));                     h(NR) = Succ(h(TD));            h(SY) = Succ(h(NR));
h(MA) = Succ(h(SY));                        h(RESYN) = Succ(h(MA));         h(EX) = Succ(h(RESYN));
h(ACT) = Succ(h(EX));                       h(CD) = Succ(h(ACT));
ofsort Bool
f1 eq f = h(f1) eq h(f);                    f1 ne f = not (f1 eq f);
endtype
(*----------------------------------------------------------------------------------------------------------------------------
```

9.1.4.5 (Initial) Synchronization Point Serial Number parameter

The Synchronization Point Serial Number and the Initial Synchronization Point Serial Number parameter values are defined by a mapping from natural numbers. Two mapping functions, *h* and *s*, are defined; *h* performs a mapping from synchronization numbers to natural numbers and *s* performs the reverse mapping. The constant *absent* represents the absence of a synchronization number (the presence of both parameters in the relevant SSPs is conditional). It maps onto the constant representing "1.000.000", a number which is outside the range used to represent synchronization numbers in SSPs. Restrictions on the range of synchronization numbers are defined in 9.1.3.5.

Next is a function that produces the successor of a given Synchronization Point Serial Number value. A number of boolean functions are defined for comparing two synchronization numbers.

```
----------------------------------------------------------------------------------------------------------------------------*)
type SSynchronizationNumber is NatRepresentations
sorts SSPSN
opns
absent: -> SSPSN                            h: SSPSN -> Nat                 s: Nat -> SSPSN
Next: SSPSN -> SSPSN                        _eq_, _ne_, _lt_, _le_, _gt_, _ge_: SSPSN, SSPSN -> Bool
eqns forall n:Nat, sn,sn1:SSPSN
ofsort Nat
h(s(n)) = n;                                h(absent) = NatNum(1 + (0 + (0 + (0 + (0 + (0 + Dec(0)))))));
ofsort SSPSN
Next (sn) = s (Succ (h (sn)));
ofsort Bool
sn1 eq sn = h(sn1) eq h(sn);                sn1 ne sn = h(sn1) ne h(sn);    sn1 lt sn = h(sn1) lt h(sn);
sn1 le sn = h(sn1) le h(sn);                sn1 gt sn = h(sn1) gt h(sn);    sn1 ge sn = h(sn1) ge h(sn);
endtype
(*----------------------------------------------------------------------------------------------------------------------------
```

9.1.4.6 Initial Assignment of Tokens parameter

The structure defined for the Initial Assignment of Tokens parameter, and the Tokens parameter used in the Resynchronize service, is that of three component sets that respectively contain the tokens which are assigned to the requestor, the acceptor, and either the requestor or acceptor ("acceptor chooses"). Type *STokensAssignment* defines values with this structure. It imports type *STokens*, which defines the representation of sets of tokens (see 9.1.4.8). Static restrictions on the contents of token sets in SSPs are defined in 9.1.3.5.

```
----------------------------------------------------------------------------------------------------------------------*)
type STokensAssignment is STokens
sorts STsAss
opns
TsAssignment: STokens, STokens, STokens -> STsAss
RqrTokens, AcrTokens, ChoiceTokens, Tokens: STsAss -> STokens
_eq_, _ne_: STsAss, STsAss -> Bool
eqns forall rts,rts1,ats,ats1,cts,cts1:STokens, a,a1:STsAss
ofsort STokens
RqrTokens (TsAssignment (rts,ats,cts)) = rts;
AcrTokens (TsAssignment (rts,ats,cts)) = ats;
ChoiceTokens (TsAssignment (rts,ats,cts)) = cts;
Tokens (a) = RqrTokens (a) Union (AcrTokens (a) Union ChoiceTokens (a));
ofsort Bool
TsAssignment(rts1,ats1,cts1) eq TsAssignment(rts,ats,cts) =
      (rts1 eq rts) and (ats1 eq ats) and (cts1 eq cts);
a1 ne a = not (a1 eq a);
endtype
(*----------------------------------------------------------------------------------------------------------------------
```

9.1.4.7 SS-user Data parameter

SS-user Data parameter values are represented as octet strings, defined by a renaming of standard type *OctetString*. Length restrictions on SS-user Data in SSPs are defined in 9.1.3.5.

```
----------------------------------------------------------------------------------------------------------------------*)
type SData is OctetString renamedby sortnames SData for OctetString endtype
(*----------------------------------------------------------------------------------------------------------------------
```

9.1.4.8 Tokens parameter

Values of the Tokens parameter (except those of the Tokens parameter of the Resynchronize service, see 9.1.4.6) are defined by actualizing the generic type *Set*. The elements of the set are defined by *SToken*. This latter type is a renaming of type *Quartet* (see 9.1.5.1), which defines a quartet with boolean equality. The constants correspond with the abbreviations for token names defined in subclause A.5.2 of ISO 8326. The constant *tkdom* is defined in an enrichment to represent the set consisting of all tokens.

```
----------------------------------------------------------------------------------------------------------------------*)
type STokens is STokens0
opns tkdom: -> STokens
eqns ofsort STokens
tkdom = Insert(dk,Insert(mi,Insert(ma,Insert(tr,{}))));
endtype

type STokens0 is Set actualizedby SToken, Boolean using
sortnames   STokens for Set          SToken for Element          Bool for FBool
endtype
```

```
type SToken is Quartet renamedby
sortnames SToken for Quartet
opnnames
dk for constant1        mi for constant2        ma for constant3        tr for constant4
endtype
(*----------------------------------------------------------------------------------------------------------------------
```

9.1.4.9 Type parameter for the Minor Synchronization Point service

Values of the Type parameter used in the Minor Synchronization Point service are defined by a renaming of type *Doublet* (see 9.1.5.1), which defines a doublet with boolean equality. The constants correspond with the values defined in subclause 13.8.2 of ISO 8326.

```
----------------------------------------------------------------------------------------------------------------------*)
type SSyncMinorType is Doublet renamedby
sortnames SSyncType for Doublet
opnnames   explicit for constant1          optional for constant2
endtype
(*----------------------------------------------------------------------------------------------------------------------
```

9.1.4.10 Resynchronize Type parameter

Values of the Resynchronize Type parameter are defined by a renaming of type *Triplet* (see 9.1.5.1), which defines a triplet with boolean equality. The constants *a*, *r* and *s*, respectively, correspond with the values "abandon", "restart" and "set" defined in subclause 13.10.2 of ISO 8326.

```
----------------------------------------------------------------------------------------------------------------------*)
type SResynchronizeType is Triplet renamedby
sortnames SResynType for Triplet
opnnames   a for constant1          r for constant2          s for constant3
endtype
(*----------------------------------------------------------------------------------------------------------------------
```

9.1.4.11 Reason parameter for the P-Exception Reporting service

Values of the Reason parameter used in the P-Exception Reporting service are defined by a renaming of type *Doublet* (see 9.1.5.1). The constants correspond with the values defined in subclause 13.11.2 of ISO 8326.

```
----------------------------------------------------------------------------------------------------------------------*)
type SPExceptionReason is Doublet renamedby
sortnames SPExcReason for Doublet
opnnames   protocolError for constant1        nonSpecificError for constant2
endtype
(*----------------------------------------------------------------------------------------------------------------------
```

9.1.4.12 Reason parameter for the U-Exception Reporting, Activy Interrupt, and Activity Discard services

Values of the Reason parameter used in the U-Exception Reporting service, the Activity Interrupt service and the Activity Discard service are defined by a renaming of the type *Septet* (see 9.1.5.1), which defines a septet with boolean equality. The constants correspond with the values defined in subclauses 13.12.1, 13.15.2 and 13.16.2 of ISO 8326, except *absent* which represents the absence of this parameter (presence of the parameter is a user option in the Activity Interrupt and Activity Discard services).

```
----------------------------------------------------------------------------------------------------------------------------*)
type SUExceptionReason is Septet renamedby
sortnames SUExcReason for Septet
opnnames
receiptProblem for constant1        localError for constant2        sequenceError for constant3
demandDk for constant4              unrecoverableError for constant5  nonSpecificError for constant6
absent for constant7
endtype
(*----------------------------------------------------------------------------------------------------------------------------
```

9.1.4.13 (Old) Activity Identifier parameter

Activity Identifier and Old Activity Identifier parameter values are represented as octet strings, defined by a renaming of standard type *OctetString*. Length restrictions on these parameters in SSPs are defined in 9.1.3.5.

```
----------------------------------------------------------------------------------------------------------------------------*)
type SActivityIdentifier is OctetString renamedby sortnames SActId for OctetString endtype
(*----------------------------------------------------------------------------------------------------------------------------
```

9.1.4.14 Result parameter for the Orderly Release service

Values of the Result parameter used in the Orderly Release service are defined by a renaming of type *Doublet* (see 9.1.5.1). The constants correspond with the values defined in subclause 14.1.2 of ISO 8326.

```
----------------------------------------------------------------------------------------------------------------------------*)
type SReleaseResult is Doublet renamedby
sortnames SRelResult for Doublet
opnnames   affirmative for constant1        negative for constant2
endtype
(*----------------------------------------------------------------------------------------------------------------------------
```

9.1.4.15 Reason parameter for the P-Abort service

Values of the Reason parameter used in the P-Abort service are defined by a renaming of type *Triplet* (see 9.1.5.1). The constants correspond with the values defined in subclause 14.3.2 of ISO 8326.

```
----------------------------------------------------------------------------------------------------------------------------*)
type SPAbortReason is Triplet renamedby
sortnames SPAbReason for Triplet
opnnames
transportDisconnect for constant1   protocolError for constant2        undefined for constant3
endtype
(*----------------------------------------------------------------------------------------------------------------------------
```

9.1.5 Auxiliary data types

9.1.5.1 Doublet, triplet, quartet and septet

Doublet defines a set of two constants, endowed with boolean equality. Sets with three, four and seven constants are also defined, by conservatively extending the immediate smaller set. These types can be used in the definition of sets of small number of values (as defined by some SSP parameters, see 9.1.4).

```
----------*)
type Doublet is Boolean
sorts Doublet
opns
constant1, constant2: -> Doublet      _eq_, _ne_: Doublet, Doublet -> Bool
eqns forall x,y:Doublet ofsort Bool
x eq x = true;                        constant1 eq constant2 = false;
constant2 eq constant1 = false;       x ne y = not (x eq y);
endtype

type Triplet is ExtendDoublet
opns constant3: -> Triplet
eqns forall x:Triplet ofsort Bool
constant3 eq constant1 = false;       constant3 eq constant2 = false;       x eq constant3 = constant3 eq x;
endtype

type ExtendDoublet is Doublet renamedby sortnames Triplet for Doublet endtype

type Quartet is ExtendTriplet
opns constant4: -> Quartet
eqns forall x:Quartet ofsort Bool
constant4 eq constant1 = false;       constant4 eq constant2 = false;
constant4 eq constant3 = false;       x eq constant4 = constant4 eq x;
endtype

type ExtendTriplet is Triplet renamedby sortnames Quartet for Triplet endtype

type Septet is ExtendQuartet
opns constant5, constant6, constant7: -> Septet
eqns forall x:Septet ofsort Bool
constant5 eq constant1 = false;       constant5 eq constant2 = false;       constant5 eq constant3 = false;
constant5 eq constant4 = false;       constant6 eq constant1 = false;       constant6 eq constant2 = false;
constant6 eq constant3 = false;       constant6 eq constant4 = false;       constant6 eq constant5 = false;
constant7 eq constant1 = false;       constant7 eq constant2 = false;       constant7 eq constant3 = false;
constant7 eq constant4 = false;       constant7 eq constant5 = false;       constant7 eq constant6 = false;
x eq constant5 = constant5 eq x;      x eq constant6 = constant6 eq x;      x eq constant7 = constant7 eq x;
endtype

type ExtendQuartet is Quartet renamedby sortnames Septet for Quartet endtype
(*----------
```

9.1.5.2 Natural number constants

Type *NatConstant* defines a number of natural number constants. These constants are used in the specification of restrictions on SSP parameters (see 9.1.3.5), e.g. the maximum length of SS-user Data or the maximum Synchronization Point Serial Number which can be carried in a SSP.

```
----------*)
type NatConstant is NatRepresentations
opns 1, 4, 6, 9, 14, 64, 512, 999998, 999999: -> Nat
eqns ofsort Nat
1 = Succ(0);                                  4 = NatNum(Dec(4));
6 = NatNum(Dec(6));                           9 = NatNum(Dec(9));
14 = NatNum(1 + Dec(4));                      64 = NatNum(6 + Dec(4));
512 = NatNum(5 + (1 + Dec(2)));               999998 = NatNum(9 + (9 + (9 + (9 + (9 + Dec(8))))));
999999 = Succ(999998);
endtype
(*----------
```

9.2 Processes for local constraints

9.2.1 Endpoint identification and local behavioural constraints

Two components of the local constraints are separated in the definition of *SCEP*: those of the identification of a SCEP and those concerning the ordering of, and other constraints on, the execution of SSPs at that endpoint.

The behavioural constraints are specified by process *SCEPBehaviour*. Process *SCEPIdentification* specifies the constraints on the address and SCEI parts of SSP events at a SCEP: both identifiers are established in the first event associated with the SC and remain thereafter constant. *SCEPBehaviour* enables the non-terminating instance of *SCEPIdentification* to be disabled.

```
----------------------------------------------------------------------------------------------------*)
process SCEP [s] (role:SSUserRole): exit :=
( SCEPIdentification [s] [> exit ) || SCEPBehaviour [s] (role)
endproc

process SCEPIdentification [s]: noexit :=
s ?sa:SAddress ?si:SCEI ?p:SSP; ConstantSCEI [s] (sa, si)
where
process ConstantSCEI [s] (sa:SAddress, si:SCEI): noexit :=
s !sa !si ?p:SSP; ConstantSCEI [s] (sa, si)
endproc
endproc (* SCEPIdentification *)
(*----------------------------------------------------------------------------------------------------
```

9.2.2 Behavioural constraints at a SCEP

The constraints on the possible behaviour at an individual SCEP are defined by annex A (the state tables) of ISO 8326. Process *SCEPBehaviour* is the formal reflection of this annex.

The SC establishment phase is specified as the sequence of process *SCEPConnectPropose* and the choice of processes *SCEPConnectAccept* and *SCEPConnectReject*. *SCEPConnectPropose* describes the local execution of the first SSP associated with the SC. *SCEPConnectAccept* describes the acceptance of the SC by the second SSP; *SCEPConnectReject* describes the rejection of the SC by the second SSP.

In case of acceptance of the SC, the data transfer phase, specified by process *SCEPDataTransfer*, is entered. The negotiated values of a number of session variables are passed to this process by *SCEPConnectAccept*, including the set of functional units selected and the assignment of tokens. Type *SLocalTokensState* is an ad-hoc data type, introduced to represent the availability and assignment of tokens. It defines functions that correspond with the functions defined in subclauses A.5.2 and A.5.3 of ISO 8326. Two additional functions, *RqrLTsState* and *AcrLTsState*, are defined that derive the local tokens state at the requestor side and at the acceptor side, respectively, from a negotiated initial assignment of tokens (the two arguments represent: tokens assigned to the requestor side, and tokens assigned to the acceptor side). Another two functions, *MakeMySide* and *MakeYrSide*, enable to change the assignment of tokens.

Process *SCEPAbort* specifies the disruptive abortion of a SC, which may happen at any point in time after the execution of the first SSP. A SC establishment attempt is not necessarily visible at the called end. It may be prematurely aborted (by the calling SS-user or the SS-provider) or refused (by the SS-provider). This is represented by the choice of ***exit*** in process *SCEPBehaviour*.

```
----------------------------------------------------------------------------------------------------------------------*)
type SLocalTokensState is STokens
sorts SLTsState
opns
LTsState: STokens, STokens, STokens -> SLTsState
MyTokens, YrTokens, NotAvTokens: SLTsState -> STokens
RqrLTsState, AcrLTsState: STokens, STokens -> SLTsState
MakeMySide, MakeYrSide: STokens, SLTsState -> SLTsState
AllI, AllA, AllII, AllAA, AnyI, AnyA, AnyII, AnyAA: STokens, SLTsState -> Bool
IsMySide, IsYrSide, IsNotAv: SToken, SLTsState -> Bool
eqns forall t:SToken, ts,mts,yts,nts,ats,rts:STokens, lts:SLTsState
ofsort STokens
MyTokens (LTsState (mts, yts, nts)) = mts;          YrTokens (LTsState (mts, yts, nts)) = yts;
NotAvTokens (LTsState (mts, yts, nts)) = nts;
ofsort SLTsState
RqrLTsState (ats, rts) = LTsState (ats, rts, tkdom Minus (ats Union rts));
AcrLTsState (ats, rts) = LTsState (rts, ats, tkdom Minus (ats Union rts));
MakeMySide (ts, lts) =
      LTsState (MyTokens(lts) Union ts, YrTokens(lts) Minus ts, NotAvTokens(lts) Minus ts);
MakeYrSide (ts, lts) =
      LTsState (MyTokens(lts) Minus ts, YrTokens(lts) Union ts, NotAvTokens(lts) Minus ts);
ofsort Bool
AllI (ts, lts) = (MyTokens(lts) Union NotAvTokens(lts)) Includes ts;
AllA (ts, lts) = (YrTokens(lts) Union NotAvTokens(lts)) Includes ts;
AllII (ts, lts) = MyTokens(lts) Includes ts;          AllAA (ts, lts) = YrTokens(lts) Includes ts;
AnyI (ts, lts) = ((MyTokens(lts) Union NotAvTokens(lts)) Ints ts) ne {};
AnyA (ts, lts) = ((YrTokens(lts) Union NotAvTokens(lts)) Ints ts) ne {};
AnyII (ts, lts) = (MyTokens(lts) Ints ts) ne {};      AnyAA (ts, lts) = (YrTokens(lts) Ints ts) ne {};
IsMySide (t, lts) = t IsIn MyTokens (lts);            IsYrSide (t, lts) = t IsIn YrTokens (lts);
IsNotAv (t, lts) = t IsIn NotAvTokens (lts);
endtype

process SCEPBehaviour [s] (role:SSUserRole): exit :=
      SCEPConnectPropose [s] (role) >> accept p:SSP in
      (     (     SCEPConnectAccept [s] (role,p) >>
                  accept fus:SFUs, lts:SLTsState, sn:SSPSN in SCEPDataTransfer [s] (fus,lts,sn,role)
            []    SCEPConnectReject [s]
            ) [> SCEPAbort [s]
      )
[]    [role eq called] -> exit
endproc
(*----------------------------------------------------------------------------------------------------------------------
```

9.2.3 Session connection establishment at a SCEP

The constraints imposed at a calling (called) SCEP may be summarized as follows:

- the first event may only be a S-CONNECT request (indication);
- the calling (called) SSAP address parameter of the S-CONNECT request (indication) must be the address of the SSAP at which the request (indication) is executed;
- the following event may be a S-CONNECT confirm (response);
- the SS-provider may constrain and affect the parameters in these events in a number of ways (see functions *IsValidSCONreq*, *IsValidSCONrsp*, *IsIndicationOf*, and *IsConResponseOf* in 3.1.3.5).

```
----------------------------------------------------------------------------------------------------------------------*)
process SCEPConnectPropose [s] (role:SSUserRole): exit (SSP) :=
[role eq calling] -> s ?sa:SAddress ?si:SCEI ?p:SSP [IsValidSCONreq(p) and (sa IsCallingOf p)]; exit (p)
[]
[role eq called] ->  s ?sa:SAddress ?si:SCEI ?p:SSP [IsSCONind(p) and (sa IsCalledOf p)]; exit (p)
endproc
```

```
process SCEPConnectAccept [s] (role:SSUserRole, p:SSP): exit (SFUs, SLTsState, SSPSN) :=
[role eq calling] ->
s ?sa:SAddress ?si:SCEI ?p1:SSP [IsSCONcnf(p1) and IsAccept(ConResult(p1))];
(     let rts:STokens = RqrTokens(TsAss(p)), rts1:STokens = RqrTokens(TsAss(p1)),
            ats:STokens = AcrTokens(TsAss(p)), ats1:STokens = AcrTokens(TsAss(p1)) in
      exit (Rqms(p) Ints Rqms(p1), RqrLTsState(rts Union rts1, ats Union ats1), SPSN(p1))
)
[]
[role eq called] ->
s ?sa:SAddress ?si:SCEI ?p1:SSP
      [IsValidSCONrsp(p1) and IsAccept(ConResult(p1)) and (p1 IsConResponseOf p)];
(     let rts:STokens = RqrTokens(TsAss(p)), rts1:STokens = RqrTokens(TsAss(p1)),
            ats:STokens = AcrTokens(TsAss(p)), ats1:STokens = AcrTokens(TsAss(p1)) in
      exit (Rqms(p) Ints Rqms(p1), AcrLTsState(rts Union rts1, ats Union ats1), SPSN(p1))
)
endproc

process SCEPConnectReject [s]: exit :=
s ?sa:SAddress ?si:SCEI ?p:SSP [IsSCONcnf(p) and IsReject(ConResult(p))]; exit
endproc

process SCEPAbort [s]: exit :=
s ?sa:SAddress ?si:SCEI ?p:SSP [IsValidSUAB(p) or IsSPABind(p)]; exit
endproc
(*-------------------------------------------------------------------------------------------------------------------------------
```

9.2.4 Data transfer phase at a SCEP

The constraints specified by *SCEPDataTransfer* are further decomposed as follows:

- restrictions on the available set of services and associated primitives as a function of the selected functional units are specified by process *SCEPServices* (see 9.2.4.1);
- the constraints that depend on the session variables defined in clause A.5 of ISO 8326 (tokens, Vact, Vrsp, Vrspnb, V(A), V(M), V(R), and Vsc) are specified by process *SCEPVariables* (see 9.2.4.2);
- the constraints on valid orderings of primitives as a function of the state of the SCEP are specified by process *SCEPSPOrdering* (see 9.2.4.3).

An instance of process *SCEPSPOrdering* successfully terminates after the release of the SC. It then enables the disabling of the instances of *SCEPServices* and *SCEPVariables*.

```
-------------------------------------------------------------------------------------------------------------------------------*)
process SCEPDataTransfer [s] (fus:SFUs, lts:SLTsState, sn:SSPSN, role:SSUserRole): exit :=
      ( ( SCEPServices [s] (fus) || SCEPVariables [s] (fus,lts,sn,role) ) [> exit )
||    SCEPSPOrdering [s] (role, dk IsIn MyTokens(lts))
endproc
(*-------------------------------------------------------------------------------------------------------------------------------
```

9.2.4.1 Constraints on the available set of services

For each functional unit, except Duplex, the available services (and, hence, the SSPs which can be executed, and the static restrictions on the parameter values) are specified as separate, guarded alternatives. Only the functionality that resulted from the negotiation of functional units is allowed by the guards. Note that the definition of an alternative associated with the Duplex functional unit is unnecessary since Duplex allows no services in addition to those of the (non-negotiable) Kernel.

```
----------------------------------------------------------------------------------------------------------------------*)
process SCEPServices [s] (fus:SFUs): noexit :=
(     (* Kernel *)
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsValidSDT(p) or IsValidSREL(p) or (IsValidSRELAK(p) and (affirmative IsRelResultOf p))]; exit
[]    [HD IsIn fus] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsValidSDT(p) or IsValidSGT(p) or IsValidSPT(p) or IsValidSREL(p) or
            (IsValidSRELAK(p) and (affirmative IsRelResultOf p))]; exit
[]    [EXCEP IsIn fus] -> s ?sa:SAddress ?si:SCEI ?p:SSP [IsValidSUER(p) or IsSPER(p)]; exit
[]    [TD IsIn fus] -> s ?sa:SAddress ?si:SCEI ?p:SSP [IsValidSTD(p)]; exit
[]    [NR IsIn fus] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsValidSGT(p) or IsValidSPT(p) or IsValidSREL(p) or IsValidSRELAK(p)]; exit
[]    [SY IsIn fus] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsValidSGT(p) or IsValidSPT(p) or IsValidSSYNm(p) or IsValidSSYNmAK(p)]; exit
[]    [MA IsIn fus] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsValidSGT(p) or IsValidSPT(p) or IsValidSSYNMa(p) or IsValidSSYNMaAK(p)]; exit
[]    [RESYN IsIn fus] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP [IsValidSRSYN(p) or IsValidSRSYNAK(p)]; exit
[]    [EX IsIn fus] -> s ?sa:SAddress ?si:SCEI ?p:SSP [IsValidSEX(p)]; exit
[]    [ACT IsIn fus] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsValidSGT(p) or IsValidSPT(p) or IsSCG(p) or IsValidSACTS(p) or
            IsValidSACTR(p) or IsSACTI(p) or IsSACTIAK(p) or IsSACTD(p) or IsSACTDAK(p) or
            IsValidSACTE(p) or IsValidSACTEAK(p)]; exit
[]    [CD IsIn fus] -> s ?sa:SAddress ?si:SCEI ?p:SSP [IsValidSCD(p) or IsValidSCDAK(p)]; exit
) >> SCEPServices [s] (fus)
endproc
(*----------------------------------------------------------------------------------------------------------------------
```

9.2.4.2 Session variable constraints

Process *SCEPVariables* specifies how the session variables constrain, and are affected by, the execution of SSPs. This process, therefore, is a formalization of the predicates and actions which are defined for the state tables in annex A of ISO 8326.

All the variables of subclause A.5.4 of ISO 8326 have a corresponding variable in *SCEPVariables*, except "Vcoll" and "Vdnr" which are indirectly represented in *SCEPSPOrdering*. One new data type, *Vrsp*, is introduced for the formal representation of "Vrsp". The constants, defined by a renaming of the *Quartet* type (see 9.1.5.1), correspond with the values defined in subclause A.5.4.2 of ISO 8326. Further, a mapping function *v* is defined to relate Resynchronize Type parameter values (see 9.1.4.10) with these constants.

In the specification of *SCEPVariables* six separate constraints are recognized, each of which is represented by a process which may be composed in parallel with the other processes. Each of the processes deals with a particular (combination of) variable(s):

- *TokensConstraints* specifies the actions and predicates related to the token settings and assignment of tokens;
- *VrspConstraints* specifies the actions and predicates related to resynchronization collision cases;
- *VAConstraints* specifies the actions and predicates related to the acknowledgement of synchronization points;
- *VMConstraints* specifies the actions and predicates related to the usage of a next synchronization point number;
- *VRConstraints* specifies the actions and predicates related to the resynchronization to a previous synchronization point;

- *VactConstraints* specifies the actions and predicates related to activity management.

The initial values of the variables are set according to subclause 11.4.2 of ISO 8326. The constraints represented by the latter five processes are only applicable when the usage of synchronization numbers is selected during SC establishment or when the Activity Management functional unit is selected. If this is not the case, the absence of constraints in this respect is represented by process *IgnoreSSP*.

```
----------------------------------------------------------------------------------------------------------------------------*)
type Vrsp is Vrsp0, SResynchronizeType
opns v: SResynType -> Vrsp
eqns ofsort Vrsp
v(a) = ra;                v(r) = rr;                v(s) = rs;
endtype

type Vrsp0 is Quartet renamedby
sortnames Vrsp for Quartet
opnnames
no for constant1          ra for constant2          rr for constant3          rs for constant4
endtype

process SCEPVariables [s] (fus:SFUs, lts:SLTsState, sn:SSPSN, role:SSUserRole): noexit :=
      TokensConstraints [s] (lts)
||    (     (     [sn ne absent] -> ( let VR:SSPSN = s(0) in exit (VR) )
            []    [ACT Isln fus] -> ( let VR:SSPSN = absent in exit (VR) )
            ) >> accept VR:SSPSN in
            (     let VA,VM:SSPSN = sn, Vrspnb:SSPSN = absent, Vrsp:Vrsp = no, Vsc,Vact:Bool = false in
                  (     VrspConstraints [s] (Vrsp,Vrspnb,role)
                  ||    VAConstraints [s] (VA,VM,Vsc)
                  ||    VMConstraints [s] (VM)
                  ||    VRConstraints [s] (VR,VM,Vrsp)
                  ||    VactConstraints [s] (Vact)
            )     )
      [] [(sn eq absent) and (ACT NotIn fus)] -> IgnoreSSP [s]
      )
where
process IgnoreSSP [s]: noexit := s ?sa:SAddress ?si:SCEI ?p:SSP; IgnoreSSP [s] endproc
endproc (* SCEPVariables *)
(*----------------------------------------------------------------------------------------------------------------------------
```

The parameter of *TokenConstraints* indicates the state of the tokens as it can be locally observed. The initial state is negotiated during SC establishment. Note that a new assignment of the available tokens can be negotiated by means of the Resynchronize service.

The specification below follows the body of text of ISO 8326 whenever an inconsistency exists between the text and annex A.

NOTES

1 - In annex A of ISO 8326, the occurrence of a S-TOKEN-PLEASE request is constrained by predicate p53: ALL(AV,RT); according to the body of text (table 8) this should be: ALL(AA,RT).

2 - In annex A of ISO 8326, the occurrence of a S-ACTIVITY-START and -RESUME request is constrained by predicate p45: (FU(ACT) & ^Vact) & I(dk) & I(mi) & I(ma); according to the body of text (table 8) this should be: (FU(ACT) & ^Vact) & I(dk) & I(mi) & II(ma).

3 - In annex A of ISO 8326, the occurrence of a S-CONTROL-GIVE request is constrained by predicate p55: (FU(ACT) & ^Vact) & ANY(II,tk-dom); according to the body of text (table 8) this should be: (FU(ACT) & ^Vact) & ALL(I,tk-dom).

```
----------------------------------------------------------------------------------------------------------------------------*)
process TokensConstraints [s] (lts:SLTsState): noexit :=
      [IsNotAv(dk,lts) or IsMySide(dk,lts)] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSDTreq(p)]; TokensConstraints [s] (lts)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSGTreq(p) and AllII(Tokens(p),lts)];
      TokensConstraints [s] (MakeYrSide(Tokens(p),lts))
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSGTind(p)]; TokensConstraints [s] (MakeMySide(Tokens(p),lts))
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSPTreq(p) and AllAA(Tokens(p),lts)];
      TokensConstraints [s] (lts) (* see NOTE 1 *)
[]    [AllI(tkdom,lts)] -> s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRELreq(p)]; TokensConstraints [s] (lts)
[]    [(IsNotAv(dk,lts) or IsMySide(dk,lts)) and IsMySide(mi,lts)] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNmreq(p)]; TokensConstraints [s] (lts)
[]    [AllI(Insert(dk,Insert(mi,{})),lts) and IsMySide(ma,lts)] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsSCDreq(p) or IsSSYNMareq(p) or IsSACTSreq(p) or IsSACTRreq(p) or IsSACTEreq(p)];
      TokensConstraints [s] (lts) (* see NOTE 2 *)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsSRSYNreq(p) and ( Tokens(TsAss(p)) eq (MyTokens(lts) Union YrTokens(lts)) )];
      TokensConstraints [s] (lts)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNrsp(p)];
      ( let lts1:SLTsState = RqrTokens(TsAss(p)), lts2:SLTsState = AcrTokens(TsAss(p)) in
        TokensConstraints [s] (AcrTokStates(lts1,lts2))  )
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNcnf(p)];
      ( let lts1:SLTsState = RqrTokens(TsAss(p)), lts2:SLTsState = AcrTokens(TsAss(p)) in
        TokensConstraints [s] (RqrTokStates(lts1,lts2))  )
[]    [AllI(tkdom,lts) and IsMySide(ma,lts)] -> s ?sa:SAddress ?si:SCEI ?p:SSP [IsSCGreq(p)];
      TokensConstraints [s] (MakeYrSide(MyTokens(lts),lts)) (* see NOTE 3 *)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSCGind(p) or IsSACTIcnf(p) or IsSACTDcnf(p)];
      TokensConstraints [s] (MakeMySide(YrTokens(lts),lts))
[]    [IsMySide(ma,lts)] -> s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIreq(p) or IsSACTDreq(p)];
      TokensConstraints [s] (lts)
[]    [IsYrSide(dk,lts] -> s ?sa:SAddress ?si:SCEI ?p:SSP [IsSUERreq(p)]; TokensConstraints [s] (lts)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIrsp(p) or IsSACTDrsp(p)];
      TokensConstraints [s] (MakeYrSide(MyTokens(lts),lts))
[]    s ?sa:SAddress ?si:SCEI ?p:SSP
            [not (IsSDTreq(p) or IsSCDreq(p) or IsSGT(p) or IsSPTreq(p) or IsSCG(p) or
            SSYNmreq(p) or IsSSYNMareq(p) or IsSRSYNreq(p) or IsSRSYNAK(p) or IsSUER(p) or
            IsSACTSreq(p) or IsSACTRreq(p) or IsSACTIreq(p) or IsSACTIAK(p) or
            IsSACTDreq(p) or IsSACTDAK(p) or IsSACTEreq(p) or IsSRELreq(p) )];
      TokensConstraints [s] (lts)
endproc
(*----------------------------------------------------------------------------------------------------------------------------
```

NOTE - Process *VrspConstraints* specifies that after the occurrence of a S-ACTIVITY-INTERRUPT or -DISCARD request, or indication, *Vrsp* is assigned value *no*. ISO 8326, however, does not define a corresponding assignment. The assignment is felt necessary to indicate that if resynchronization was in progress, it will be aborted as a result of the occurrence of one of the above SSPs.

```
----------------------------------------------------------------------------------------------------------------------------*)
process VrspConstraints [s] (Vrsp:Vrsp, Vrspnb:SSPSN, role:SSUserRole): noexit :=
      [(Vrsp eq ra) implies (role eq calling)] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNreq(p) and (ResynType(p) eq a)];
      VrspConstraints [s] (ra, SPSN(p), role)
[]    [(Vrsp ne ra) and ((Vrsp eq rs) implies (role eq calling))] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNreq(p) and (ResynType(p) eq s)];
      VrspConstraints [s] (rs, SPSN(p), role)
[]    [Vrsp eq no] -> s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNreq(p) and (ResynType(p) eq r)];
      VrspConstraints [s] (rr, SPSN(p), role)
```

```
[]    [Vrsp eq rr] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsSRSYNreq(p) and (ResynType(p) eq r) and
            ( (SPSN(p) lt Vrspnb) or ( (SPSN(p) eq Vrspnb) and (role eq calling) ) )];
      VrspConstraints [s] (rr, SPSN(p), role)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNind(p)];
      VrspConstraints [s] (v(ResynType(p)), SPSN(p), role)
[]    [(Vrsp eq ra) or (Vrsp eq rr)] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNrsp(p) and (SPSN(p) eq Vrspnb)];
      VrspConstraints [s] (no, Vrspnb, role)
[]    [Vrsp eq rs] ->  s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNrsp(p)];
      VrspConstraints [s] (no, Vrspnb, role)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNcnf(p) or IsSACTI(p) or IsSACTD(p)];
      VrspConstraints [s] (no, Vrspnb, role) (* see NOTE *)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP
            [not (IsSRSYN(p) or IsSRSYNAK(p) or IsSACTI(p) or IsSACTD(p))];
      VrspConstraints [s] (Vrsp, Vrspnb, role)
endproc
(*-----------------------------------------------------------------------------------------------------------------------
```

In the following processes, variable "V(A)" is represented by *VA*, variable "V(M)" is represented by *VM*, and variable "V(R)" is represented by *VR*. The specification below follows the body of text of ISO 8326 whenever an inconsistency exists between the text and annex A.

NOTE - Annex A of ISO 8326 defines that, after occurrence of a S-SYNC-MINOR response and confirm, action [25] is performed: Set V(M) = serial number + 1; according to the body of text (subclause 11.4.3) this should be: Set V(A) = serial number + 1.

```
-----------------------------------------------------------------------------------------------------------------------*)
process VAConstraints [s] (VA,VM:SSPSN, Vsc:Bool): noexit :=
      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNmreq(p) or IsSSYNMareq(p) or IsSACTEreq(p)];
      (     [Vsc eq true] -> VAConstraints [s] (VM, Next(VM), false)
      []    [Vsc eq false] -> VAConstraints [s] (VA, Next(VM), Vsc)    )
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNmind(p)];
      (     [Vsc eq true] -> VAConstraints [s] (VA, Next(VM), Vsc)
      []    [Vsc eq false] -> VAConstraints [s] (VM, Next(VM), true)    )
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNMaind(p) or IsSACTEind(p)];
      (     [Vsc eq true] -> VAConstraints [s] (VA, Next(VM), Vsc)
      []    [Vsc eq false] -> VAConstraints [s] (VM, Next(VM), Vsc)    )
[]    [Vsc eq true] ->
      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNmrsp(p) and (SPSN(p) ge VA)];
      VAConstraints [s] (Next(SPSN(p)), VM, Vsc) (* see NOTE *)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNmcnf(p)];
      VAConstraints [s] (Next(SPSN(p)), VM, Vsc) (* see NOTE *)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNMaAK(p) or IsSACTEAK(p)];
      VAConstraints [s] (VM, VM, Vsc)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNind(p) and (ResynType(p) eq a)];
      VAConstraints [s] (VA, SPSN(p), Vsc)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNAK(p)]; VAConstraints [s] (SPSN(p), SPSN(p), Vsc)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTS(p)]; VAConstraints [s] (s(1), s(1), Vsc)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTR(p)];
      VAConstraints [s] (Next(SPSN(p)), Next(SPSN(p)), Vsc)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP
            [not (IsSSYNm(p) or IsSSYNmAK(p) or IsSSYNMa(p) or IsSSYNMaAK(p) or
            ( IsSRSYNind(p) and (ResynType(p) eq a) ) or IsSRSYNAK(p) or IsSACTS(p) or IsSACTR(p) or
            IsSACTE(p) or IsSACTEAK(p))];
      VAConstraints [s] (VA, VM, Vsc)
endproc
```

```
process VMConstraints [s] (VM:SSPSN): noexit :=
        s ?sa:SAddress ?si:SCEI ?p:SSP
            [(IsSSYNm(p) or IsSSYNMa(p) or IsSACTE(p)) and (SPSN(p) eq VM)];
        VMConstraints [s] (Next(VM))
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNmrsp(p) and (SPSN(p) lt VM)]; VMConstraints [s] (VM)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNreq(p) and (ResynType(p) eq r) and (SPSN(p) le VM)];
        VMConstraints [s] (VM)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [(IsSRSYNind(p) and (ResynType(p) eq a)) or IsSRSYNAK(p)];
        VMConstraints [s] (SPSN(p))
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTS(p)]; VMConstraints [s] (s(1))
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTR(p)]; VMConstraints [s] (Next(SPSN(p)))
[]      s ?sa:SAddress ?si:SCEI ?p:SSP
            [not (IsSSYNm(p) or IsSSYNmrsp(p) or IsSSYNMa(p) or
            ( IsSRSYNreq(p) and (ResynType(p) eq r) ) or ( IsSRSYNind(p) and (ResynType(p) eq a) ) or
            IsSRSYNAK(p) or IsSACTS(p) or IsSACTR(p) or IsSACTE(p))];
        VMConstraints [s] (VM)
endproc

process VRConstraints [s] (VR, VM:SSPSN, Vrsp:Vrsp): noexit :=
        s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNm(p) or IsSSYNMa(p) or IsSACTE(p)];
        VRConstraints [s] (VR, Next(VM), Vrsp)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNMaAK(p) or IsSACTEAK(p)];
        VRConstraints [s] (VM, VM, Vrsp)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsSRSYNreq(p) and ( (ResynType(p) eq r) implies (SPSN(p) ge VR) )];
        VRConstraints [s] (VR, VM, v(ResynType(p)))
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNind(p)];
        (       [ResynType(p) eq a] -> VRConstraints [s] (VR, SPSN(p), ra)
        []      [ResynType(p) ne a] -> VRConstraints [s] (VR, VM, v(ResynType(p)))    )
[]      [(Vrsp eq ra) or (Vrsp eq rs)] ->
        s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNAK(p)]; VRConstraints [s] (s(0), SPSN(p), no)
[]      [Vrsp eq rr] ->
        s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNAK(p)]; VRConstraints [s] (VR, SPSN(p), no)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTS(p)]; VRConstraints [s] (s(1), s(1), Vrsp)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTR(p)]; VRConstraints [s] (s(1), Next(SPSN(p)), Vrsp)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTI(p) or IsSACTD(p)]; VRConstraints [s] (VR, VM, no)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP
            [not (IsSSYNm(p) or IsSSYNMa(p) or IsSRSYN(p) or IsSRSYNAK(p) or
            IsSACTS(p) or IsSACTR(p) or IsSACTI(p) or IsSACTD(p) or
            IsSACTE(p) or IsSACTEAK(p))];
        VRConstraints [s] (VR, VM, Vrsp)
endproc

process VactConstraints [s] (Vact:Bool): noexit :=
        [Vact eq true] ->
        (       s ?sa:SAddress ?si:SCEI ?p:SSP
                    [IsSSYNm(p) or IsSSYNmAK(p) or IsSSYNMa(p) or IsSRSYN(p) or
                    IsSACTI(p) or IsSACTD(p) or IsSACTE(p)]; VactConstraints [s] (Vact)
        []      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIAK(p) or IsSACTDAK(p) or IsSACTEAK(p)];
                VactConstraints [s] (false)
        )
[]      [Vact eq false] ->
        (       s ?sa:SAddress ?si:SCEI ?p:SSP [IsSCD(p) or IsSCG(p)]; VactConstraints [s] (Vact)
        []      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTS(p) or IsSACTR(p)]; VactConstraints [s] (true)    )
[]      s ?sa:SAddress ?si:SCEI ?p:SSP
            [not (IsSCD(p) or IsSCG(p) or IsSSYNm(p) or IsSSYNmAK(p) or IsSSYNMa(p) or
            IsSRSYN(p) or IsSUER(p) or IsSACTR(p) or IsSACTI(p) or IsSACTIAK(p) or
            IsSACTD(p) or IsSACTDAK(p) or IsSACTE(p) or IsSACTEAK(p))];
        VactConstraints [s] (Vact)
endproc
(*-----------------------------------------------------------------------------------------------------------------------
```

9.2.4.3 Ordering of session service primitives

The possible ordering of SSPs at a SCEP is specified by process *SCEPSPOrdering*. It formalizes the state tables of annex A of ISO 8326 without predicates and actions. The initial state after the successful establishment of a SC is the data transfer state (STA 713). This state and the other states are represented by processes with a corresponding name. Any such process specifies the possible SSP events of the state, and transitions to other states.

All the processes are parameterized with the role of the local SS-user and a boolean value. The SS-user role is used only to determine the ordering of SSPs in case of a Release collision. The boolean value has only significance when the data token is available, in which case it indicates whether the token is locally owned or not; it is used to determine the ordering of SSPs after a S-U- or S-P-EXCEPTION-REPORT indication. Process *AwaitSSYNMrsp* has an additional parameter indicating a synchronization number (derived from the previous S-SYNCHRONIZE-MAJOR indication), and process *AwaitSRSYNrsp* a parameter indicating a tokens assignment (derived from the previous S-RESYNCHRONIZE indication). In both cases the additional parameter is used to specify a constraint between the indication and the corresponding, awaited response SSP.


```
----------------------------------------------------------------------------------------------------------------------*)
process SCEPSPOrdering [s] (role:SSUserRole, dkOwned:Bool): exit := DataTransfer [s] (role, dkOwned)
endproc

process DataTransfer [s] (role:SSUserRole, dkOwned:Bool): exit :=
        s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsSDT(p) or IsSEX(p) or IsSTD(p) or (IsSGT(p) and (dk NotIn Tokens(p))) or IsSPT(p) or
            IsSSYNm(p) or IsSSYNmAK(p) or ((IsSUERind(p) or IsSPERind(p)) and not(dkOwned)) or
            IsSACTS(p) or IsSACTR(p)]; DataTransfer [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [(IsSGTreq(p) and (dk IsIn Tokens(p))) or IsSCGreq(p)];
        DataTransfer [s] (role,false)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [(IsSGTind(p) and (dk IsIn Tokens(p))) or IsSCGind(p)];
        DataTransfer [s] (role,true)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSCDreq(p)]; AwaitSCDcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSCDind(p)]; AwaitSCDrsp [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNMareq(p)]; AwaitSSYNMcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNMaind(p)]; AwaitSSYNMrsp [s] (role,SPSN(p),dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNreq(p)]; AwaitSRSYNcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNind(p)]; AwaitSRSYNrsp [s] (role,TsAss(p),dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSUERreq(p)]; AwaitRecInd [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [(IsSUERind(p) or IsSPERind(p)) and dkOwned];
        AwaitRecReq [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIreq(p)]; AwaitSACTIcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIind(p)]; AwaitSACTIrsp [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDreq(p)]; AwaitSACTDcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDind(p)]; AwaitSACTDrsp [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTEreq(p)]; AwaitSACTEcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTEind(p)]; AwaitSACTErsp [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRELreq(p)]; AwaitSRELcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRELind(p)]; AwaitSRELrsp [s] (role,dkOwned)
endproc

process AwaitSCDcnf [s] (role:SSUserRole, dkOwned:Bool): exit :=
        s ?sa:SAddress ?si:SCEI ?p:SSP [(IsSGTind(p) and (dk NotIn Tokens(p))) or IsSPTind(p)];
        AwaitSCDcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSGTind(p) and (dk IsIn Tokens(p))]; AwaitSCDcnf [s] (role,true)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSCDcnf(p)]; DataTransfer [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSPERind(p)]; AwaitRecReq [s] (role,dkOwned)
endproc
```

```
process AwaitSCDrsp [s] (role:SSUserRole, dkOwned:Bool): exit :=
      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSPTreq(p)]; AwaitSCDrsp [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSCDrsp(p)]; DataTransfer [s] (role,dkOwned)
endproc

process AwaitSSYNMcnf [s] (role:SSUserRole, dkOwned:Bool): exit :=
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsSDTind(p) or IsSEXind(p) or IsSTDind(p) or (IsSGT(p) and (dk NotIn Tokens(p))) or
            IsSPTind(p) or IsSSYNmcnf(p)]; AwaitSSYNMcnf [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSGT(p) and (dk IsIn Tokens(p))]; DataTransfer [s] (role,IsInd(p))
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNMacnf(p)]; DataTransfer [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNreq(p) and (ResynType(p) ne r)];
      AwaitSRSYNcnf [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRYNind(p)]; AwaitSRSYNrsp [s] (role,TsAss(p),dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [(IsSUERind(p) or IsSPERind(p)];
      (     [dkOwned] -> AwaitRecReq [s] (role,dkOwned)
      []    [not(dkOwned)] -> DataTransfer [s] (role,dkOwned)  )
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIreq(p)]; AwaitSACTIcnf [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDreq(p)]; AwaitSACTDcnf [s] (role,dkOwned)
endproc

process AwaitSSYNMrsp [s] (role:SSUserRole, sn:SSPSN, dkOwned:Bool): exit :=
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsSDTreq(p) or IsSEXreq(p) or IsSTDreq(p) or (IsSGT(p) and (dk NotIn Tokens(p))) or
            IsSPTreq(p) or (IsSSYNmrsp(p) and (SPSN(p) ne sn))]; AwaitSSYNMrsp [s] (role,sn,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSGT(p) and (dk IsIn Tokens(p))];
      AwaitSSYNMrsp [s] (role,sn,IsInd(p))
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSSYNMarsp(p)]; DataTransfer [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNreq(p)]; AwaitSRSYNcnf [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsSRSYNind(p) and ((ResynType(p) eq a) or (ResynType(p) eq s))];
      AwaitSRSYNrsp [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSUERreq(p)]; AwaitRecInd [p] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIind(p)]; AwaitSACTIrsp [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDind(p)]; AwaitSACTDrsp [s] (role,dkOwned)
endproc

process AwaitSRSYNcnf [s] (role:SSUserRole, dkOwned:Bool): exit :=
      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNcnf(p)];
      DataTransfer [s] (role, dk IsIn RqrTokens(TsAss(p)))
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNind(p)]; AwaitSRSYNrsp [s] (role,TsAss(p),dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIind(p)]; AwaitSACTIrsp [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDind(p)]; AwaitSACTDrsp [s] (role,dkOwned)
endproc

process AwaitSRSYNrsp [s] (role:SSUserRole, a:STsAss, dkOwned:Bool): exit :=
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsSRSYNrsp(p) and ( RqrTokens(TsAss(p)) Includes RqrTokens(a) ) and
            ( AcrTokens(TsAss(p)) Includes AcrTokens(a) ) and ( Tokens(TsAss(p)) eq Tokens(a) )];
      DataTransfer [s] (role, dk IsIn AcrTokens(TsAss(p)))
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNreq(p)]; AwaitSRSYNcnf [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIreq(p)]; AwaitSACTIcnf [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDreq(p)]; AwaitSACTDcnf [s] (role,dkOwned)
endproc
```

```
process AwaitRecInd [s] (role:SSUserRole, dkOwned:Bool): exit :=
        s ?sa:SAddress ?si:SCEI ?p:SSP
              [(IsSGTind(p) and (dk NotIn Tokens(p))) or IsSUERind(p) or IsSPERind(p)];
        AwaitRecInd [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSGTind(p) and (dk IsIn Tokens(p))]; DataTransfer [s] (role,true)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNind(p)]; AwaitSRSYNrsp [s] (role,TsAss(p),dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIind(p)]; AwaitSACTIrsp [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDind(p)]; AwaitSACTDrsp [s] (role,dkOwned)
endproc

process AwaitRecReq [s] (role:SSUserRole, dkOwned:Bool): exit :=
        s ?sa:SAddress ?si:SCEI ?p:SSP [(IsSGT(p) and (dk NotIn Tokens(p)))];
        AwaitRecReq [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSGT(p) and (dk IsIn Tokens(p))]; DataTransfer [s] (role,IsInd(p))
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNreq(p)]; AwaitSRSYNcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNind(p)]; AwaitSRSYNrsp [s] (role,TsAss(p),dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIreq(p)]; AwaitSACTIcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIind(p)]; AwaitSACTIrsp [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDreq(p)]; AwaitSACTDcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDind(p)]; AwaitSACTDrsp [s] (role,dkOwned)
endproc

process AwaitSACTIcnf [s] (role:SSUserRole, dkOwned:Bool): exit :=
s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIcnf(p)]; DataTransfer [s] (role,dkOwned)
endproc

process AwaitSACTIrsp [s] (role:SSUserRole, dkOwned:Bool): exit :=
s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIrsp(p)]; DataTransfer [s] (role,dkOwned)
endproc

process AwaitSACTDcnf [s] (role:SSUserRole, dkOwned:Bool): exit :=
s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDcnf(p)]; DataTransfer [s] (role,dkOwned)
endproc

process AwaitSACTDrsp [s] (role:SSUserRole, dkOwned:Bool): exit :=
s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDrsp(p)]; DataTransfer [s] (role,dkOwned)
endproc

process AwaitSACTEcnf [s] (role:SSUserRole, dkOwned:Bool): exit :=
        s ?sa:SAddress ?si:SCEI ?p:SSP
              [IsSDTind(p) or IsSEXind(p) or IsSTDind(p) or (IsSGT(p) and (dk NotIn Tokens(p))) or
              IsSPTind(p) or IsSSYNmcnf(p)]; AwaitSACTEcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSGT(p) and (dk IsIn Tokens(p))];
        AwaitSACTEcnf [s] (role,IsInd(p))
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTEcnf(p)]; DataTransfer [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNind(p)]; AwaitSRSYNrsp [s] (role,TsAss(p),dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSUERind(p) or IsSPERind(p)];
        (       [dkOwned] -> AwaitRecReq [s] (role,dkOwned)
        []      [not(dkOwned)] -> DataTransfer [s] (role,dkOwned)  )
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIreq(p)]; AwaitSACTIcnf [s] (role,dkOwned)
[]      s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDreq(p)]; AwaitSACTDcnf [s] (role,dkOwned)
endproc
```

```
process AwaitSACTErsp [s] (role:SSUserRole, dkOwned:Bool): exit :=
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsSDTreq(p) or IsSEXreq(p) or IsSTDreq(p) or (IsSGT(p) and (dk NotIn Tokens(p))) or
            IsSPTreq(p) or IsSSYNmrsp(p)]; AwaitSACTErsp [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSGT(p) and (dk IsIn Tokens(p))];
      AwaitSACTErsp [s] (role,IsInd(p))
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTErsp(p)]; DataTransfer [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNreq(p)]; AwaitSRSYNcnf [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSUERreq(p)]; AwaitRecInd [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTIind(p)]; AwaitSACTIrsp [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSACTDind(p)]; AwaitSACTDrsp [s] (role)
endproc

process AwaitSRELcnf [s] (role:SSUserRole, dkOwned:Bool): exit :=
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsSDTind(p) or IsSEXind(p) or IsSTDind(p) or IsSPTind(p) or IsSSYNmcnf(p)];
      AwaitSRELcnf [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRELind(p)];
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [(IsSRELrsp(p) and (role eq calling)) or (IsSRELcnf(p) and (role eq called))];
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [(IsSRELcnf(p) and (role eq calling)) or (IsSRELrsp(p) and (role eq called))]; exit
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRELcnf(p)];
      (     [negative IsRelResultOf p] -> DataTransfer [s] (role,dkOwned)
      []    [affirmative IsRelResultOf p] -> exit   )
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNind(p)]; AwaitSRSYNrsp [s] (role,TsAss(p),dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSUERind(p) or IsSPERind(p)]; AwaitRecReq [s] (role,dkOwned)
endproc

process AwaitSRELrsp [s] (role:SSUserRole, dkOwned:Bool): exit :=
      s ?sa:SAddress ?si:SCEI ?p:SSP
            [IsSDTreq(p) or IsSEXreq(p) or IsSTDreq(p) or IsSPTreq(p) or IsSSYNmrsp(p)];
      AwaitSRELrsp [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRELrsp(p)];
      (     [negative IsRelResultOf p] -> DataTransfer [s] (role,dkOwned)
      []    [affirmative IsRelResultOf p] -> exit   )
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSRSYNreq(p)]; AwaitSRSYNcnf [s] (role,dkOwned)
[]    s ?sa:SAddress ?si:SCEI ?p:SSP [IsSUERreq(p)]; AwaitRecInd [s] (role,dkOwned)
endproc
(*-------------------------------------------------------------------------------------------------------
```

10 End-to-end constraints for a single session connection

End-to-end constraints are not completely defined in ISO 8326. In particular, the state table description in annex A of ISO 8326 is confined to the definition of the Session Service local constraints. The formally defined end-to-end constraints are partially derived from ISO 8327; they abstract, however, from the Session Protocol specification in the sense that internal behaviour of the SS-provider (that is, manipulation of SPDUs) is not defined, but only externally visible behaviour (in terms of SSP events). Data types for end-to-end constraints have therefore no correspondents in ISO 8326 nor in ISO 8327.

The specification of end-to-end constraints is based on maintaining a history of SSP events at each SCEP, restricted to those events which are relevant for deriving the behaviour at the other SCEP. The SSP events which are relevant for the history of a SCEP are requests, responses, and SS-provider (locally) generated indications at that SCEP. They may lead to corresponding remote indications or confirms at the other SCEP. For the sake of brevity, we will hereafter refer to the first events as "Requests" and to the latter as "Indications".

Requests that are recorded in a history and Indications that are derived from that history relate to a single direction of transfer. We will adopt the convention of referring to the SCEP where the Requests are executed at the "requestor" endpoint and the SCEP where the Indications are executed at the "acceptor" endpoint, for the direction of concern. Furthermore, Requests and Indications for the other direction of transfer are said to be "observed" at the acceptor endpoint and requestor endpoint, respectively. Certain observed SSP events may change the end-to-end characteristics of the association for the direction of concern.

10.1 Association data types

The association data types are formed by type definitions for the representation of state information related to the association that exists between two SCEPs. They consist of definitions for the construction of a SS-provider state (see 10.1.1), flow control option (see 10.1.2), and history of Requests (see 10.1.3).

10.1.1 Session service provider state

End-to-end constraints for a particular direction of transfer depend upon the state of the SS-provider associated with that direction of transfer. Three states are distinguished with different end-to-end characteristics (see 10.1.3.3); we will refer to these states as the "normal", "resynchronization" and "exception" state. Type *SProviderState0* defines the three SS-provider state values. Type *SProviderState* is an enrichment that defines functions which yield the SS-provider state resulting from a given SSP event.

We may summarize the state transitions as follows:

- in the normal state, observation of a S-RESYNCHRONIZE request at the acceptor side will effect a transition to resynchronization state; observation of a S-U-EXCEPTION-REPORT request leads to a transition to the exception state;
- in the resynchronization state, observation of any Request at the acceptor side, or execution of any Indication at that side will effect a transition to normal state;
- in the exception state, any recovery event - in accordance to subclauses 13.11.1 and 13.12.1 of ISO 8326, see also 10.1.3.2 - that is executed or observed at the acceptor side leads to a state transition. The normal state is entered if the event is not a S-RESYNCHRONIZE request, otherwise the resynchronization state is entered.

NOTE - Transitions to exception state caused by a S-P-EXCEPTION-REPORT indication are specified in the subprocesses of *SSPEvent* (see 10.2.3), and not reflected in type *SProviderState*, since they depend on the endpoint where the indication occurs (the functions introduced by the type make no distinction between the endpoints for provider generated indications) and the position of the data token.


```
----------------------------------------------------------------------------------------------------------------------*)
type SProviderState is SProviderState0, SSPBasicClassifiers
opns ProvStateOnObs, ProvStateOnExec: SSP, SProvState -> SProvState
eqns forall p:SSP, s:SProvState ofsort SProvState
IsSRSYNreq(p) => ProvStateOnObs(p,normalState) = resynState;
IsSUERreq(p) => ProvStateOnObs(p,normalState) = excepState; (* see NOTE *)
not (IsSRSYNreq(p) or IsSUERreq(p)) => ProvStateOnObs(p,normalState) = normalState;
ProvStateOnExec(p,normalState) = normalState;

IsReq(p) or IsProvGenerated(p) => ProvStateOnObs(p,resynState) = normalState;
not (IsReq(p) or IsProvGenerated(p)) => ProvStateOnObs(p,resynState) = resynState;
IsInd(p) => ProvStateOnExec(p,resynState) = normalState;
not (IsInd(p)) => ProvStateOnExec(p,resynState) = resynState;
```

```
IsSACTDreq(p) or IsSACTIreq(p) or IsSUABreq(p) or IsSPABind(p) or
      (IsSGTreq(p) and (dk IsIn Tokens(p))) => ProvStateOnObs(p,excepState) = normalState;
IsSRSYNreq(p) => ProvStateOnObs(p,excepState) = resynState;
IsSACTDind(p) or IsSACTIind(p) or IsSRSYNind(p) or IsSUABind(p) or IsSPABind(p) or
      (IsSGTind(p) and (dk IsIn Tokens(p))) => ProvStateOnExec(p,excepState) = normalState;
not (IsSACTDreq(p) or IsSACTIreq(p) or IsSRSYNreq(p) or IsSUABreq(p) or IsSPABind(p) or
      (IsSGTreq(p) and (dk IsIn Tokens(p)))) => ProvStateOnObs(p,excepState) = excepState;
not (IsSACTDind(p) or IsSACTIind(p) or IsSRSYNind(p) or IsSUABind(p) or IsSPABind(p) or
      (IsSGTind(p) and (dk IsIn Tokens(p)))) => ProvStateOnExec(p,excepState) = excepState;
endtype

type SProviderState0 is Triplet renamedby
sortnames SProvState for Triplet
opnnames   normalState for constant1        resynState for constant2        excepState for constant3
endtype
(*-----------------------------------------------------------------------------------------------------------------------
```

10.1.2 Flow control option

Two types of flow control can be exerted by the SS-provider, viz. "basic" and "extended" control; one of the types is selected during SC establishment (by means of the Extended Control QOS component parameter, see 9.1.4.3). Type *SFlowControl0* defines the two flow control type values. Type *SFlowControl* is an enrichment that defines a function, *FlowCtlOn*, which yield the flow control type resulting from a given SSP event (it follows that only a S-CONNECT response or confirm may effect a change of control).

```
-----------------------------------------------------------------------------------------------------------------------*)
type SFlowControl is SFlowControl0, SSPParameterSelectors
opns FlowCtlOn: SSP, SFlowCtl -> SFlowCtl
eqns forall p:SSP, c:SFlowCtl ofsort SFlowCtl
IsSCONAK(p) and (ExtControl(QOS(p)) eq desired) => FlowCtlOn(p,c) = extCtl;
not (IsSCONAK(p)) or (IsSCONAK(p) and (ExtControl(QOS(p)) ne desired)) => FlowCtlOn(p,c) = c;
endtype

type SFlowControl0 is Doublet renamedby
sortnames SFlowCtl for Doublet
opnnames   basicCtl for constant1        extCtl for constant2
endtype
(*-----------------------------------------------------------------------------------------------------------------------
```

10.1.3 History of Requests

10.1.3.1 Basic construction of Request histories

The basic construction of Request history values is accomplished by way of the functions *empty* (a constant denoting the empty history) and *AddLast* (to update the history with the latest SSP event). A few boolean functions are also introduced, with the usual interpretation.