# Syntax and Static Semantics of the Interlocking Design and Application Language

R. N. Bol

*Department of Computing Science, Eindhoven University of Technology,*

*P.O. Box 513, 5600 MB, Eindhoven, The Netherlands*

*E-mail:* `bol@win.tue.nl`

J.W.C. Koorn,
L.H. Oei

*Programming Research Group, University of Amsterdam,*

*Kruislaan 403, 1098 SJ, Amsterdam, The Netherlands*

*E-mail:* `koorn@fwi.uva.nl, lhoei@fwi.uva.nl`

S.F.M. van Vlijmen

*Department of Philosophy, Utrecht University,*

*Heidelberglaan 8, 3584 CS, Utrecht, The Netherlands*

*E-mail:* `vlijmen@phil.ruu.nl`

**Abstract**

An interlocking is a system used at railway yards for guaranteeing safety of train movements. This report opens a series of studies on tightly related specification languages for interlockings. These languages together are called ISL: Interlocking Specification Language. ISL is designed by P. Middelraad of the Dutch railway company, NS, *de Nederlandse Spoorwegen*. NS have developed a symbolic intermediate tool interface language, IDEAL, for ISL. IDEAL is the *Interlingua* for all men and machines using ISL. This report introduces interlockings, the long-term goals of the ISL project, and, as a first step, we present the syntax and static semantics of IDEAL in ASF+SDF, an algebraic specification formalism. Several improvements of IDEAL are suggested.

*Key words & Phrases: Safety-critical systems, application specific languages, railway signaling, interlocking, ASF+SDF, programming-environments*

## 1 Introduction

### A Retrospect of Railway Interlocking

Over many years, systems for safely controlling railway infrastructures have evolved from mechanical ones to ones involving electro-mechanical devices and computers. In the early days safety was enforced by mechanical means only: it involved enabledness of handles which was controlled by shafts, nocks, pins, holes, etc. The word *interlocking* originates from this period. Later, the term interlocking became more equivalent to *safety guaranteeing equipment* in a broader sense. Nowadays, interlockings consist of systems implemented by relays and, increasingly important, computer equipment. Additional control machinery, more or less integrated with the interlockings, is responsible for generating commands to the infrastructure in order to regulate train traffic. A contemporary interlocking *filters* these commands: if a command is safe, it is handed over to the electro-mechanical devices that set or reset points and the like, otherwise it is rejected. Furthermore, an interlocking should take measures when failures of equipment or hazardous situations are detected. In general, the task of *scheduling* train traffic is thus not handled by an interlocking. An interlocking may be allowed to take some small regulation decisions though.

## Future Railway Interlocking Prospects

Needless to say, safety of the infrastructure is of utmost importance to the Dutch Railway Company, NS. NS are currently investing large sums of money in its infrastructure as well as in modernization of its control equipment. Furthermore, external forces require NS to transport more passengers than ever before, with a smaller budget for exploitation. Modernization of infrastructure and control equipment implies the need for new interlockings, the design of which is a very time consuming activity. P. Middelraad of NS therefore concluded that the situation called for a language to describe interlocking behavior in an abstract and unambiguous way. Work on this language, the Interlocking Specification Language (ISL), started in the early eighties. Originally, work on ISL was performed in the context of the Rotterdam Central Station project, later on, work on ISL was continued in commission 7A/16 of L' UNION INTERNATIONALE DE CHEMIN DE FER. Two documents were published on ISL, [6] and [7] [1].

## ISL and Logic Sequence Charts

The philosophy behind ISL is based on two things. First, the observation that every railway yard consists of elements taken from a small set of rail elements such as signals, points, and tracks. These elements may differ from application to application, e.g., the maximum speed allowed at a track may vary. Second, experience has shown that safety can sufficiently be guaranteed by defining interlocking protocols at a local—element—level. ISL assigns a *process* to each rail element. These processes operate in cooperation with neighbor processes of elements in order to guarantee safety. Furthermore these processes are generic, that is, they can be adapted by means of parametrization to serve in most, if not all, situations.

ISL is not just one language. It consists of four sublanguages: TL, for specifying Track Layouts, ECL for specifying Element Connection Layouts, LECL for specifying Logical Element Connection Layouts, and finally LSC for specifying Logic Sequence Charts. LSCs specify the behavior of the processes assigned to rail elements as described above. In LECL, one parameterizes and connects LSCs to form a railway yard. In ECL, one abstracts from LECL, and in TL, one abstracts from ECL. This subclassification can be viewed upon as levels of abstraction. Specification analysis according to the four levels of abstraction is depicted by Figure 1 (after [6]). Three of the four consecutive diagrams, in TL, ECL, LECL style respectively, describe the same fragment of a railway route, which is in this specific case a signal connected to a point. To save space only a fragment of an LSC is drawn.

## ISL Project Goal

Eventually we want to provide *a uniform and unambiguous interpretation of ISL specifications.*

With this aim in mind, we study ISL from a formal language perspective, rather than from an NS perspective. Here, we concentrate on the syntax and static semantics of IDEAL.

[1] In these documents the name EURIS, The European Railway Interlocking Specification, is used. We prefer to refer to it as ISL (Interlocking Specification Language).

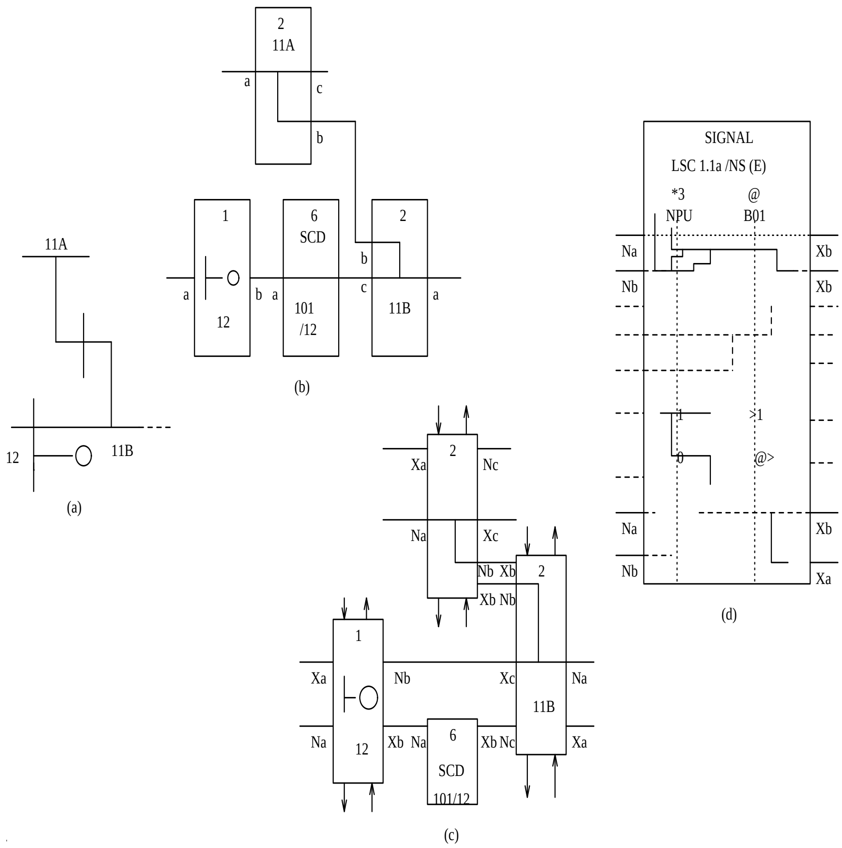


Figure 1: (a) Track Layout - (b) Element Connection - (c) Logical Element Connection and (d) Logic Sequence Chart of the bottom left element in (b)

A semantics is helpful for constructing tools, testing of tools, testing of specifications, and especially for verification of specifications. Furthermore, a semantics may help to solve disputes on the meaning of specifications and it eliminates the need for gurus for the interpretation of specifications. Academically it is of importance to study program construction in industrial languages like ISL. This provides feedback for research in formal semantics of program and specification languages.

The study of the semantics concerns data objects and operational behavior. In this report we concentrate on syntax and static semantics of those data objects. For this purpose we will use algebraic specification techniques [8].

## Intended Readers

With this report we are addressing the following groups of people. First of all, people involved at the NS in the design, development, and use of ISL. Secondly individuals who will be involved in the next stages of this project. Since it is our philosophy that a dynamic project team composition prevents the uprise of gurus, we have to make sure that previous results are efficiently communicated to the next generation of project members. This document might therefore give the impression to be overweighed with respect to some housekeeping information. This is an attempt to concentrate in one document all concepts involved in this project that can be found in primary and secondary literature, and also in people's mind. Topics which are necessary to touch upon but do not fall within the scope of this study are discussed in simple, intuitive, and perhaps also a bit naive way.

## Paper Overview

To familiarize the reader with interlockings we present an introduction in Section 2. The origin of IDEAL, its current status, and some ways to achieve a formal semantics for ISL are discussed in Section 3. Section 4 presents an overview of the IDEAL language. This overview is then used in Section 6 where a number of static requirements on IDEAL programs are listed. These requirements are algebraically defined in the ASF+SDF formalism which is introduced in Section 5. The definitions themselves are to be found in the appendices. Section 7 lists some improvements for the syntax of IDEAL. Finally, we list related work in Section 8 and we close with a discussion in Section 9.

## Acknowledgements

We like to acknowledge Jan Bergstra and Peter Middelraad as being the initiators of this project. Furthermore we wish to thank André Klap and Frits Makkinga for answering our questions on IDEAL and for providing us with all the documents and sources, Steven Klusener for suggestions of improvement of this paper, and Paul Klint for his guidance. Finally we acknowledge David Griffioen and Frits Vaandrager for sharing their LaTeX I/O Automaton specification macro package.

# Contents

## 2 An Introduction to Railway Interlocking

Consider two travelers catching a train at station Amsterdam Amstel—possibly two members of the Programming Research Group— heading for a meeting in an office nearby Utrecht Central Station the Engineers Bureau of the NS at 10:00 AM.

One of them, being a morning person and not really bothered to consult the timetable, boards the local train of 9:21 AM. The other has perfectly synchronized his alarm clock with the departure of an intercity of 9:26 AM. The local train is the first to arrive at the station of the suburb Abcoude at 9:32 AM. It has to let the intercity pass by, since they both use the same track. The intercity passes Abcoude about 9:34 AM and arrives at Utrecht Central at 9:47 AM. The local train departs from Abcoude station at 9:35 to reach its destination at 9:53 AM, now being the last of the two trains to arrive as it should be[2].

In this technical document, we are not interested what has become of the two passengers in the different trains, nor in the meeting that they are supposed to attend. We are interested in what actually happened in Abcoude from a Railway Interlocking point of view. We abstract from NS practice and try to visualize the situation in a more intuitive manner. It is depicted in Figure 2 (after [33]).

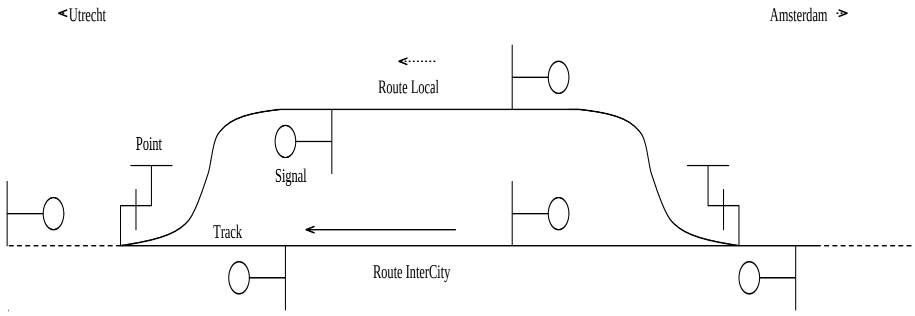


Figure 2: Station Abcoude at 9:34 am.

At an abstract level it is imaginable for the railway interlocking activities involved at this situation to be as follows. First, from higher control level—the man machine interface—which is in this case effectively an operator of a control panel, a request for reservation (locking) for the route is made. We now make the assumption that the control level already *knows* that an intercity is coming soon, and that the local train has to wait for the intercity to pass. Therefore, the entry point at the incoming side has to be set to *right* and the exit signal for the local has to be set to *red*. Next, after the local has made its standstill at the platform, the entry point has to be adjusted for the intercity. Only after the intercity has passed the station or the route has been manually released or canceled for some reason, the local train can be given its exit signal to proceed its course. It is clear that the signal in front of the waiting local may never turn green before the route has been released, i.e., the interlocking should guard against crashes; it is also immediate that this same signal in front of the local train has to turn green eventually, even if the intercity should be hijacked before entering Abcoude, i.e., the interlocking should guarantee liveness [3].

Railway Interlocking processing basically consists of: the setting of routes, the release of routes (either manually or automatically), and approach monitoring. The process of route setting can further be subdivided in route proving (availability

[2]Data taken from the *NS Timetable* '93-'94.

[3]We suggest interested readers who want to get a more vivid demonstration of how railway interlocking is done (in the past) to visit the Railway Museum in Utrecht.

test), route marking, and route locking. The process of Railway Interlocking is an intricate interplay of several levels of communications and control, this is depicted in Figure 3 (after [6]).

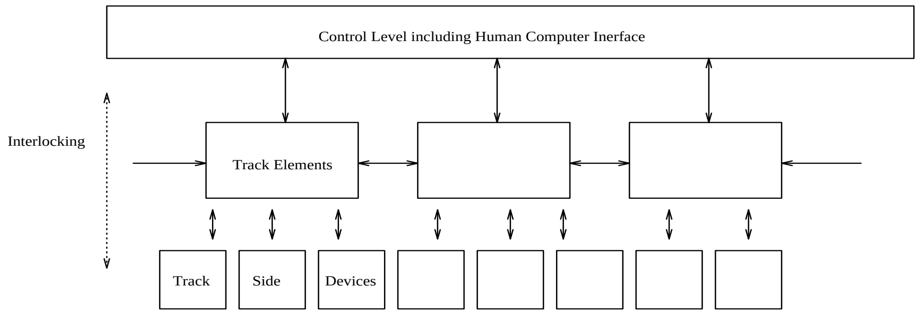


Figure 3: Hierarchy of Railway Interlocking

From this discussion, some basic safety criteria could be derived. Morley suggests the following criteria [32]:

1. No more than one of the sub-routes over a track section is set at any one time.
2. If a route is set, then all its component sub-routes are set.
3. If a sub-route over a track section containing points is set, then the points are correctly aligned with that sub-route.

It is clear that the above description of signaling practice is an oversimplification of reality. Work in this area is currently progressing in the United Kingdom, at the firms WESTINGHOUSE SIGNALS LTD., GEC ALSTHORN SIGNALING LTD., and at the University of Manchester, Institute of Science and Technology. The safety criteria that railway interlockings should obey are specified from scratch, by applying hazard analysis methods used for nuclear installations [36].

# 3 The Interlocking Design and Application Language

## 3.1 Origins

At the *Ingenieursbureau* of the NS, *Sector Elektrotechnische Systemen, Department Design Automation*, Frits Makkinga and André Klap [31] designed the language IDEAL, a non graphical representation of LSCs and LECLs (Logic Sequence Charts and Logical Element Connection Layouts), using LEX and YACC [30, 22]. Routes in IDEAL, which are the Logical Element Connection Layouts in ISL, can be viewed as a *connectivity graph* with LSCs as nodes. Tools were developed for drawing LSCs, translation of these drawings to IDEAL, and translation of IDEAL to VHDL [31]. VHDL is a standard IEEE-1076 simulation language for which commercial simulators are available on PCs and Workstations [4].

[4]VHDL-1076 (`VHSIC` (Very High Speed Integrated Circuits) Hardware Description Language) is an IEEE Standard since 1987. It is

> "a formal notation intended for use in all phases of the creation of electronic systems. ... it supports the development, verification, synthesis, and testing of hardware designs, the communication of hardware design data ...".

[Preface to the IEEE Standard VHDL Language Reference Manual]

## 3.2 Status and Development

Figure 4, taken from [23], depicts the current status of the IDEAL language within the ISL project at the NS.

LSCs are drawn by hand with the commercial CAD application ACE+ that is marketed by INTERGRAPH BENELUX BV. The textual representation of these drawings is automatically translated to IDEAL code using a compiler that has been built by HOLLAND SIGNAAL. The output of this transformation process serves as input for an IDEAL to VHDL compiler, which is an NS in house product.

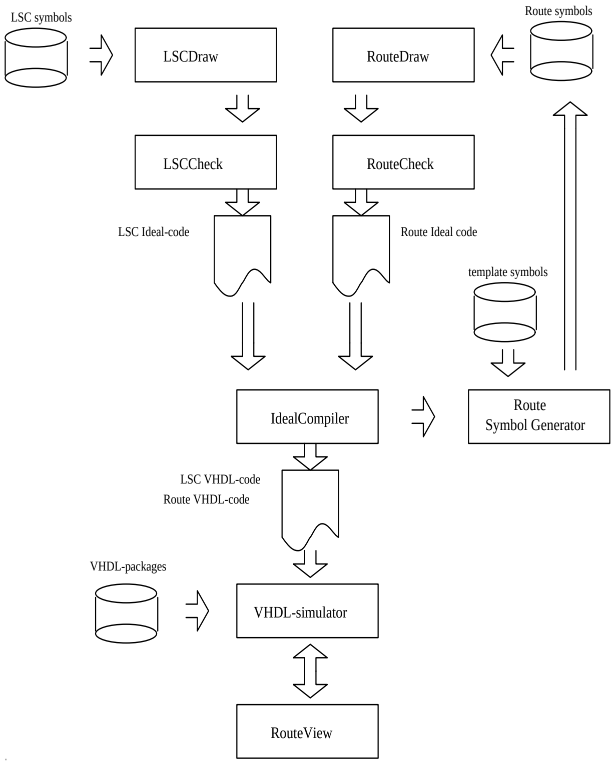


Figure 4: LSC to IDEAL to VHDL Transformations

It is clear that the current status of IDEAL is therefore a *machine interface language*.

The current study has been based upon *preliminary* versions of the "IDEAL Guide and Reference" [31] and the "Euris-Simulation Tutorial" [23]. As yet the final draft of these manuals are not available. Additional information was drawn from "The EURIS Document" [6].

## 3.3 IDEAL as an Interlingua

Now, after all preliminary remarks to get the overall picture of this project, we return our focus to this specific study, which is the evaluation of the IDEAL language. Our strategy has been to redo the exercise of defining a syntax for IDEAL, albeit in ASF+SDF instead of LEX&YACC and to try to provide a comprehensive list of checks on the static semantics of IDEAL. For this purpose we made extensive use of the ASF+SDF Meta-environment [24] shortly introduced in Section 5 below.

Within our project we are considering a number of translations. These translations are briefly discussed below. Figure 5 depicts an overview of them.

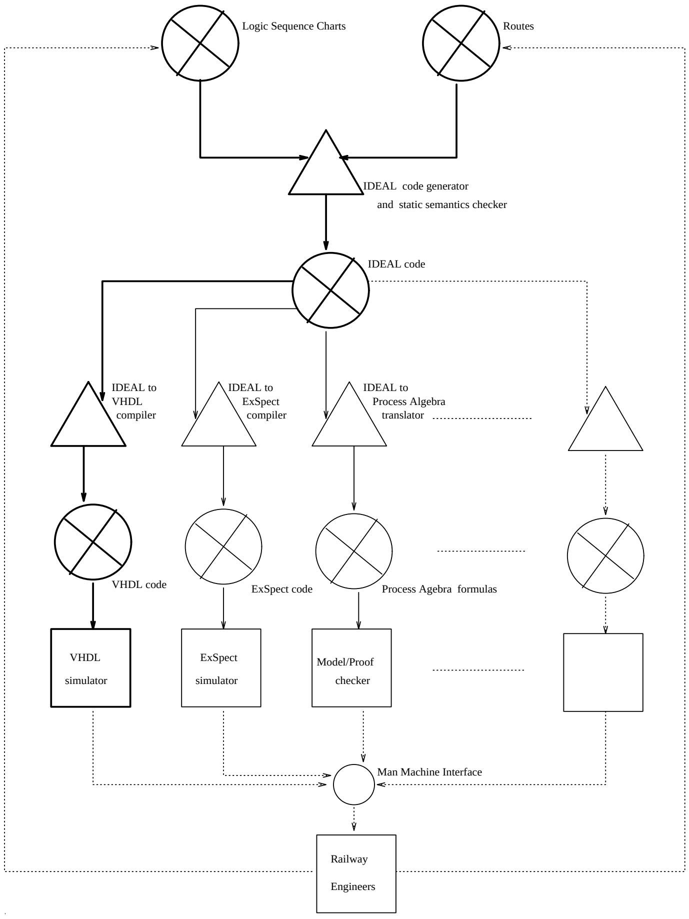


Figure 5: Petri Net depiction of IDEAL as an Interlingua within the ISL project.

### Operational Semantics and Validation

**IDEAL to ExSpect.** ExSpect—an Executable Specification Tool—is a commercial specification and graphical simulation package developed at the Technical University of Eindhoven and marketed by Bakkenist Consultancy. ExSpect is also used by the Exploitation Department of the NS for simulating train traffic.

The theory behind ExSpect is based on Petri Nets [35]. This thirty years old theory has been extended with Coloring (parametrization), hierarchy of Petri Nets, and time stamping. A translation from IDEAL to ExSpect is a serious option when we decide to look for a ready at hand operational semantics for validation purposes. For a more thorough treatment of this subject we refer to [1, 5].

### Axiomatic Semantics and Verification

**IDEAL to Process Algebra.** Railway Interlocking is time critical. Therefore it seems to demand a Real Time variant of Process Algebra [4, 3].

Another variant of Process Algebra is $\mu$CRL [19]. This is a formalism specially designed for reasoning about processes with data. The major advantage of this formalism is that there is currently a considerable amount of activity in developing tools and theory for it. A side effect of using this formalism for this project would be that it will provide further insights in its strengths and weaknesses.

To sketch the possibility of translating IDEAL to $\mu$CRL, consider the following hand crafted example.

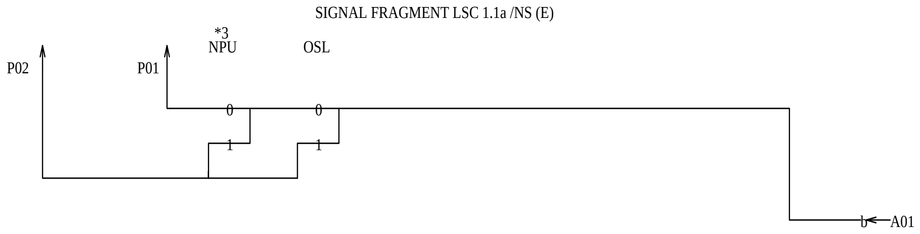


Figure 6: Fragment of a Logic Sequence Chart

This fragment translates to the following IDEAL LSC Program below:

```
LSC_PROGRAM SIGNAL;    // Top Flow of Signal LSC 1.1a EURIS document
LSC_VAR
  OSL : BOOL_TYPE;
  NPU : BOOL_FUNC_TYPE;     // function declaration returns bool

LSC_MERGE LABEL_1;
  LSC_SEND P02:MMI;
LSC_MERGE_END;

LSC_FLOW A01:LBRL;
  LSC_SPLIT OSL=0;
    LSC_SPLIT NPU=0;
      LSC_SEND P01:MMI;
    LSC_SPLIT_END;
    LSC_SPLIT NPU=1;
      LSC_CONT_AT LABEL_1;
    LSC_SPLIT_END;
  LSC_SPLIT_END;
  LSC_SPLIT OSL=1;
    LSC_CONT_AT LABEL_1;
  LSC_SPLIT_END;
LSC_FLOW_END;

LSC_PROGRAM_END;
```

The above IDEAL fragment could plausibly be mapped to Process Algebra with Data using the $\mu$CRL Formalism as follows.

```
%%   Int module defined elsewhere
sort TelegramName, TelegramPort, ParVar, ElementId
func signal                                    -> ElementId
     MMI, LBRL:                                -> TelegramPort
     A01, P02, P01:                            -> TelegramName
     OSL: ParVar                               -> Int
     NPU: ParVar                               -> Int
     PV: Int                                   -> Int
var  pv:                                          ParVar
     int0:                                        Int
rew  %% NPU defined elsewhere analogous to Ideal convention
     OSL(PV(int0)) = int0
act  send, receive, communicate: ElementId # TelegramName # TelegramPort
comm send | receive = communicate
proc Signal(tn: TelegramName, tp: TelegramPort, pv: ParVar) =
        receive(signal, tn, tp) . Flow(tn, tp, pv)

     Flow(tn: TelegramName, tp: TelegramPort, pv: ParVar) =
```

```
    SplitA01LBRLOsl(tn, tp, pv)
       <| and(eq(tn, A01), eq(tp, LBRL)) |> delta

  SplitA01LBRLOsl(tn: TelegramName, tp: TelegramPort, pv: ParVar) =
    SplitA01LBRLNpu(tn: TelegramName, tp: TelegramPort, pv: ParVar)
      <| eq(OSL(pv), 0) |> delta

  SplitA01LBRLOsl(tn: TelegramName, tp: TelegramPort, pv: ParVar) =
    MergeA01LBRL(tn, tp, pv) <| eq(OSL(pv), 1) |> delta

  SplitA01LBRLNpu(tn: TelegramName, tp: TelegramPort, pv: ParVar) =
    send(signal, P01, MMI) . Signal(tn, tp, pv) <| eq(NPU(pv), 0) |>
    delta

  SplitA01LBRLNpu(tn: TelegramName, tp: TelegramPort, pv: ParVar) =
    MergeA01LBRL(tn, tp, pv) <| eq(NPU(pv), 1) |> delta

  MergeA01LBRL(tn: TelegramName, tp: TelegramPort, pv: ParVar) =
    send(signal, P02, MMI) . Signal(tn, tp, pv)
```

An element, here a `Signal(...)`, displays the overall behavior of a process with a global data structure associated `pv:ParVar`, waiting for an incoming message `tn:TelegramName`. Depending on the telegram it receives and the sender, denoted by `tp:TelegramPort`, it makes a transition to a state where it checks and updates its global data structure and either retransmits the message, reacts with reply message, or simply remains to its original waiting state. `Split(...)`-s and `Merge(...)`-s in this formalism become threads of subprocesses associated with a specific telegram, its port of entry and the data store involved.

**Other options.** Logical Sequence Charts has most elegantly been expressed using the I/O Automaton formalism. See [21] for an application of I/O Automata for modeling railway interlockings. With a little handwaving, the above fragment, now an I/O Automaton Signal, then reads as follows.

## Signal

| | | |
|---|---|---|
| **Input** | $FlowLBRL$ | |
| **Output** | $SendMMI$ | |
| **Discrete** | OSL | **Bool** |
| | NPU | Int |
| | ct | TelegramName |

**Initialization** $ct{=}NoTelegram$

$SendMMI(tn : TelegramName)$

**Precondition**

$(tn{=}P01 \rightarrow (ct{=}A01 \wedge OSL{=}0 \wedge NPU{=}0)) \wedge$
$(tn{=}P02 \rightarrow (ct{=}A01 \wedge (OSL{=}1 \vee NPU{=}1)))$

**Effect**

$ct := NoTelegram$

$FlowLBRL(tn : TelegramName)$

**Precondition**

$ct{=}NoTelegram$

**Effect**

$ct := tn$

Using the I/O Automaton model makes the internals of an LSC quite transparent, which comes down to taking the following steps:

- accept incoming message;
- test on certain conditions with side effects; and
- reroute the message accordingly.

# 4 IDEAL, a Quick Language Overview

This section does not replace the original "IDEAL Guide and Reference" [31], in which the IDEAL language is defined in terms of a notation that is similar to BNF and natural language. With this section we want to provide a quick overview over the language, highlighting some major characteristics and pointing at features that are not immediately clear at first reading of the IDEAL Guide&Ref. Some of these features have been elaborated in the "Euris-Simulation Tutorial" [23].

This section should not be read as a language tutorial. When learning a new language, typically a programming or specification language, one usually is concerned to find the semantics of those language constructs that are useful for the things we want to express. In this study, however, we want to make sure that IDEAL is able to express all valid LSCs and Routes that the NS might want to draw.

In other words, we need to put the compiler writer hat on our heads and asks questions like, "what does this mean?" and "where does this come from?", rather than wondering what we might want to write down using IDEAL grammar and vocabulary.

## 4.1 Main Components

### 4.1.1 Classification, Sublanguages and Side Files

The IDEAL language consists of two separate languages for LSCs and Routes. Recall that LSCs in ISL are Logic Sequence Charts of individual Track Elements. For example, the graphical representation of track element Signal has its own collection of LSCs spread over several pages. With the symbolic representation of LSCs however, the notion of 'Charts' has become somewhat obsolete. Instead the word 'Program' has been adopted.

A source of confusion might be introduced here. Whereas the *S* of ISL stands for *Specification*, in IDEAL—a *Design* and *Application* Language—the link has explicitly been made with operational semantics. It seems that the designers of IDEAL have tried to emphasize the hybrid character of this language, that for instance features the assignment statement. Most general purpose specification languages, however, are functional languages without assignments. One last very important characteristic of specification languages is that without exception they are all meant to be readable for humans; although we admit that there is a trend to attach a graphical interface to them. We have made our point of current usage of IDEAL at daily NS practice above. Having said this, we stop dwelling on the philosophical issue whether IDEAL is a specification or a programming language. We hold onto our original point of view, which is: IDEAL, within the boundaries of this study, can most conveniently be considered as an *Interlingua*.

IDEAL as an application language does fall in the category of imperative programming languages. The IDEAL language, however, is very weak in the sense that it supports neither iteration nor recursion, i.e., iteration is technically possible, but is not considered good LSC drawing practice; recursion on the other hand is forbidden and this is checked by the IDEAL compiler. Obviously, there must be a run-time loop that executes LSCs over and over again. An IDEAL Program carries a global data structure and does support user defined functions. An IDEAL LSC Program can be best defined in terms of its origin, namely quoting [6]:

> A Logic & Sequence Chart (LSC) is a chart which shows the logical activities that occur within an **element**, how they are initiated, the sequence *(Flow)* in which they occur and the resultant action taken.

The track elements that are given as examples in [6] are [5]:

1. signal
2. point
3. track
4. approach monitoring
5. approach starting
6. signal clearance delay

To the approach monitoring track element, a *central element* is associated, namely: the approach monitoring central element. The approach monitoring central element functions as a control process that coordinates communications of all track related approach monitoring devices.

The last three elements of the list above are *functional elements*, again quoting [6],

> ... since they perform a function which is not related to a track side device at the place where it is used in the LSC sequence.

Route Programs are the modules in which LSCs are instantiated. That is, LSCs are connected and parameters and variables are bound.

Because ISL has been proposed as standard specification language for European railways, the designers of IDEAL have also left out the definition of data, at least in principle. This means that another Railway company with different signaling practice, can use the language definition without inheriting the data definition as well. So parameters, variables and telegrams with their associated data fields are defined in separate *side files*. These side files have also a fixed syntax but they are not considered to be part of IDEAL.

We now present the definition of LSC Programs.

### 4.1.2 LSC Program Top View

An LSC Program has the following syntax[6]:

```
<ideal-program> ::= <program-start-block>
      [<configuration-block>]
      [<parameter-block>]
      [<variable-block>]
      <function-block>*
      <merge-block>*
      <flow-block>+
    <program-end-block>.
```

[5]Other railway companies may define their own set of elements

[6]IDEAL is case insensitive. The semicolon ';' is used as a delimiter. Note that the period '.' after each production rule is part of the *extended BNF* that is used here.

### The IDEAL Preamble

The <configuration-block> is the program header containing documentary information like version number etc. It has no operational semantics—with the exception of the TARGET field, which serves as a meta toggle for the compiler. This 'configuration field' can therefore be most conveniently be viewed as a *compiler directive*.

### IDEAL Declarations

Parameters in the <parameter-block> are global constants associated with the logic circuit of the element in which they are defined. A mix of Variable and Function declarations and 'process behavior' (associated with certain qualifiers) can be found in the <variable-block>. Parameters and Variables in the declaration blocks may have identical identifiers. These are disambiguated outside the declaration block by means of the P. V. prefixes, respectively. This is, for instance, necessary to prevent the occurrences of Parameters at the left hand side of an assignment. Other restrictions for Parameters are listed in Section 4.2.8.

### 4.1.3 User defined Functions and Function Statements

A Function block has the following syntax:

```
<function-block> ::= lsc_function <par-var-identifier> ;
                         <function-statement>*
                         <merge-block>*
                     lsc_function_end ;.
```

An LSC Function is a 'real' function in the sense that it performs its computations locally with respect to the program structure, on the basis of the global state and returns a value to its caller. Functions can be nested. Functions cannot call themselves. Functions are not parametrized. Functions do not have local constants or variables. The seemingly undeclared identifiers that occur in all Ideal Program block definitions are Telegram Data fields and are of type Integer by default. See Section 4.2.9 for more information on Telegram Data.

In the scope of a Function definition, only 'Function Statements' may be used. These are: lsc_set, lsc_continue at, lsc_split, which are effectively the *assignment*, the *goto* and the *case* statement.

In blocks outside the scope of functions, however, three other statements may occur: lsc_test, lsc_send, lsc_cnt_send, which are effectively the *if*, the atomic communication action *send*, and the atomic communication action *central send* respectively.

Now that we have covered the LSC Statements, note that <merge-block>* are simply starting points of blocks of IDEAL code to jump to, for a flow to continue at. The crucial difference between a <flow-block> and a <merge-block> is that the former is always triggered by a telegram or an telegram initiating variable and may never start from a label, while the latter always begins at a point where at least two different flows meet. We have understood this as a semantical constraint rather than experience of usage.

### 4.1.4 Flowblocks

Flow blocks, the building blocks which define the element process behavior, have the following production rules:

```
<flow-initiator>  ::=  <telegram-name> : <telegram-in-type>
                     | [<par-var-tag>] <par-var-identifier> <initiator>
```

```
                          [( <direction-list> )]
                        .
<par-var-tag>     ::=   V\. | P\.
                        .
<flow-block>      ::=   lsc_flow <flow-initiator>;
                          [<statements>]
                        lsc_flow_end;
                        .
```

The other language elements are described below.

## 4.2 LSC Built-in Types and Their Usage

### 4.2.1 Booleans

From an Algebraic Specification point of view, we characterized the IDEAL Booleans to be the lexical tokens "1" and "0". Boolean Expressions, however, are in LSCs expressed by telegram flow. For instance `lsc_split` might be considered as an 'or', while a continuous flow represents boolean expressions 'and-*ed*'. In IDEAL, Booleans variables or parameters are declared with the keyword `BOOL_TYPE`.

### 4.2.2 Integers

The syntax and semantics of Integers are rather straightforward. For details see the specification in Appendix 9. The only point of interest here is the lexical ambiguity of Ideal Booleans with Integers. A suggestion for improvement is given in a Section 7. In IDEAL, Integer variables and parameters are declared with the keyword `INT_TYPE`.

#### Integer Operators

All arithmetic operators, with their arities in superscript, $-^1, +^2, -^2, *^2, |\ \ |^1$ apply only to expressions of type Integer and must fall within the range [-$2^{31}$..$2^{31}$].

#### Relational Operators

Integers are also the sole type to be subject to an ordering. Therefore all relational operators $<>^i, <^i, >^i, <=^i, >=^i$, with $i \geq 2$, are only applicable to expressions of Integer types, and evaluate to Boolean values. Note that IDEAL relational operators are *varyadic*, which means that they can take any number of arguments greater than two. This is a different notion from *associativity*, which means that in case of left associativity the expression `1 < 2 < 3` is implicitly interpreted as `(( 1 < 2 ) < 3 )`. This phenomenon is not documented but has been affirmed by its designers and this is also suggested by LSC 6.1a/NS example in [6]. For example, one could write test expressions of the form `1 < 2 < 3`. This introduces a too great degree of freedom. In Section 7 some suggestions for improvements are given.

#### Equivalence

The equivalence operator = applies to IDEAL Integers, Booleans and Enumerated Types only. It evaluates to a Boolean value.

### 4.2.3 Enumerated Types

In IDEAL, Enumerated Types, `ENUM_TYPE`, are user defined types in the sense that they are predefined as a comma separated list of `Letters` defined in either the parameter or the variable side file. For example the following entry in a parameter side file:

```
DPP : ENUM(L,R)  : Dpp is an Enumerated Type with possible values 'L', 'R'.
APP : ENUM(X,Y,Z): App is an Enumerated Type with possible values 'X', 'Y', 'Z'.
```

Note that in the declaration block of an LSC Program these variables would be simply declared as `LSC_PAR DPP: ENUM_TYPE; APP: ENUM_TYPE;`.

### 4.2.4 List Types

IDEAL allows Lists of Booleans and Integers, `INT_LIST_TYPE, BOOL_LIST_TYPE`. Algebraically, they are best defined in terms of *mappings* or *tables*, i.e., each list element has an *unique label* associated to it. List element retrieval is denoted as:

```
<par-var-identifier> [ <simple-expression> ]
```

where `<par-var-identifier>` is of type list and `<simple-expression>` within the [ ] list access operator returns either an Integer or a Boolean value.

List element initialization uses the following shorthand in Route Programs:

```
PSPS : (12, 12), (4, 12), (0, 4)
```

means that the elements of PSPS at labels 4 and 12 each get the value 12, and the element at address 0 (not necessarily the first element, since list addresses could be negative) gets the value 4. If a node at an address does not yet exist, it is appended, otherwise the node is updated to the new value. Thus, the length of the list equals the number of appends applied to it. The order in which the list is traversed depends on the implementation and is not discussed here.

It is not possible to compare two lists. Two list elements (of type Booleans or Integers), however, can be tested on equivalence.

### 4.2.5 Function Types

Function Types, `INT_FUNC_TYPE, BOOL_FUNC_TYPE`, declare the result types of LSC functions. For this reason Parameters, which are IDEAL constants, cannot be of type `FUNC_TYPE`. So

```
lsc_var NPU : INT_FUNC_TYPE;
```

would be interpreted in Pascal as a function declaration:

```
forward function npu(): integer;
```

### 4.2.6 Central Types and Central Parameters

Central Types, `INT_CNT_TYPE`, `BOOL_CNT_TYPE`, with their associated `<central-parameter-list>` are just a syntactical feature to allow a special kind of data parametrization.

```
<central-parameter-list> ::= {<identifier>}
                           | {<identifier>, <identifier>}
                           | {<identifier>, <identifier>, <identifier>}.
```

Typical instantiations for `<identifier>` are `track` or `side`. The arity of three correctly suggests the three spatial dimensions. The third dimension is reserved for future extension and is currently not used in LSC design. The typical value of `side` is 2, because normally trains can come from both sides of the tracks. The values of the associated variables determine the actual dimensions of the track element layout. So in effect a *central parameter list* induces a data structure that closely resembles *a two dimensional array of tracks and sides*. Example (after [31]):

```
lsc_var
  AMV: bool_cnt_type {track, side};

lsc_cnt_flow T01: MMI {track, side} // with, e.g., associated values 2
  lsc_set AMV:0;                    // and b bound at the Route level.
  lsc_cnt_send U01:MMI {track, side};
lsc_cnt_flow_end;
```

means that the telegram `T01` is received from approach monitoring of, e.g., track 2 side b, then after the 2b version of AMV is set to the value of 0, a send communication action is performed involving a central element carrying a message named U01, through the machine man interface (MMI) port, to the element associated with track 2 side b.

### Central Parameter Operators

Two conditional operators can be applied to variables of Central Type: `{*}, {#}`; quoting [31]:

> To access and to apply a test or set statement to all instantiations and continue if *all* or if *at least one* is true respectively.

These operators resemble the *Universal* quantifier $\forall$ and the *Existential* quantifier $\exists$.

### 4.2.7 Telegram In Types, Telegram Out Types, and Directions

The terminology of telegram i/o types is slightly misleading. What is meant here are the communication ports that are attached to a certain element. There are three kinds of communication ports: HORIZONTAL TELEGRAMS, VERTICAL TELEGRAMS, and CENTRAL TELEGRAMS.

The ports for horizontal communications, i.e., from track element to another track element, are encoded as follows:

```
<element-port-name> ::= <category> <direction> <flow-direction>.
<category>          ::= L.
<direction>         ::= A | B | C | D.
<flow-direction>    ::= LR | RL.
```

E.g., `lsc_send A02:LBLR` means: the send communication action of a telegram named A02 over the element data port (the category L) that is associated with the connection point of the current element labeled with a 'B' in the direction of the train movement ('LR'). It is very important to note here that 'LR' represents the direction of train movement, while 'RL' stands for the opposite direction of train movement. This means that 'right' and 'left' here have nothing to do with the right or left side of an LSC drawing. Therefore, they do not represent graphical information.

The ports for vertical communications, i.e., communication ports involving the man machine interface or track side elements, are `MMI` and `TRK` respectively.

The ports for communications with central elements are denoted by `CNT_UP`, `CNT_DOWN`.

### Directions and Sides

Directions in LSCs are used to distinguish different versions of parameters or variables with the same Identifier. Furthermore, one uses directions to encode horizontal communication ports, i.e., ports dealing with inter track element communication.

See the previous section on Telegram Types. The notion of direction is strongly attached to the *direction of train movement*. 'A' is normally the entry *side* with respect of train movement, while 'B' is the exit *side* of train movement. It is very tempting to suggest that Directions are simply the communications ports of elements. This is, however, an oversimplification: there are other communication ports associated with an element—the ports for vertical communications—and Directions are also used for tagging variables, as explained above. We may safely state, however, that *directions* are among others used as subclassifications of communication ports. Consider the following example:

```
a/bc
XYZ
```

means that there is a double version of variable XYZ, namely: XYZ(a) and XYZ(bc). While

```
   a/bc
   XYZ

---1*--
```

means that the first version from left to right is tested whether it has the value 1. In this case XYZ(a). The `*` is introduced in [23] as the *don't-care-character*.

The notion *side* is actually used in [6] everywhere *direction* is used in [31]. Most notably quoting [6]

> Each of the telegram flows received at either the a, b or c side of the element branches according to the value...

In this document we consider them to be interchangeable.

#### 4.2.8 IDEAL Qualifiers

IDEAL improves ISL with respect to Qualifiers. Most ISL Qualifiers have been well analyzed and subcategorized into types, shorthand, and commentary.

The IDEAL Guide and Reference [31] mentions the following categories:

1. Directions
2. Data types
3. Initiators
4. Miscellaneous
5. Additional information

`!`, $\diamond$ are treated to be of commentary nature by the IDEAL language designers, and do not feature in IDEAL; `!` for instance, is a qualifier that shows that the value of the associated variable changes to the status of a track-side device. This aspect can be taken into account by a simulator, providing the user an extra facility for generating track-side data. Since IDEAL does not support these qualifiers, we leave these qualifiers for further research.

#### Initiators

Flow Initiators form a challenge when looking for the operational semantics of an LSC. They represent complex process behavior. Initiators have the following syntax:

```
<initiator> ::= @ | $ | & | I | P
```

Informally, these initiators have the following meaning:

**@** after a variable means: if variable set, then repeat send until variable reset.
**$** after a variable means: if variable set or reset, then send once.
**&** after a variable means: if variable set, then send once;
reset variable.
**I** after a variable means: if variable set, then start timer;
stop timer when variable is reset.
**P** after a variable means: if variable set, then start timer;
stop timer when variable is reset;
when timer-delay is expired send telegram once
and reset variable.

For instance, the IDEAL statement `lsc_set V. B05(A, @):1` means: set the variable B05-version-A cyclic to 1 (read: start the cycle). The 'A' has nothing to do with the port where the telegram has made its entry but denotes its version (the 'A' direction version of variable `B05`). This statement has the effect of repetitive send communication actions of the B05 Telegram until the telegram initiating variable B05 is reset.

### 4.2.9 Telegram Data and Ranges

As mentioned, Telegram Data is not explicitly defined in IDEAL. Telegram Fields are currently typed as Integers by default. When a Telegram is initiated it does not hold any fields at all. Valid ranges are associated with each Telegram Field. The current implementation of the data structure of telegrams is the dynamic linked list. When a field is set within a flow and the field is not present in the list, then the field is instantiated, set, and appended to the list of the telegram.

#### Other Communication Related Topics

Within this paper we are not trying to address questions how communications actually take place: asynchronous or synchronous, queuing or polling of telegrams and the like. We leave the elaboration of these operational semantics for further study.

## 5 The ASF+SDF Specification Formalism

ASF+SDF [8, chapter 6] is a modular algebraic specification formalism in which both the syntax and the semantics of a language can be described. It is a combination of two formalisms SDF (Syntax Definition Formalism) [20] and ASF (Algebraic Specification Formalism) [8, chapter 1]. ASF+SDF is developed in the GIPE (Generation of Interactive Programming Environments) project[7] which aims at generating interactive programming environments from formal language definitions. Within the GIPE project, an interactive system has been implemented for the development

[7]Esprit contract nr. 2177.

of ASF+SDF specifications, called the *ASF+SDF Meta-environment* [24]. The user-interface of this system is based on a collection of generic editors, all instances of GSE (Generic Syntax-directed Editor) [26, 27], a hybrid syntax-directed editor. In the ASF+SDF Meta-environment, a user can not only develop a language definition, but also *use* the language definition immediately in one or more GSE instances, which thus form the generated environment. The User Interface of GSE instances in the generated environment may be customized by adding buttons, menu entries, etc. by using SEAL [25, 27].

As an example of how ASF+SDF is used to define a language, consider the language of Boolean expressions. This ASF+SDF specification consists of two modules, one defining the syntax of the language (Figure 7(a)) and the other defining its semantics. (Figure 7(b)).

```
module Boolean-syntax
imports Layout
exports
  sorts BOOL
  context-free syntax
    true          -> BOOL
    false         -> BOOL
    not BOOL      -> BOOL
    BOOL and BOOL -> BOOL {assoc}
    BOOL or  BOOL -> BOOL {assoc}
    "(" BOOL ")"  -> BOOL {bracket}
  priorities
    not > and > or

          (a)
```

```
module Boolean-semantics
imports Boolean-syntax
  variables
    b                   -> BOOL
equations
  [1] true  and b = b
  [2] false and b = false
  [3] true  or b  = true
  [4] false or b  = b
  [5] not true    = false
  [6] not false   = true

          (b)
```

Figure 7: (a) Module Boolean-syntax, (b) Module Boolean-semantics.

In module `Boolean-syntax` (Figure 7(a)), the syntax of the Boolean language is defined by introducing a *sort* `BOOL`, which contains two *constants* `true` and `false`. Furthermore, the standard Boolean functions `and`, `or` and `not` are defined as well as parentheses. The attribute `assoc` declares `and` and `or` as associative functions and the *priorities* section of the definition defines function grouping when no parentheses are present. Module `Boolean-syntax` imports the module `Layout`, in which layout characters as well as a comment convention are defined. A `Layout` module can be found in ASF+SDF definition of the IDEAL language in Appendix 9.

In module `Boolean-semantics` (Figure 7(b)) the semantics of the Boolean language is defined by introducing a *variable* `b` of sort `BOOL` and *equations* which define equalities on Boolean *terms*, thus providing a semantics for the Boolean language. The principle of defining functions like standard Boolean `and`, which does nothing more than filling in a boolean table of truth values, can be generalized to functions that perform transformations on terms.

## Using ASF+SDF to define a programming environment for IDEAL

The language of the Booleans is a small language indeed, but one can imagine large terms of Boolean expressions or functions that takes large language constructs as input terms, like for instance an IDEAL program.

We might want to define a function `check-type(Ideal-program)` → `BOOL` or a function `translate(Route-program)` → `ExSpect-spec` that compiles a IDEAL Route program into a suitable ExSpect specification.

After defining the syntax of IDEAL displayed in the Module window, we combine the facilities offered by the Generic Syntax-directed Editor (GSE) [26], the Semantics-directed Environment Adaptation Language (SEAL) [25] and some pretty printing services for error reporting, to obtain the generated programming environment for IDEAL depicted in Figure 8.

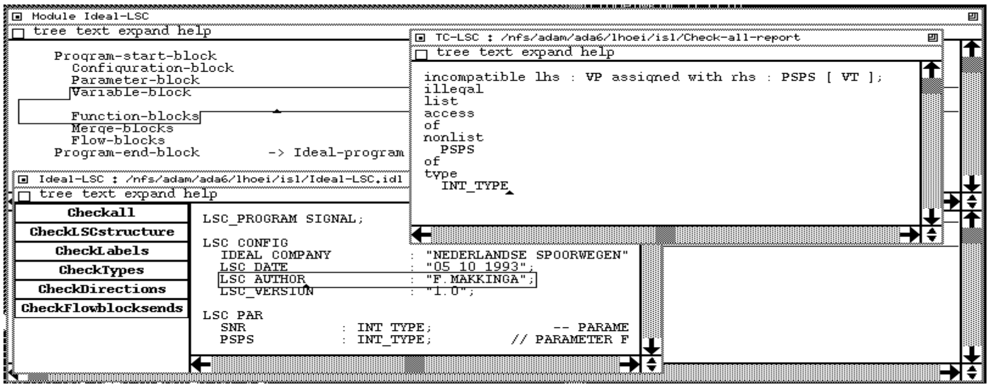


Figure 8: An IDEAL Programming Environment in ASF+SDF and SEAL.

All three windows in this screen are instances of GSE. The main window contains the main module of the SDF definition of the IDEAL language. The window with the check buttons attached contains a sample IDEAL LSC Program taken from [31]. The focus displayed on the `Ideal-LSC` window and the `Ideal-LSC.idl` window tell us that the terms contained in these windows are syntactically correct [8]. The SEAL formalism manages the communication between the User Interface—for instance the `Checkall` button—and the GSE instances by means of a user defined SEAL-script (not shown). The typecheck error report, also a term, has been preprocessed by a standard pretty-printer before display.

# 6 Static Semantical Correctness of IDEAL

In this section we suggest a number of checks on IDEAL Programs. Some of these checks verify the proper operation of the ACE+ to IDEAL compiler and should be disabled after being performed once. They can be considered as parts of an automated proof of correctness of that compiler. We have been quite conservative in formulating these tests. They might look redundant from the viewpoint that IDEAL code is generated from drawings rather than written by humans. This generation process, however, is not formally specified and may have slipped in errors.

The other tests are static semantic tests every IDEAL Program should undergo. The most efficient architecture might be indeed to have a separate LSC and Route static semantics checker. Other IDEAL compilers then need not to do these checks all over again. Since we are not domain experts, it is highly probable that we only touched the surface and have overlooked many static checks. The tests we have specified using ASF+SDF are marked with (∗)and we supply a reference to the function implementing the test as well as a reference to the appropriate section in the Appendices. Tests not marked with (∗)could have been implemented using ASF+SDF, but we did not do so yet. For this study we have restricted our list to cover only LSC Programs. At NS, some of the checks listed have already been implemented in LSCDraw and the IDEAL Compiler. We do not have an exhaustive

[8] The third window would also have contained a focus should we have double clicked into it with our pointing device

list which checks have already been implemented by NS. We only mention them here for the sake of completeness.

According to our point of view, static semantics of a language is part of a language definition. In COBOL, for instance, it occurs in the so-called *programming rules*. We suggest to use the list below as a starting point for a list of programming rules. Once completed, this list should become part of the IDEAL definition.

1. (∗)**Numbers** are interpreted as Booleans, Integers or Labels depending on their context (cf. Appendix M).

2. **Telegram Names** should begin with a letter and end with two digits. The letter classifies the telegram type as defined in a predefined table for standardization. When a telegram is associated with a telegram type in a program, this telegram type has to conform with the telegram class associated with the letter in the table for standardization. E.g., defining telegram A10 to be of type MMI is incorrect by definition, since the letter 'A' must encode a horizontal telegram (at least according to EURIS [6]).

3. (∗)**Parameters** may not be assigned a value (cf. `check-lhs` Appendix I). (∗)They must be uniquely declared in the parameter block (cf. `double-check-par-declarations` Appendix S).

4. (∗)**Variables** must be uniquely declared in the variable block (cf. `double-check-var-declarations` Appendix W).

5. (∗)**Central Parameter List** must contain unique identifiers (cf. `double-check-par-declarations` Appendix S).

6. (∗)**Telegram Side Files** must contain unique telegram names. All telegrams must have their type declared (cf. `check-telegram-unicity` Appendix V).

7. (∗)**Telegram Data Side Files** must contain unique telegram data names. (∗)All telegram data names must have their range declared (cf. `check-tel-dat-range` Appendix V).

8. (∗)**Parameter Side Files and Variable Side Files** must contain unique identifiers (cf. `double-check-par-declarations` Appendix S and `double-check-var-declarations` Appendix W). All parameters and variables must have their type and/or range declared.

9. (∗)**Directions** must contain either one, two, three, or four unique directions (cf. `check-directions` Appendix C and `check-qualifiers` Appendix H).

10. (∗)**Directions as Qualifiers.** Parameters or variables that are of type BOOL_LIST, INT_TYPE, or INT_CNT_TYPE, can have at most one associated direction (cf. `check-qualifiers` Appendix H).

11. (∗)**IDEAL Programs** must contain at least one Flow block (cf. `check-presence-of-flow-block` Appendix B).

12. (∗)**Function Blocks** may call other functions but not themselves, i.e., no recursion or mutual recursion (cf. `check-lables` Appendix G). (∗)Function blocks and merge blocks that lie within the scope of a function may not contain the set, send, and central send statement (cf. `csi-function-blocks` and `csi-function-merge-blocks` Appendix B). Functions must contain a set statement, i.e., functions must return a value.

13. (∗)**Merge Blocks** must have a unique label(cf. `check-lables` Appendix G). (∗)There must be at least one continue-at label referring to a merge block, otherwise this piece of code is unreachable (cf. `check-lables` Appendix G).

14. (∗)**Normal Flow Blocks** must be initiated by a telegram-in-type of type normal (either horizontal or vertical) or a telegram initiating identifier of non central type (cf. `check-normal-flow` Appendix I). (∗)Direction lists associated to a telegram initiator, must be valid, i.e., they must obey the static semantical requirements stated for direction lists above.

15. (∗)**Central Flow Blocks** must be initiated by a telegram-in-type of central type or a telegram initiating identifier of central type (cf. `check-central-flow` Appendix I). (∗)The central parameter list must be declared in the variable block. Direction lists associated with a telegram initiator must be valid.

16. (∗)**Set Statements** must contain a valid set expression (cf. `tc-set-expres sion` Appendix I).

17. **Split Statements**

    - Consecutive splits—we recognize them as *split clusters*—may not have overlapping test expressions, in other words: for any value of the expression tested only one split may yield true. E.g.,

      ```
      lsc_function NPU;
        lsc_split C >= 0;
          lsc_cont_at label_1;
        lsc_split_end;
        lsc_split C <= 2;
          lsc_cont_at label_2;
        lsc_split_end;
      lsc_function_end;
      ```

      is syntactically correct, but is static semantically incorrect, since the test conditions are not mutually exclusive. As a consequence, split clusters are *bound* by *one* par-var-identifier or telegram-data-name.

    - Splits occur at least in couples. E.g.,

      ```
      lsc_function NPU;
        lsc_split F = 0;
          lsc_cont_at label_1;
        lsc_split_end;
      lsc_function_end;
      ```

      is syntactically correct, but is static semantically incorrect, since when there is only one choice, `lsc_test` should be used.

    A very strong constraint actually holds: within a block (either function, merge, flow or 'split block' (see Section 7), there can be at the top level at most one *cluster of splits*. Such a split cluster may be nested, but cannot be followed by any statement.

18. (∗)**Continue at label Statements** must refer to an existing merge label within its scope, otherwise it is a jump to nowhere; i.e., a Cont_at statement in a function must refer to a Merge Label in that same function block and a continue at outside a function may never refer to a Merge Label inside a function (cf. `check-labels` Appendix G). Cont_at is always the last statement of the smallest block it is contained in. E.g.:

```
lsc_split F = 0;
  lsc_cont_at label_1;
  lsc_set C : 1;
lsc_split_end;
```

is syntactically correct, but is static semantically incorrect; `lsc_cont_at` is semantically equivalent to a 'goto' statement, so any statement following a Cont_at is *unreachable code.* (cf. `check-labels` Appendix G).

19. (∗)**Test Statements** must contain a valid test expression (cf. `tc-test-expres sion` Appendix D).

20. (∗)**Send Statements** must contain a telegram of type normal, either horizontal or vertical, that is predefined as such in the telegram side file (cf. `check-sends` Appendix E).

21. (∗)**Central Send Statements** must contain a telegram of type central that is predefined in the telegram side file. (∗)For both send statements hold that they always have to be the last statement of the smallest block it is contained in (cf. `check-sends` Appendix E). E.g.

```
lsc_merge label_1;
  lsc_send P01;    -- or lsc_cnt_send
  lsc_set C : 0;
lsc_merge_end;
```

is syntactically correct, but is static semantically incorrect.

22. (∗)**Arithmetic Operators**, including the modulus operator, accept only expressions of integer type. (cf. `tc-simple-expression` Appendix D).

23. (∗)**Relational Operators** accept only expressions of integer type. (cf. `tc-test-expression` Appendix D).

24. (∗)**The Equivalence Operator** accepts expressions of all types except list types. (cf. `tc-test-expression` Appendix D).

25. (∗)**A List Access** must be indexed with an expression of type integer and has to access an identifier of either boolean list type, or integer list type (cf. `tc-list-access` Appendix D).

26. (∗)**Central Variable Operators** {`*`}and {`#`} may only be applied to variables of type central. (cf. `tc-simple-expression` Appendix D).

27. (∗)**Set Expressions** may not assign a value to a parameter (cf. `check-lhs` Appendix I). (∗)The left hand side type must be type assign-compatible with the right hand side (cf. `check-lhs` Appendix I). Functions may be 'assigned' a value, i.e., like in Pascal return values of functions are denoted with an expression that resembles an assignment. Recursion or mutual recursion are, however, forbidden.

28. (∗)**Par Var Expressions** must contain a variable if they also contain a var-attribute or a central variable operator (cf. `tc-var-attr` Appendix D). (∗)If the var-attribute involves directions, they must be valid and declared as such.

# 7 Some Recommendations for Improving the Syntax of IDEAL

We close our discussion with some remarks on the IDEAL Syntax.

We are aware that changing the IDEAL syntax is easier said than done for both economical and political reasons. Our suggestions should therefore rather be taken as reminders for future work on IDEAL or the ISL specification language.

We omit cosmetic criteria. IDEAL, as we understood it, is primarily a tool interface language. Our recommendations are meant to improve the IDEAL languages readability for *Computers.*

The line between syntax and static semantics is not as sharp as one would like. All syntax checks can be performed static semantically. The opposite is not true in general. The main advantage of solving things in syntax is that it is fast and often comes for free. The greatest advantage to put things in static semantic checks is, that it unburdens the syntax definition with all kind of 'exceptions'. As a result, it makes the syntax more transparent and the user can get a decent error report when (s)he makes a mistake.

## BNF

The IDEAL Guide and Reference should at least contain an Appendix with a formal BNF (or extended BNF) definition of the IDEAL syntax. The current document leaves too much room for interpretation.

## Side Files

Although side files are not considered to be part of the IDEAL Language it would be very helpful to have their syntax included in the Appendix of the Guide and Reference.

## Identifiers

It is not advisable to include naming conventions in a language. Although for the sake of standardization naming conventions are applaudable, when these are not enforced to be mutually exclusive, they introduce ambiguities for the parser. To illustrate this, consider the following IDEAL expression.

```
lsc_flow A03:@;
lsc_flow_end;
```

Obeying the syntax, which is:

```
<flow-block> ::= lsc_flow <telegram-name> : <telegram-in-type>;
                   <flow-statement>*
                 lsc_flow_end;
               |
                 lsc_flow <par-var-identifier> : <telegram-in-type>;
                   <flow-statement>*
                 lsc_flow_end;.

<telegram-name>      ::= <letter> <digit> <digit>
<par-var-identifier> ::= <letter> <letter-or-digit> <letter-or-digit>+
```

A03, stands for both the telegram and for its associated telegram initiating variable. This leaves the parser at a loss whether it sees a telegram or a variable.

Our advice is to enforce the naming convention of telegrams at the static semantic level by a function `check-telegram-name(Identifier)` and to abandon the distinction of telegram name and par-var-identifier at the syntactical level.

The definition `<telegram-data-name>`, however, is correctly put in the syntax, since it does not clash with the other identifier definitions which consist of more letters.

## Booleans vs. Ranges

Although we know that the IDEAL language designers have tried to make a one-to-one correspondence between LSCs and IDEAL, we consider using "0" and "1" to represent the Boolean constructors at the lexical level to be undesirable, since it introduces lexical ambiguities with the Integer representations of zero and one. Moreover, IDEAL is not supposed to be read by humans, therefore the argument that IDEAL has to be recognizable as a representation of LSCs is less compelling.

There are two options. One can either take "0" and "1" to be Integers lexically and interpret them as boolean values in certain kind of expressions like in C. Or one can regard them as truly Boolean constructors with their own lexical tokens associated with them like `true` and `false` as in Pascal. We prefer the first option to keep the symmetry of the data types. This option also allows us to drop the notion of Booleans altogether and move toward Ranges of Integers as the generalization of all IDEAL Types.

## Blocks and Statements

The distinction between merge-blocks and function-merge-blocks, or function-statements and flow-statements cannot be made at the syntactical level. Perhaps this is only suggested by [31] and implemented differently. We want to stress this point, however, to prevent future misunderstandings (or misinterpretations of [31]). E.g.,

```
lsc_split NPU=0;
  lsc_set C:1;
lsc_split_end;
```

obeying the following syntax (simplified for this example):

```
<function-statement> :: lsc_set <set-expression>;.
<flow-statement>     :: <function-statement>.

<function-statement> ::= lsc_split <test-expression>;
                           <function-statement>*
                         lsc_split_end;.
<flow-statement>     ::= lsc_split <test-expression>;
                           <flow-statement>*
                         lsc_split_end;.
```

leaves the parser at a loss whether `lsc_set` is a flow-statement or a function-statement. An analogous argument applies to function-merge-blocks. We suggest the test for restricted usage of the `test, send, cnt send` statements outside the function-block scope to be performed at the static semantic level and to regard all IDEAL statements to be simply `<statements>` at the syntactical level.

## Expressions

To remain compatible with the varyadic nature of IDEAL relational operators we should have used the syntax for Expressions below.

```
<simple-expression> ::= Identifier
                      | Number
                      | Id '['<simple-expression>']'
                      .   %% all the par-var-exp in IDEAL
                      .
                      | - <simple-expression>
                      |  '|' <simple-expression> '|'
                      |  <simple-expression> +   <simple-expression>
                      |  <simple-expression> -   <simple-expression>
                      |  <simple-expression> *   <simple-expression>
                      |  ( <simple-expression> )
                      .
<test-expression>   ::=   <simple-expression> <   <simple-expression>
                      | <simple-expression> >   <simple-expression>
                      | <simple-expression> <=  <simple-expression>
                      | <simple-expression> >=  <simple-expression>
                      | <simple-expression> <>  <simple-expression>
                      | <simple-expression> =   <simple-expression>
                           %% allows 1=1=1, in C: (1==1==1)

                      | <simple-expression> <   <test-expression>
                      | <simple-expression> >   <test-expression>
                      | <simple-expression> <=  <test-expression>
                      | <simple-expression> >=  <test-expression>
                      | <simple-expression> <>  <test-expression>
                      | <simple-expression> =   <test-expression>
                      .
```

Unfortunately, we would run into problems with this syntax when we want to do type evaluation. `Exp < Exp` is expected to return a boolean. This causes the type evaluation of an expression like `1 < 2 < 3` to be problematic. The point here is that defining a proper type evaluation is only possible when the *operational* semantics of such expressions is known. I.e., only if an expression like `1 < 2 < 3` is operationally equivalent to `1 < (2 < 3)`, then we know that an integer value is compared to a boolean value. Note that this may still be allowed since boolean values in IDEAL have an integer range. If comparison of a boolean and an integer value is not allowed, we have caught an error. Conversely, if an expression like `1 < 2 < 3` is operationally equivalent to `(1 < 2) and (2 < 3)`, the expression is properly typed.

Since we have interpreted the IDEAL relational operators to be of binary nature, this would mean a major rewrite for our typechecker, because there is no `and` construction in IDEAL. Straightforward type evaluation therefore fails. On the other hand rewriting our typechecker might not be necessary, because of the following important observation — quoting [23]:

> The tool (Ace+) will only recognize that the actions belong to a VAR_PAR symbol when the placement of the actions is *exactly* under the VAR_PAR symbol.

This suggests there is only a limited space for expressions below a par-var-identifier. More importantly, from a certain position in an LSC, only the par-var-identifier exactly above it can be referred to. This again means that at most one par-var-identifier is involved in an expression. The other observation we make is that the basic IDEAL constructs seem to be expressive enough to provide alternative way to formulate more complicated expressions. E.g., the expression in LSC 6.1a /NS (E) [6]:

```
    [>>]        [>>]
    MSW2         MSW1

--( MSW2 >= VW > MSW1 )----
```

can be rephrased by:

```
      [>>]          [>>]
      MSW2           MSW1

   --(MSW2>=VW)---(VW>MSW1)----
```

In the programming language Pascal, relational operators are binary and non-associative. Range checks are typically done with boolean expressions: `( 1 < x ) and ( x < 3 )` . Even a range check on a variable can be decomposed in an LSC drawing. E.g.,

```
             []
             XYZ

     ------( XYZ > 40 )--
                        |
         __( XYZ < 300 )-
         |
         -------------------
```

In C the situation is different. The important point in C is, however, that unlike in IDEAL, one may use 'boolean' expressions in integer expressions. That is, there is no 'boolean' type at all, instead, integer values are used to represent Boolean values as, for instance, in:

```
int foo(int a, int b)
{
  printf("result = %d\n",  ( a == 3 ) * b );
}
```

**CONJECTURE 1** *The following three facts were derived from LSCs*

1. *the limited space on a physical sheet of paper below a par-var-identifier*
2. *the binding of an expression to at most one par-var-identifier and*
3. *the option to rephrase test-expressions into a flow with splits and merges*

*This suggests that there might be an unnecessary degree of expressiveness in IDEAL test-expressions as is currently defined.*

We do not claim, however, that we fully understand all the implications of this conjecture. We have spent so much attention on the IDEAL relational operators, since this is the major point where we have decided not to follow the current IDEAL definition.

## Split Blocks and Split Clusters

Closely related to the previous paragraph is our understanding of the existence of *Split Blocks* and *Split Clusters* (not defined in IDEAL). We refer to Section 6 for the motivation behind split blocks. We introduce, on the fly, the notions of *Flow Initiators* and *Flow Terminators* for the sake of symmetry. Putting every constraint in syntax as much as possible (which does not necessarily means to be the best solution) the next modified IDEAL syntax for split blocks, split clusters and their scope might look like [9]:

[9]Note that a `<flow-terminator>` may be an empty statement.

```
<set-or-test-statement> ::= <test-statement>
                          | <set-statement>.
<flow-terminator>       ::=   <send-statement>
                          | <continue-at-statement>
                          | <split-cluster>.

<split-block>           ::= <test-expression>
                              <set-or-test-statement>*
                              [<flow-terminator>].
<split-blocks>          ::= <split-block> <split-block>+.

<split-cluster>         ::=  lsc_split par-var-exp;
                               <split-blocks>
                             lsc_split_end;
                             |
                             lsc_split telegram-data-name;
                               <split-blocks>
                             lsc_split_end;
                             .

<flow-block>            ::= lsc_flow <flow-initiator>;
                              <set-or-test-statement>*
                              [<flow-terminator>]
                            lsc_flow_end;
                            .
```

with the associated static check that all test-expressions in the split-blocks within the split statement must be bound by the par-var-exp or the telegram-data-name immediately preceding the split. E.g.,

```
lsc_function NPU;
  lsc_split F;
    F = 1 lsc_split SNR;
            SNR <> NS lsc_cont_at label_1;
            SNR =  NS lsc_cont_at label_2;
          lsc_split_end;
    F = 0 lsc_cont_at label_2;
  lsc_split_end;

  lsc_merge label_1;
    lsc_set NPU:1;
  lsc_merge_end;
```

etc. The adaptation of the syntax of the other block constructs, i.e., function blocks, merge blocks, and split blocks, should mimic the new flow block construct. Using this syntax makes the inner workings of an LSC nearly as transparent as the I/O Automaton model (cf. [21]).

**CONJECTURE 2** *It is possible that the presently defined syntax and static semantics related to the split statement in IDEAL LSC Programs is not strict enough to enforce its intended static semantical correctness.*

We hurry again to state that we do not claim to understand all implications of this conjecture.

# 8 Related Work

We discuss work related to ours in the context of our final goal: the verification of ISL programs. This might seem rather preliminary, but we feel that it should be mentioned here for the sake of completeness and, as stated in Section 1, it serves our aim to give an overview of the ISL project as well.

The literature in which railway yards are discussed can roughly be divided into two categories. One in which railway yards are used as a good illustration of general techniques and ideas, and one in which is focussed specifically on railway related questions.

Examples in the first category are [15], [9], [29], [14] and [28]. These books and papers are mentioned because they highlight a number of important aspects, e.g., specification, correctness criteria, the relation between criteria and implementation, and scheduling optimization. In [15] trains sets are studied; a complete specification in COLD-1 is presented of a class of railway yards, including trains, rudimentary user interface and interlocking philosophy. Broy also gives a full presentation, but a much simpler system with extra attention to verification of some invariants [9]. In [29] a general framework for the requirements analysis of safety-critical systems is presented. A distinction is made between *safety requirements* and *mission requirements*. It turns out that the requirements presented in this paper are mostly mission requirements. These specify the situations that should be avoided, whereas safety requirements specify what should happen when something goes wrong after all. M. Feather discusses in [14] how to derive an implementation from a specification, this in contrast to "verification of an implementation after some unknown invention process". In [28] a solution is presented for the Merlin-Randell problem of train journeys.

In the other category of the literature is reported on research into specific systems being designed or used by railway companies and railway suppliers. Here it makes sense to further categorize into *elementary unit* systems and *free wired* systems. This distinction is presented in [39] where it is argued that extrapolation of the two usual approaches, elementary unit and free wired, do not lead to systems which fully exploit the possibilities offered by programmable electronics; a full treatment of interlocking specification and design is given for implementation by means of programmable electronics.

In elementary unit systems, every element of the track, e.g., signal and points, is an autonomous device, which communicates with neighboring elements; layout of the railway yard is reflected in the structure of the interlocking. In free wired case, larger chucks of a railway yard are controlled more or less centrally, the layout of the railway yard is less obvious in the structure. ISL typically follows the elementary unit approach.

To begin with the elementary unit systems, P. Middelraad of NS initiated the development of the ISL language for the specification of these systems. This language is currently is use with NS and also received internationally attention [6, 7]. Besides this paper further study of the language is taking place in an academic setting [5, 34]. Siemens is an important supplier of elementary unit based interlockings, among other things, work is done on verification by means of process algebra [16].

Free wired systems are also used by NS. In this case *VPI* (Vital Processor Interlocking)[10] is used. Verification of *VPI* programs, based on satisfiability checking of propositional logic formulas, e.g. by using an improved BDD based theorem prover [11, 17], is discussed in [18]. The free wired system used in the U.K. by British Rail known as the Solid State Interlocking (*SSI*). *SSIs* received considerable attention in the literature [2, 10, 12, 21, 32, 33]. The *SSI* is a more complex machine than the *VPI*, one reason for this is that the *SSI* is used to take more complex scheduling decisions than the *VPI*. Here, the HOL theorem prover is used to

[10] *VPI* and Vital Processor Interlocking are registered trademarks of the General Railway Signal Company.

verify safety properties [38]. Some Swedish interlockings seem to have been verified using tautology checking in a way that seems similar to what is presented in [18], see [37]. Finally, in France a consortium of GEC Alsthom, Matra Transport and CSEE executed a number of projects in the railway sector where formal specification and formal verification were applied [13].

# 9 Conclusions and Further Research

The need for an interface language for the (EURIS)-Simulation Project is beyond discussion. The tool interface language IDEAL seems to fulfill its primary goals, which is supporting for automated syntax and static semantical checks of drawings, validation and simulation. However, the language definition of IDEAL can be strengthened. See the section on static semantical correctness and recommendations for improvement of IDEAL syntax in Sections 6, 7. In conclusion, we suggest to strive for a strong typed language, by simplifying the syntax of IDEAL and by completing the definition and implementation of static semantical checks on IDEAL.

Should the NS ever consider to use a symbolic LSC/Route specification language intended to be written and read by humans, we advice them to start from scratch and take the suggestions listed above into account.

At the time of this writing, it has already been decided to follow the path of Real Time Colored Petri-Nets and to study the feasibility of a machine translation of IDEAL into the ExSpect simulation language [5, 34]. This is a rather practical step. Formal approaches that can be considered are the careful study of semantics in Process algebra, as sketched in Section 3.3, and/or Petri-Net, and the study of verification strategies. The way ISL is structured seems to call for *compositional correctness*, that is, proof correctness of elements and of combinational operators on elements.

# References


[1] W.M.P. van der Aalst and M.A. Odijk. Analysis of railway stations by means of interval timed coloured Petri nets. Technical report, Eindhoven Technical University, 1993. To appear in *Real-Time systems*, Kluwer Academic Publishers, Boston, 1995.

[2] S. Anderson and G. Cleland. Formal approaches to safety in programmable electronic systems. Technical report, University of Edinburgh, LFCS, 1993.

[3] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Journal of Formal Aspects of Computing Science*, 3(2):142–188, 1991.

[4] J.C.M. Baeten and W.P. Weijland. *Process algebra, Cambridge Tracts in Theoretical Computer Science*, volume 18. Cambridge University Press, 1990.

[5] A.A. Basten, R.N. Bol, and M. Voorhoeve. Simulating and Analyzing Railway Interlockings in Exspect. Computing Science Report 94/37, Eindhoven University of Technology, Department of Mathematics and Computing Science, Eindhoven, The Netherlands, September 1994.

[6] J. Berger, P. Middelraad, and A.J. Smith. EURIS, European Railway Interlocking Specification. Technical report, UIC, Committee 7A/16, 1992.

[7] J. Berger, P. Middelraad, and A.J. Smith. The European Railway Interlocking Specification. In *Proceedings*. The Institution of Railway Signal Engineers, January 1993.

[8] J.A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic Specification.* ACM Press Frontier Series. The ACM Press in co-operation with Addison-Wesley, 1989.

[9] M. Broy. Specification of a railway system. Technical Report MIP-8715, Universität Passau, 1987.

[10] G. Bruns. A case study in safety-critical design. In G. von Bochmann and D.K. Probst, editors, *Computer Aided Verification*, volume 663 of *LNCS*, pages 220–233, 1992.

[11] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[12] G.V. Conroy and C. Pulley. Logical methods in the formal verification of safety-critical software. Technical report, UMIST, 1993.

[13] D. Craigen, S. Gerhart, and T. Ralston. *An International Survey of Industrial Applications of Formal Methods–Case Studies*, volume 2. NIST (National Institute of Standards and Technology), 1993.

[14] M.S. Feather. Towards a Derivational Style of Distributed System Design - An Example. *Automated Software Engineering*, 1(1), 1994.

[15] L.M.G. Feijs, H.B.M. Jonkers, and C.A. Middelburg. *Notations for Software Design.* FACIT Series. Springer Verlag, 1994. To appear in October 1994.

[16] S. Fischer, A. Scholz, and D. Taubner. Verification in process algebra of the distributed control of track vehicles – a case study. In G. von Bochmann and D.K. Probst, editors, *Computer Aided Verification*, volume 663 of *LNCS*, pages 192–205, 1992.

[17] J.F. Groote. Hiding propositional constants in BDDs. Technical Report 120, Logic Group Preprint Series, Utrecht University, 1994.

[18] J.F. Groote, J.W.C. Koorn, and S.F.M. van Vlijmen. The safety guaranteeing system at station Hoorn-Kersenboogerd. Technical Report 121, Logic Group Preprint Series, Utrecht University, 1994.

[19] J.F. Groote and A. Ponse. $\mu$CRL: A base for analysing processes with data. In E. Best and G. Rozenberg, editors, *Proceedings of the third Workshop on Concurrency and Compositionality*, volume 191 of *GMD-Studien*, pages 125–130, 1991.

[20] J. Heering and P. Klint. The syntax definition formalism SDF. In J.A. Bergstra, J. Heering, and P. Klint, editors, *Algebraic Specification*, ACM Press Frontier Series. The ACM Press in co-operation with Addison-Wesley, 1989.

[21] M. Ingleby and I. Mitchell. Proving safety of a railway signalling system incorporating geographic data. In Heinz H. Frey, editor, *Proceedings of SAFECOMP '92.* Pergamon Press, 1992.

[22] S.C. Johnson. Yacc—yet another compiler compiler. Technical Report Computing Science Technical Report 32, AT&T Bell Laboratories, Murray Hill, N.J., 1975.

[23] H.A. Klap. Euris-Simulation Tutorial. Technical report, Ingenieursbureau Nederlandse Spoorwegen, 1994. Preliminary version 0.3.

[24] P. Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology*, 2(2):176–201, 1993.

[25] J.W.C. Koorn. Connecting semantic tools to a syntax-directed user-interface. Report P9222, Programming Research Group, University of Amsterdam, 1992. Available by *ftp* from ftp.cwi.nl:/pub/gipe as Koo92a.ps.Z.

[26] J.W.C. Koorn. GSE: A generic text and structure editor. In J.L.G. Dietz, editor, *Conference Proceedings of Computing Science in the Netherlands, CSN'92*, pages 168–177. SION, 1992. Also appeared as Report P9202, University of Amsterdam.

[27] J.W.C. Koorn. *Generating uniform user-interfaces for interactive programming environments*. PhD thesis, University of Amsterdam, 1994. ILLC Dissertation series 1994-2.

[28] M. Koutny. The Merlin-Randell problem of train journeys. *Acta Informatica*, 23:429–463, 1986.

[29] R. de Lemos, A. Saeed, and T. Anderson. A train set as a case study for the requirements analysis of safety-critical systems. *The Computer Journal*, 35(1):30–40, 1992.

[30] M.E. Lesk. Lex—a lexical analyzer generator. Technical Report Computing Science Technical Report 39, AT&T Bell Laboratories, Murray Hill, N.J., 1975.

[31] F. Makkinga. IDEAL, Interlocking Design and Application Language Guide and Reference. Technical report, Ingenieursbureau Nederlandse Spoorwegen, 1994. Preliminary version 0.3.

[32] M.J. Morley. Modelling British Rail's Interlocking logic: Geographic data correctness. Technical Report ECS-LFCS-91-186, University of Edinburgh, LFCS, November 1991.

[33] M.J. Morley. Safety in railway signalling data: A behavioural analysis. In J. Joyce and C. Seger, editors, *Higher Order Logic Theorem Proving and its Applications*, volume 780 of *LNCS*. Springer Verlag, 1993.

[34] L.H. Oei. Pruning the search tree of interlocking design and application language operational semantics. Report P9418, Programming Research Group, University of Amsterdam, 1994.

[35] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[36] C. Pulley. Personal communications.

[37] G. Stålmark and M. Säflund. Modelling and verifying systems and software in propositional logic. In *Proceedings of SAFECOMP '90*, pages 31–36. Pergamon Press, 1990.

[38] Wai Wong. *A Formal Theory of Railway Track Networks in Higher-order Logic an its Applications in Interlocking Design*. PhD thesis, University of Warwick, Dept. of Engineering, 1991.

[39] L. Zigterman. *A new approach to specification and design of a railway-interlocking: the 'ROUTE' approach*. PhD thesis, Technische Hogeschool Delft, 1984.

# Syntax and static semantics of IDEAL in ASF+SDF

We will not discuss the specification in its entirety for brevity's sake. We confine ourselves to an introduction. Recall that an introduction to the ASF+SDF formalism can be found in Section 5.

Checking the static semantics amounts to defining a mapping from IDEAL programs to a list of error messages. To produce the complete list of errors found in an IDEAL program, one applies the function `check-all` to that program as defined in module `TC-LSC` (cf. Appendix U), which is the "top level" module of the specification. This function in its turn invokes a number of functions that check a particular constraint, such as checking labels, checking directions, etc. These auxiliary functions are defined in separate modules. For instance, the function that checks labels is called `check-labels` which is defined in module `Check-labels` (cf. Appendix G). Results of these auxiliary functions are concatenated to a list by using the function `join-errors` defined in module `Errors` (cf. Appendix J). The reader interested in a particular constraint is referred to Section 6 to obtain a pointer to the module and function implementing that constraint.

## A Booleans

**Standard Booleans** The specification of Booleans is explained in Section 5. It should be noted that this module defines the syntax and semantics of the 'Standard Booleans', *not* the BOOL_TYPE's of IDEAL which are defined in the module Ideal-Types. All equations within this specification use the Standard Booleans rather than the IDEAL 'Booleans'. The reason for this is that the IDEAL language does not have the standard Boolean operators. and these functions will be used in functions defined for the static semantics.

**imports** Layout$^{(R)}$
**exports**
  **sorts** BOOL
  **context-free syntax**
    *true* → BOOL
    *false* → BOOL
    BOOL "*or*" BOOL → BOOL {**left**}
    BOOL "*and*" BOOL → BOOL {**left**}
    *not*(BOOL) → BOOL
    "(" BOOL ")" → BOOL {**bracket**}
  **variables**
    *Bool* [*0-9′*]*→ BOOL
  **priorities**
    "*or*" < "*and*"

**equations**

[1] $true\ or\ Bool\ =\ true$

[2] $false\ or\ Bool\ =\ Bool$

[3] $true\ and\ Bool\ =\ Bool$

[4] $false\ and\ Bool\ =\ false$

[5] $not(false)\ =\ true$

[6] $not(true)\ =\ false$

# B Check-LSC-structure

**csi**-functions checks statements in LSC Program constructs. (**csi** is an abbreviation of Check Statements In). These checks allow us to abandon the distinction between Function-statements and Flow-statements, and between Function-merg-blocks and Merge-blocks. Now the scoping of 'Flow-statements' are checked static semantically. See recommendations (cf. Section 7) for further motivation.

**imports** LSC-mappings$^{(Q)}$ Errors$^{(J)}$
**exports**
**context-free syntax**

| | |
|---|---|
| *check-lsc-struct*(Ideal-program) | → ERROR-LIST |
| *check-presence-of-flow-block*(Ideal-program) | → ERROR-LIST |
| *csi-function-blocks*(Function-blocks) | → ERROR-LIST |
| *csi-function-merge-blocks*(Merge-blocks, Identifier) | → ERROR-LIST |
| *csi-function-statements*(Statements, Identifier) | → ERROR-LIST |
| *csi-function-statement*(Statement, Identifier) | → ERROR-LIST |
| *is-function-statement*(Statement) | → BOOL |
| *missing* "*Flow-block␣in␣Ideal-program*" Program-start-block | → ERROR |
| *illegal statement* Statement *within* "*Function-block*" Identifier "*SET␣SPLIT␣or␣CONTINUE␣AT*" expected | → ERROR |

**equations**

[1] *check-lsc-struct*(Ideal-program$'$) =
*join-errors*(*check-presence-of-flow-block*(Ideal-program$'$),
*csi-function-blocks*(*function-blocks*(Ideal-program$'$)))

[2]
*flow-blocks*(Ideal-program$'$) =
―――――――――――――――――――――――――――――――――
*check-presence-of-flow-block*(Ideal-program$'$) =
*missing Flow-block␣in␣Ideal-program program-start-block*(Ideal-program$'$)

[3] *check-presence-of-flow-block*(Ideal-program$'$) = **otherwise**

[4] *csi-function-blocks*() =

[5] *csi-function-blocks*(Function-block$'$ Function-blocks$'$) =
*join-errors*(*csi-function-statements*(*function-statements*(Function-block$'$),
*name*(Function-block$'$)),
*join-errors*(*csi-function-merge-blocks*(*function-merge-blocks*(Function-block$'$),
*name*(Function-block$'$)),
*csi-function-blocks*(Function-blocks$'$)))

[6] *csi-function-merge-blocks*(, Identifier$'$) =

[7] $$\frac{}{\begin{array}{l}\textit{csi-function-merge-blocks}(\text{Merge-block}'\ \text{Merge-blocks}',\ \text{Identifier}') = \\ \quad \textit{join-errors}(\textit{csi-function-statements}(\textit{merge-statements}(\text{Merge-block}'), \text{Identifier}'), \\ \qquad \textit{csi-function-merge-blocks}(\text{Merge-blocks}', \text{Identifier}'))\end{array}}$$

[8] $\textit{csi-function-statements}(, \text{Identifier}') =$

[9] $$\begin{array}{l}\textit{csi-function-statements}(\text{Statement}'\ \text{Statements}',\ \text{Identifier}') = \\ \quad \textit{join-errors}(\textit{csi-function-statement}(\text{Statement}', \text{Identifier}'), \\ \qquad \textit{csi-function-statements}(\text{Statements}', \text{Identifier}'))\end{array}$$

[10] $$\frac{\textit{is-function-statement}(\text{Statement}') = \textit{false}}{\begin{array}{l}\textit{csi-function-statement}(\text{Statement}', \text{Identifier}') = \\ \textit{illegal statement } \text{Statement}' \\ \textit{within Function-block } \text{Identifier}'\ \textit{SET}_\sqcup\textit{SPLIT}_\sqcup\textit{or}_\sqcup\textit{CONTINUE}_\sqcup\textit{AT expected}\end{array}}$$

[11] $$\begin{array}{l}\textit{csi-function-statement}(\textit{LSC\_SPLIT}\ \text{Test-expression}'; \\ \qquad \text{Statements}'\ \textit{LSC\_SPLIT\_END};\ , \text{Identifier}') = \\ \quad \textit{csi-function-statements}(\text{Statements}', \text{Identifier}')\end{array}$$

[12] $\textit{csi-function-statement}(\text{Statement}', \text{Identifier}') =$ **otherwise**

[13] $\textit{is-function-statement}(\textit{LSC\_SET}\ \text{Set-expression}';) = \textit{true}$

[14] $\textit{is-function-statement}(\textit{LSC\_TEST}\ \text{Test-expression}';) = \textit{false}$

[15] $\textit{is-function-statement}(\textit{LSC\_SPLIT}\ \text{Test-expression}';\ \text{Statements}'\ \textit{LSC\_SPLIT\_END};) = \textit{true}$

[16] $\textit{is-function-statement}(\textit{LSC\_SEND}\ \text{Identifier}'\ :\ \text{Telegram-output-type}';) = \textit{false}$

[17] $\textit{is-function-statement}(\textit{LSC\_CNT\_SEND}\ \text{Identifier}'\ :\ \text{Telegram-output-type}'\ \text{Central-parameter-list}';) = \textit{false}$

[18] $\textit{is-function-statement}(\textit{LSC\_CONT\_AT LABEL\_}\ \text{Label}';) = \textit{true}$

# C Check-directions

**check-directions** makes sure that directions used in set statements must be declared, unique and never exceed a total of four. The difference with some of the direction checks in the module Check-qualifiers is that in that module directions in declarations and test expressions are checked. The scattering of directions checks over several modules has more to do with project management between more than one individual rather than for technical reasons.

**imports** Check-qualifiers$^{(H)}$
**exports**
**context-free syntax**

| | |
|---|---|
| *check-directions*(Ideal-program) | → ERROR-LIST |
| | |
| *check-fubs-directions*(TENV, Function-blocks) | → ERROR-LIST |
| *check-fub-directions*(TENV, Function-block) | → ERROR-LIST |
| *check-mbs-directions*(TENV, Merge-blocks) | → ERROR-LIST |
| *check-mb-directions*(TENV, Merge-block) | → ERROR-LIST |
| *check-flbs-directions*(TENV, Flow-blocks) | → ERROR-LIST |
| *check-flb-directions*(TENV, Flow-block) | → ERROR-LIST |
| | |
| *check-dir*(TENV, Statements, Identifier) | → ERROR-LIST |
| *check-dir*(TENV, Statements, Label) | → ERROR-LIST |
| *check-dir*(TENV, Statements, Flow-initiator) | → ERROR-LIST |
| | |
| *get-set-statements-from-stmts*(Statements) | → Statements |
| *get-set-statements-from-fmbs*(Merge-blocks) | → Statements |
| *get-fumb-set-statements*(Merge-block) | → Statements |
| *get-par-var-exp-set-stmts*(Statements) | → Statements |
| | |
| *check-dir-in-par-var-exp*(TENV, Statements, Identifier) | → ERROR-LIST |
| *check-dir-in-par-var-exp*(TENV, Statements, Label) | → ERROR-LIST |
| *check-dir-in-par-var-exp*(TENV, Statements, Flow-initiator) | → ERROR-LIST |
| | |
| *join-stmts*(Statements, Statements) | → Statements |
| | |
| *these stmts from function block* Identifier should be checked further Statements TENV | → ERROR-LIST |
| | |
| *function block* Identifier *contains set statement with illegal* "*direction(s)*" Statement | → ERROR |
| *these stmts from merge block* Label should be checked further Statements TENV | → ERROR-LIST |
| | |
| *merge block* Label *contains set statement with* illegal "*direction(s)*" Statement | → ERROR |
| *these stmts from flow block* Flow-initiator should be checked further Statements TENV | → ERROR-LIST |
| | |
| *flow block* Flow-initiator *contains set statement with illegal* "*direction(s)*" Statement | → ERROR |
| | |
| *has-proper-directions*(TENV, Statement) | → BOOL |
| *has-par-var-exp-with-directions*(Set-expression) | → BOOL |
| *has-var-attribute*(Par-var-exp) | → BOOL |
| *has-directions*(Var-attribute) | → BOOL |
| *check-directions*(TENV, Directions, Identifier) | → BOOL |
| *is-set-statement*(Statement) | → BOOL |

*in-directions-declaration*(Directions, Dir-list) $\to$ BOOL
*direction-in-declaration*(Direction, Dir-list) $\to$ BOOL
*in-directions*(Direction, Directions) $\to$ BOOL

*dir-par-var*(Par-var-exp) $\to$ Identifier

**equations**

[1]
$$\frac{\begin{array}{l}[\mathit{IdealTelegramData},\\ \quad \mathit{parameter\text{-}declarations}(\mathit{parameter\text{-}block}(\text{Ideal-program}')),\\ \quad \mathit{variable\text{-}declarations}(\mathit{variable\text{-}block}(\text{Ideal-program}'))] \quad = \mathit{Tenv}'\end{array}}{\begin{array}{l}\mathit{check\text{-}directions}(\text{Ideal-program}') =\\ \mathit{join\text{-}errors}(\mathit{check\text{-}fubs\text{-}directions}(\text{Tenv}', \mathit{function\text{-}blocks}(\text{Ideal-program}')),\\ \qquad \mathit{join\text{-}errors}(\mathit{check\text{-}mbs\text{-}directions}(\text{Tenv}', \mathit{merge\text{-}blocks}(\text{Ideal-program}')),\\ \qquad\qquad \mathit{check\text{-}flbs\text{-}directions}(\text{Tenv}', \mathit{flow\text{-}blocks}(\text{Ideal-program}'))))\end{array}}$$

[2] $\mathit{check\text{-}fubs\text{-}directions}(\text{Tenv}', \text{Function-block}'\ \text{Function-blocks}') =$
$\quad \mathit{join\text{-}errors}(\mathit{check\text{-}fub\text{-}directions}(\text{Tenv}', \text{Function-block}'),$
$\qquad \mathit{check\text{-}fubs\text{-}directions}(\text{Tenv}', \text{Function-blocks}'))$

[3] $\mathit{check\text{-}fubs\text{-}directions}(\text{Tenv}', ) =$

[4] $\mathit{check\text{-}mbs\text{-}directions}(\text{Tenv}', \text{Merge-block}'\ \text{Merge-blocks}') =$
$\quad \mathit{join\text{-}errors}(\mathit{check\text{-}mb\text{-}directions}(\text{Tenv}', \text{Merge-block}'),$
$\qquad \mathit{check\text{-}mbs\text{-}directions}(\text{Tenv}', \text{Merge-blocks}'))$

[5] $\mathit{check\text{-}mbs\text{-}directions}(\text{Tenv}', ) =$

[6] $\mathit{check\text{-}flbs\text{-}directions}(\text{Tenv}', \text{Flow-block}'\ \text{Flow-blocks}') =$
$\quad \mathit{join\text{-}errors}(\mathit{check\text{-}flb\text{-}directions}(\text{Tenv}', \text{Flow-block}'),$
$\qquad \mathit{check\text{-}flbs\text{-}directions}(\text{Tenv}', \text{Flow-blocks}'))$

[7] $\mathit{check\text{-}flbs\text{-}directions}(\text{Tenv}', ) =$

[8]
$$\begin{array}{l}\mathit{check\text{-}fub\text{-}directions}(\text{Tenv}', \mathit{LSC\_FUNCTION}\ \text{Identifier}';\\ \qquad\qquad\qquad\qquad\qquad\qquad \text{Statements}'\\ \qquad\qquad\qquad\qquad\qquad\qquad \text{Merge-blocks}'\ \mathit{LSC\_FUNCTION\_END};) =\\ \mathit{join\text{-}errors}(\mathit{check\text{-}dir}(\text{Tenv}',\\ \qquad\qquad \mathit{get\text{-}set\text{-}statements\text{-}from\text{-}stmts}(\text{Statements}'), \text{Identifier}'),\\ \qquad \mathit{check\text{-}dir}(\text{Tenv}',\\ \qquad\qquad \mathit{get\text{-}set\text{-}statements\text{-}from\text{-}fmbs}(\text{Merge-blocks}'), \text{Identifier}'))\end{array}$$

[9]
$$\frac{\begin{array}{c}\mathit{Statement}' = \mathit{Statement}',\\ \mathit{is\text{-}set\text{-}statement}(\mathit{Statement}') = \mathit{true}\end{array}}{\begin{array}{l}\mathit{get\text{-}set\text{-}statements\text{-}from\text{-}stmts}(\text{Statement}'\ \text{Statements}') =\\ \mathit{join\text{-}stmts}(\text{Statement}',\\ \qquad \mathit{get\text{-}set\text{-}statements\text{-}from\text{-}stmts}(\text{Statements}'))\end{array}}$$

[10]
$$\frac{\begin{array}{c}\mathit{Statement}' = \mathit{Statement}',\\ \mathit{is\text{-}set\text{-}statement}(\mathit{Statement}') \neq \mathit{true}\end{array}}{\begin{array}{l}\mathit{get\text{-}set\text{-}statements\text{-}from\text{-}stmts}(\text{Statement}'\ \text{Statements}') =\\ \mathit{get\text{-}set\text{-}statements\text{-}from\text{-}stmts}(\text{Statements}')\end{array}}$$

[11] $\textit{get-set-statements-from-stmts}() \ =$

[12] $\textit{get-set-statements-from-fmbs}(\text{Merge-block}'\ \text{Merge-blocks}') =$
$\textit{join-stmts}(\textit{get-fumb-set-statements}(\text{Merge-block}'),$
$\quad \textit{get-set-statements-from-fmbs}(\text{Merge-blocks}'))$

[13] $\textit{get-set-statements-from-fmbs}() \ =$

[14] $\textit{get-fumb-set-statements}(\textit{LSC\_MERGE}{\sqcup}\textit{LABEL\_}\ \text{Label}';$
$\quad \text{Statements}'\ \textit{LSC\_MERGE\_END};) =$
$\textit{get-set-statements-from-stmts}(\text{Statements}')$

[15] $\textit{check-dir}(\text{Tenv}', \text{Statements}', \text{Identifier}') =$
$\textit{check-dir-in-par-var-exp}(\text{Tenv}',$
$\quad \textit{get-par-var-exp-set-stmts}(\text{Statements}'), \text{Identifier}')$

[16] $$\frac{\textit{has-par-var-exp-with-directions}(\textit{set-expr}(\text{Statement}')) = \textit{true}}{\begin{array}{l}\textit{get-par-var-exp-set-stmts}(\text{Statement}'\ \text{Statements}') = \\ \textit{join-stmts}(\text{Statement}', \textit{get-par-var-exp-set-stmts}(\text{Statements}'))\end{array}}$$

[17] $$\frac{\textit{has-par-var-exp-with-directions}(\textit{set-expr}(\text{Statement}')) \neq \textit{true}}{\begin{array}{l}\textit{get-par-var-exp-set-stmts}(\text{Statement}'\ \text{Statements}') = \\ \textit{get-par-var-exp-set-stmts}(\text{Statements}')\end{array}}$$

[18] $\textit{get-par-var-exp-set-stmts}() \ =$

[19] $\textit{check-dir-in-par-var-exp}(\text{Tenv}', \text{Statements}', \text{Identifier}') =$
*these stmts from function block* Identifier′
*should be checked further* Statements′ Tenv′

[20] $\textit{check-mb-directions}(\text{Tenv}', \textit{LSC\_MERGE}{\sqcup}\textit{LABEL\_}\ \text{Label}';$
$\quad \text{Statements}'\ \textit{LSC\_MERGE\_END};) =$
$\textit{check-dir}(\text{Tenv}', \textit{get-set-statements-from-stmts}(\text{Statements}'), \text{Label}')$

[21] $\textit{check-dir}(\text{Tenv}', \text{Statements}', \text{Label}') =$
$\textit{check-dir-in-par-var-exp}(\text{Tenv}',$
$\quad \textit{get-par-var-exp-set-stmts}(\text{Statements}'), \text{Label}')$

[22] $\textit{check-dir-in-par-var-exp}(\text{Tenv}', \text{Statements}', \text{Label}') =$
*these stmts from merge block* Label′
*should be checked further* Statements′ Tenv′

[23] $\textit{check-flb-directions}(\text{Tenv}',$
$\quad \textit{LSC\_FLOW}\ \text{Flow-initiator}';$
$\quad \text{Statements}'\ \textit{LSC\_FLOW\_END};) =$
$\textit{check-dir}(\text{Tenv}',$
$\quad \textit{get-set-statements-from-stmts}(\text{Statements}'), \text{Flow-initiator}')$

[24]
$$\begin{aligned}&\mathit{check\text{-}flb\text{-}directions}(\mathrm{Tenv}',\\ &\qquad\qquad \mathit{LSC\_CNT\_FLOW}\ \mathrm{Flow\text{-}initiator}'\\ &\qquad\qquad\qquad \mathrm{Central\text{-}parameter\text{-}list}';\\ &\qquad\qquad\qquad \mathrm{Statements}'\ \mathit{LSC\_CNT\_FLOW\_END};) =\\ &\mathit{check\text{-}dir}(\mathrm{Tenv}', \mathrm{Statements}', \mathrm{Flow\text{-}initiator}')\end{aligned}$$

[25]
$$\begin{aligned}&\mathit{check\text{-}dir}(\mathrm{Tenv}', \mathrm{Statements}', \mathrm{Flow\text{-}initiator}') =\\ &\mathit{check\text{-}dir\text{-}in\text{-}par\text{-}var\text{-}exp}(\mathrm{Tenv}',\\ &\qquad\qquad \mathit{get\text{-}par\text{-}var\text{-}exp\text{-}set\text{-}stmts}(\mathrm{Statements}'), \mathrm{Flow\text{-}initiator}')\end{aligned}$$

[26]
$$\begin{aligned}&\mathit{check\text{-}dir\text{-}in\text{-}par\text{-}var\text{-}exp}(\mathrm{Tenv}', \mathrm{Statements}', \mathrm{Flow\text{-}initiator}') =\\ &\mathit{these\ stmts\ from\ flow\ block}\ \mathrm{Flow\text{-}initiator}'\\ &\mathit{should\ be\ checked\ further}\ \mathrm{Statements}'\ \mathrm{Tenv}'\end{aligned}$$

[27]
$$\frac{\mathit{Statements}''' = \mathit{Statements}'\ \mathit{Statements}''}{\mathit{join\text{-}stmts}(\mathrm{Statements}', \mathrm{Statements}'') = \mathrm{Statements}'''}$$

[28]
$$\mathit{is\text{-}set\text{-}statement}(\mathit{LSC\_SET}\ \mathrm{Set\text{-}expression}';) = \mathit{true}$$

[29]
$$\mathit{has\text{-}par\text{-}var\text{-}exp\text{-}with\text{-}directions}(\mathrm{Telegram\text{-}data\text{-}name}' : \mathrm{R\text{-}value}') = \mathit{false}$$

[30]
$$\begin{aligned}&\mathit{has\text{-}par\text{-}var\text{-}exp\text{-}with\text{-}directions}(\mathrm{Par\text{-}var\text{-}exp}' : \mathrm{R\text{-}value}') =\\ &\mathit{has\text{-}var\text{-}attribute}(\mathrm{Par\text{-}var\text{-}exp}')\end{aligned}$$

[31]
$$\mathit{has\text{-}var\text{-}attribute}(\mathrm{Identifier}') = \mathit{false}$$

[32]
$$\begin{aligned}&\mathit{has\text{-}var\text{-}attribute}(\mathrm{Identifier}'(\mathrm{Var\text{-}attribute}')) =\\ &\mathit{has\text{-}directions}(\mathrm{Var\text{-}attribute}')\end{aligned}$$

[33]
$$\mathit{has\text{-}var\text{-}attribute}(\mathrm{Identifier}'\ \{*\}) = \mathit{false}$$

[34]
$$\mathit{has\text{-}var\text{-}attribute}(\mathrm{Identifier}'\ \{\#\}) = \mathit{false}$$

[35]
$$\begin{aligned}&\mathit{has\text{-}var\text{-}attribute}(\mathrm{Identifier}'(\mathrm{Var\text{-}attribute}')\\ &\qquad\qquad [\mathrm{Simple\text{-}expression}']) =\\ &\mathit{has\text{-}directions}(\mathrm{Var\text{-}attribute}')\end{aligned}$$

[36]
$$\mathit{has\text{-}var\text{-}attribute}(\mathrm{Identifier}'[\mathrm{Simple\text{-}expression}']) = \mathit{false}$$

[37]
$$\mathit{has\text{-}var\text{-}attribute}(V.\ \mathrm{Identifier}') = \mathit{false}$$

[38]
$$\begin{aligned}&\mathit{has\text{-}var\text{-}attribute}(V.\ \mathrm{Identifier}'(\mathrm{Var\text{-}attribute}')) =\\ &\mathit{has\text{-}directions}(\mathrm{Var\text{-}attribute}')\end{aligned}$$

[39] $\textit{has-var-attribute}(\textit{V. }\text{Identifier}'\ \{*\}) \ = \ \textit{false}$

[40] $\textit{has-var-attribute}(\textit{V. }\text{Identifier}'\ \{\#\}) \ = \ \textit{false}$

[41] $\textit{has-var-attribute}(\textit{V. }\text{Identifier}'(\text{Var-attribute}')\ [\textit{Simple-expression}']) = \textit{has-directions}(\text{Var-attribute}')$

[42] $\textit{has-var-attribute}(\textit{V. }\text{Identifier}'[\textit{Simple-expression}']) = \textit{false}$

[43] $\textit{has-directions}(\text{Directions}', \text{Initiator}') \ = \ \textit{true}$

[44] $\textit{has-directions}(\text{Directions}') \ = \ \textit{true}$

[45] $\textit{has-directions}(\text{Initiator}') \ = \ \textit{false}$

[46] $$\frac{\textit{is-set-statement}(\textit{Statement}') \neq \textit{true}}{\textit{has-proper-directions}(\text{Tenv}', \textit{Statement}') = \textit{true}}$$

[47] $$\frac{\begin{array}{c}\textit{is-set-statement}(\textit{Statement}') = \textit{true},\\ \textit{Statement}' = \textit{Statement}',\\ \textit{has-par-var-exp-with-directions}(\textit{set-expr}(\textit{Statement}')) \neq \textit{true}\end{array}}{\textit{has-proper-directions}(\text{Tenv}', \textit{Statement}') = \textit{true}}$$

[48] $$\frac{\begin{array}{c}\textit{is-set-statement}(\textit{Statement}') = \textit{true},\\ \textit{Statement}' = \textit{Statement}',\\ \textit{has-par-var-exp-with-directions}(\textit{set-expr}(\textit{Statement}')) = \textit{true},\\ \textit{has-var-attribute}(\textit{par-var-exp}(\textit{set-expr}(\textit{Statement}'))) \neq \textit{true}\end{array}}{\textit{has-proper-directions}(\text{Tenv}', \textit{Statement}') = \textit{true}}$$

[49] $$\frac{\begin{array}{c}\textit{is-set-statement}(\textit{Statement}') = \textit{true},\\ \textit{Statement}' = \textit{Statement}',\\ \textit{has-par-var-exp-with-directions}(\textit{set-expr}(\textit{Statement}')) = \textit{true},\\ \textit{has-var-attribute}(\textit{par-var-exp}(\textit{set-expr}(\textit{Statement}'))) = \textit{true},\\ \textit{has-directions}(\textit{var-attribute}(\textit{par-var-exp}(\textit{set-expr}(\textit{Statement}')))) \neq \textit{true}\end{array}}{\textit{has-proper-directions}(\text{Tenv}', \textit{Statement}') = \textit{true}}$$

[50] $$\frac{\begin{array}{c}\textit{is-set-statement}(\textit{Statement}') = \textit{true},\\ \textit{Statement}' = \textit{Statement}',\\ \textit{has-par-var-exp-with-directions}(\textit{set-expr}(\textit{Statement}')) = \textit{true},\\ \textit{has-var-attribute}(\textit{par-var-exp}(\textit{set-expr}(\textit{Statement}'))) = \textit{true},\\ \textit{has-directions}(\textit{var-attribute}(\textit{par-var-exp}(\textit{set-expr}(\textit{Statement}')))) = \textit{true}\end{array}}{\begin{array}{l}\textit{has-proper-directions}(\text{Tenv}', \textit{Statement}') = \\ \textit{check-directions}(\text{Tenv}',\\ \qquad \textit{directions}(\textit{var-attribute}(\textit{par-var-exp}(\textit{set-expr}(\textit{Statement}')))),\\ \qquad \textit{dir-par-var}(\textit{par-var-exp}(\textit{set-expr}(\textit{Statement}'))))\end{array}}$$

[51] $$\frac{\textit{has-proper-directions}(\text{Tenv}', \textit{Statement}') = \textit{true}}{\begin{array}{l}\textit{these stmts from function block } \text{Identifier}'\\ \textit{should be checked further Statement}'\ \text{Statements}'\ \text{Tenv}' = \\ \textit{these stmts from function block } \text{Identifier}'\\ \textit{should be checked further } \text{Statements}'\ \text{Tenv}'\end{array}}$$

$$[52]\quad \frac{\textit{has-proper-directions}(\mathrm{Tenv}', \mathrm{Statement}') \neq \textit{true}}{\begin{array}{l} \textit{these stmts from function block } \mathrm{Identifier}' \\ \textit{should be checked further } \mathrm{Statement}' \\ \qquad\qquad\qquad\qquad\qquad \mathrm{Statements}' \\ \qquad\qquad\qquad\qquad\qquad \mathrm{Tenv}' \qquad\qquad = \\ \textit{join-errors}(\textit{function block } \mathrm{Identifier}' \\ \qquad\qquad \textit{contains set statement with illegal direction}(s)\ \mathrm{Statement}', \\ \qquad\qquad \textit{these stmts from function block } \mathrm{Identifier}' \\ \qquad\qquad \textit{should be checked further } \mathrm{Statements}' \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathrm{Tenv}') \end{array}}$$

$$[53]\quad \begin{array}{l} \textit{these stmts from function block } \mathrm{Identifier}' \\ \textit{should be checked further } \ \mathrm{Tenv}' \qquad\qquad = \end{array}$$

$$[54]\quad \frac{\textit{has-proper-directions}(\mathrm{Tenv}', \mathrm{Statement}') = \textit{true}}{\begin{array}{l} \textit{these stmts from merge block } \mathrm{Label}' \\ \textit{should be checked further } \mathrm{Statement}' \ \mathrm{Statements}' \ \mathrm{Tenv}' = \\ \textit{these stmts from merge block } \mathrm{Label}' \\ \textit{should be checked further } \mathrm{Statements}' \ \mathrm{Tenv}' \end{array}}$$

$$[55]\quad \frac{\textit{has-proper-directions}(\mathrm{Tenv}', \mathrm{Statement}') \neq \textit{true}}{\begin{array}{l} \textit{these stmts from merge block } \mathrm{Label}' \\ \textit{should be checked further } \mathrm{Statement}' \\ \qquad\qquad\qquad\qquad\qquad \mathrm{Statements}' \\ \qquad\qquad\qquad\qquad\qquad \mathrm{Tenv}' \qquad = \\ \textit{join-errors}(\textit{merge block } \mathrm{Label}' \\ \qquad\qquad \textit{contains set statement with illegal direction}(s)\ \mathrm{Statement}', \\ \qquad\qquad \textit{these stmts from merge block } \mathrm{Label}' \\ \qquad\qquad \textit{should be checked further } \mathrm{Statements}' \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathrm{Tenv}') \end{array}}$$

$$[56]\quad \textit{these stmts from merge block } \mathrm{Label}' \textit{ should be checked further } \ \mathrm{Tenv}' \ =$$

$$[57]\quad \frac{\textit{has-proper-directions}(\mathrm{Tenv}', \mathrm{Statement}') = \textit{true}}{\begin{array}{l} \textit{these stmts from flow block } \mathrm{Flow\text{-}initiator}' \\ \textit{should be checked further } \mathrm{Statement}' \ \mathrm{Statements}' \ \mathrm{Tenv}' = \\ \textit{these stmts from flow block } \mathrm{Flow\text{-}initiator}' \\ \textit{should be checked further } \mathrm{Statements}' \ \mathrm{Tenv}' \end{array}}$$

$$[58]\quad \frac{\textit{has-proper-directions}(\mathrm{Tenv}', \mathrm{Statement}') \neq \textit{true}}{\begin{array}{l} \textit{these stmts from flow block } \mathrm{Flow\text{-}initiator}' \\ \textit{should be checked further } \mathrm{Statement}' \\ \qquad\qquad\qquad\qquad\qquad \mathrm{Statements}' \\ \qquad\qquad\qquad\qquad\qquad \mathrm{Tenv}' \qquad\qquad = \\ \textit{join-errors}(\textit{flow block } \mathrm{Flow\text{-}initiator}' \\ \qquad\qquad \textit{contains set statement with illegal direction}(s)\ \mathrm{Statement}', \\ \qquad\qquad \textit{these stmts from flow block } \mathrm{Flow\text{-}initiator}' \\ \qquad\qquad \textit{should be checked further } \mathrm{Statements}' \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathrm{Tenv}') \end{array}}$$

[59] $$\begin{array}{l}\mathit{these\ stmts\ from\ flow\ block}\ \text{Flow-initiator}'\\ \mathit{should\ be\ checked\ further}\ \ \text{Tenv}' \qquad\qquad =\end{array}$$

[60] $$\frac{\begin{array}{c}\mathit{check\text{-}Directions}(\text{Directions}') = \,,\\ \mathit{in\text{-}directions\text{-}declaration}(\text{Directions}',\ \mathit{lookup\text{-}directions}(\text{Tenv}',\ \text{Identifier}')) = \mathit{true}\end{array}}{\mathit{check\text{-}directions}(\text{Tenv}',\ \text{Directions}',\ \text{Identifier}') = \mathit{true}}$$

[61] $\mathit{check\text{-}directions}(\text{Tenv}',\ \text{Directions}',\ \text{Identifier}')\ =\ \mathit{false}$ **otherwise**

[62] $$\begin{array}{l}\mathit{direction\text{-}in\text{-}declaration}(\text{Direction}',\ \text{Directions}') =\\ \quad \mathit{in\text{-}directions}(\text{Direction}',\ \text{Directions}')\end{array}$$

[63] $$\begin{array}{l}\mathit{direction\text{-}in\text{-}declaration}(\text{Direction}',\ \text{Directions}',\ \text{Directions}'') =\\ \quad \mathit{in\text{-}directions}(\text{Direction}',\ \text{Directions}')\\ \quad \mathit{or\ in\text{-}directions}(\text{Direction}',\ \text{Directions}'')\end{array}$$

[64] $$\begin{array}{l}\mathit{direction\text{-}in\text{-}declaration}(\text{Direction}',\ \text{Directions}',\ \text{Directions}'',\ \text{Directions}''') =\\ \quad \mathit{in\text{-}directions}(\text{Direction}',\ \text{Directions}')\\ \quad \mathit{or\ in\text{-}directions}(\text{Direction}',\ \text{Directions}'')\\ \quad \mathit{or\ in\text{-}directions}(\text{Direction}',\ \text{Directions}''')\end{array}$$

[65] $$\begin{array}{l}\mathit{direction\text{-}in\text{-}declaration}(\text{Direction}',\\ \qquad\qquad\qquad\qquad\qquad \text{Directions}',\\ \qquad\qquad\qquad\qquad\qquad \text{Directions}'',\\ \qquad\qquad\qquad\qquad\qquad \text{Directions}''',\\ \qquad\qquad\qquad\qquad\qquad \text{Directions}'''') =\\ \quad \mathit{in\text{-}directions}(\text{Direction}',\ \text{Directions}')\\ \quad \mathit{or\ in\text{-}directions}(\text{Direction}',\ \text{Directions}'')\\ \quad \mathit{or\ in\text{-}directions}(\text{Direction}',\ \text{Directions}''')\\ \quad \mathit{or\ in\text{-}directions}(\text{Direction}',\ \text{Directions}'''')\end{array}$$

[66] $\mathit{in\text{-}directions\text{-}declaration}(\text{Directions}',\ \mathit{Empty\text{-}directions})\ =\ \mathit{false}$

[67] $$\begin{array}{l}\mathit{in\text{-}directions\text{-}declaration}(\text{Direction}',\ \text{Dir-list}') =\\ \quad \mathit{direction\text{-}in\text{-}declaration}(\text{Direction}',\ \text{Dir-list}')\end{array}$$

[68] $$\begin{array}{l}\mathit{in\text{-}directions\text{-}declaration}(\text{Direction}'\ \text{Direction}'',\ \text{Dir-list}') =\\ \quad \mathit{direction\text{-}in\text{-}declaration}(\text{Direction}',\ \text{Dir-list}')\\ \quad \mathit{and\ direction\text{-}in\text{-}declaration}(\text{Direction}'',\ \text{Dir-list}')\end{array}$$

[69] $$\begin{array}{l}\mathit{in\text{-}directions\text{-}declaration}(\text{Direction}'\ \text{Direction}''\ \text{Direction}''',\ \text{Dir-list}') =\\ \quad \mathit{direction\text{-}in\text{-}declaration}(\text{Direction}',\ \text{Dir-list}')\\ \quad \mathit{and\ direction\text{-}in\text{-}declaration}(\text{Direction}'',\ \text{Dir-list}')\\ \quad \mathit{and\ direction\text{-}in\text{-}declaration}(\text{Direction}''',\ \text{Dir-list}')\end{array}$$

[70] $$\begin{array}{l}\mathit{in\text{-}directions\text{-}declaration}(\text{Direction}'\ \text{Direction}''\ \text{Direction}'''\ \text{Direction}'''',\ \text{Dir-list}') =\\ \quad \mathit{direction\text{-}in\text{-}declaration}(\text{Direction}',\ \text{Dir-list}')\\ \quad \mathit{and\ direction\text{-}in\text{-}declaration}(\text{Direction}'',\ \text{Dir-list}')\\ \quad \mathit{and\ direction\text{-}in\text{-}declaration}(\text{Direction}''',\ \text{Dir-list}')\\ \quad \mathit{and\ direction\text{-}in\text{-}declaration}(\text{Direction}'''',\ \text{Dir-list}')\end{array}$$

[71] $$\begin{array}{l}\mathit{in\text{-}directions}(\text{Direction}',\\ \qquad\qquad\quad \text{Direction}') = \mathit{true}\end{array}$$

[72] $\mathit{in\text{-}directions}(\mathit{Direction}', \mathit{Direction}'\ \mathit{Direction}'') = \mathit{true}$

[73] $\mathit{in\text{-}directions}(\mathit{Direction}', \mathit{Direction}''\ \mathit{Direction}') = \mathit{true}$

[74] $\mathit{in\text{-}directions}(\mathit{Direction}', \mathit{Direction}'\ \mathit{Direction}''\ \mathit{Direction}''') = \mathit{true}$

[75] $\mathit{in\text{-}directions}(\mathit{Direction}', \mathit{Direction}''\ \mathit{Direction}'\ \mathit{Direction}''') = \mathit{true}$

[76] $\mathit{in\text{-}directions}(\mathit{Direction}', \mathit{Direction}''\ \mathit{Direction}'''\ \mathit{Direction}') = \mathit{true}$

[77] $\mathit{in\text{-}directions}(\mathit{Direction}', \mathit{Direction}'\ \mathit{Direction}''\ \mathit{Direction}'''\ \mathit{Direction}'''') = \mathit{true}$

[78] $\mathit{in\text{-}directions}(\mathit{Direction}', \mathit{Direction}''\ \mathit{Direction}'\ \mathit{Direction}'''\ \mathit{Direction}'''') = \mathit{true}$

[79] $\mathit{in\text{-}directions}(\mathit{Direction}', \mathit{Direction}''\ \mathit{Direction}'''\ \mathit{Direction}'\ \mathit{Direction}'''') = \mathit{true}$

[80] $\mathit{in\text{-}directions}(\mathit{Direction}', \mathit{Direction}''\ \mathit{Direction}'''\ \mathit{Direction}''''\ \mathit{Direction}') = \mathit{true}$

[81] $\mathit{in\text{-}directions}(\mathit{Direction}', \mathit{Directions}') = \mathit{false}$ **otherwise**

[82] $\mathit{dir\text{-}par\text{-}var}(\mathit{Identifier}'(\mathit{Var\text{-}attribute}')) = \mathit{Identifier}'$

[83] $\mathit{dir\text{-}par\text{-}var}(P.\ \mathit{Identifier}'(\mathit{Var\text{-}attribute}')) = \mathit{Identifier}'$

[84] $\mathit{dir\text{-}par\text{-}var}(V.\ \mathit{Identifier}'(\mathit{Var\text{-}attribute}')) = \mathit{Identifier}'$

[85] $\mathit{dir\text{-}par\text{-}var}(\mathit{Identifier}'(\mathit{Var\text{-}attribute}')\ [\mathit{Simple\text{-}expression}']) = \mathit{Identifier}'$

[86] $\mathit{dir\text{-}par\text{-}var}(P.\ \mathit{Identifier}'(\mathit{Var\text{-}attribute}')\ [\mathit{Simple\text{-}expression}']) = \mathit{Identifier}'$

[87] $\mathit{dir\text{-}par\text{-}var}(V.\ \mathit{Identifier}'(\mathit{Var\text{-}attribute}')\ [\mathit{Simple\text{-}expression}']) = \mathit{Identifier}'$

# D Check-expressions

This module contains type checking functions for simple, test and par-var expressions. It both deals with type evaluation and error reporting. For instance the function `tc-test-expression` returns an error on invalid test-expression, while its counterpart `test-expression-type` evaluate its type.

**imports** Check-identifiers$^{(F)}$ Check-qualifiers$^{(H)}$
**exports**
**context-free syntax**
*is-var*(TENV, Identifier) $\rightarrow$ BOOL

| | |
|---|---|
| *is-list-id*(TENV, Identifier) | → BOOL |
| *is-central-id*(TENV, Identifier) | → BOOL |
| *is-integer-expression*(TENV, Simple-expression) | → BOOL |
| *tc-test-expression*(TENV, Test-expression) | → ERROR-LIST |
| *tc-simple-expression*(TENV, Simple-expression) | → ERROR-LIST |
| *tc-par-var-exp*(TENV, Simple-expression) | → ERROR-LIST |
| *tc-list-access* "(" TENV "," Identifier "[" Simple-expression "]" ")" | → ERROR-LIST |
| *tc-list-id*(TENV, Identifier) | → ERROR-LIST |
| *tc-list-index*(TENV, Simple-expression) | → ERROR-LIST |
| *tc-central-id*(TENV, Identifier) | → ERROR-LIST |
| *tc-var-attr*(TENV, Identifier, Var-attribute) | → ERROR-LIST |
| *test-expression-type*(TENV, Test-expression) | → Par-type |
| *simple-expression-type*(TENV, Simple-expression) | → Par-var-type |
| *par-var-exp-type*(TENV, Simple-expression) | → Par-var-type |
| "*Unresolved-test-expression*" | → Par-var-type |
| *incompatible* "*Test-expression*" *types lhs* ":" Par-var-type rhs ":" Par-var-type | → ERROR |
| *illegal types in* "*Test-expression*" *with lhs* ":" Par-var-type rhs ":" Par-var-type | → ERROR |
| *integer-expressions-expected* | → ERROR |
| *integer expression expected after monadic operator* "−" | → ERROR |
| *instead of* Par-var-type | → ERROR |
| *integer expressions expected for the binary* "+" *operator* | → ERROR |
| *instead of lhs* ":" Par-var-type rhs ":" Par-var-type | → ERROR |
| *integer expressions expected for the binary* "−" *operator* | → ERROR |
| *integer expressions expected for the binary* "∗" *operator* | → ERROR |
| *integer expression expected for list access function* | → ERROR |
| *instead of* "[" Simple-expression "]" *of type* Par-var-type | → ERROR |
| *integer expression expected for modulus operator* | → ERROR |
| *instead of* "\|" Simple-expression "\|" *of* Par-var-type | → ERROR |
| *illegal usage central parameter operator* "{∗}/{#}" | → ERROR |
| *after* "(*non␣var*)" *non central* Identifier of type Par-var-type | → ERROR |
| *illegal central-parameter-operator* "{∗}" *for* "*Parameter␣P.*" Identifier | → ERROR |
| *illegal central-parameter-operator* "{#}" *for* "*Parameter␣P.*" Identifier | → ERROR |
| *illegal* "*Var-attributes*" "(" Var-attribute ")" after "*Parameter␣P.*" Identifier | → ERROR |
| *illegal central-parameter-operator* "{∗}" | → ERROR |
| *for non central variable* "*V.*" Identifier | → ERROR |
| *illegal central-parameter-operator* "{#}" | → ERROR |

*illegal list access of nonlist* Identifier
of type Par-var-type $\rightarrow$ ERROR

*parameter* Identifier "*in␣lhs*" *of assignment* $\rightarrow$ ERROR

*illegal use of* "*Var-attribute␣(*" Var-attribute ")" $\rightarrow$ ERROR
*after non var* Identifier $\rightarrow$ ERROR

*direction list error in test-expression* "*%␣=*" $\rightarrow$ ERROR

**equations**

$$[1]\quad \frac{\textit{lookup-var}(\text{Identifier}', \textit{var-declarations}(\text{Tenv}')) \neq \textit{Undefined-par-var-type}}{\textit{is-var}(\text{Tenv}', \text{Identifier}') = \textit{true}}$$

$$[2]\quad \textit{is-var}(\text{Tenv}', \text{Identifier}') = \textit{false} \qquad \textbf{otherwise}$$

$$[3]\quad \textit{is-central-id}(\text{Tenv}', \text{Identifier}') = \textit{is-central-type}(\textit{lookup-var}(\text{Identifier}', \textit{var-declarations}(\text{Tenv}')))$$

$$[4]\quad \textit{is-list-id}(\text{Tenv}', \text{Identifier}') = \textit{is-list-type}(\textit{lookup-par-var}(\text{Tenv}', \text{Identifier}'))$$

$$[5]\quad \textit{is-integer-expression}(\text{Tenv}', \text{Simple-expression}') = \textit{test-compatible}(\textit{simple-expression-type}(\text{Tenv}', \text{Simple-expression}'), \textit{INT\_TYPE})$$

Error report section of test expressions

$$[6]\quad \frac{\begin{array}{c}\textit{simple-expression-type}(\text{Tenv}', \text{Simple-expression}') = \text{Par-var-type}', \\ \textit{simple-expression-type}(\text{Tenv}', \text{Simple-expression}'') = \text{Par-var-type}'', \\ \textit{type-equivalent}(\text{Par-var-type}', \text{Par-var-type}'') = \textit{false}\end{array}}{\begin{array}{l}\textit{tc-test-expression}(\text{Tenv}', \text{Simple-expression}' = \text{Simple-expression}'') = \\ \textit{join-errors}(\textit{illegal types in Test-expression with lhs} : \text{Par-var-type}' \\ \qquad \textit{rhs} : \text{Par-var-type}'', \\ \qquad \textit{join-errors}(\textit{integer-expressions-expected}, \\ \qquad\qquad \textit{join-errors}(\textit{tc-simple-expression}(\text{Tenv}', \text{Simple-expression}'), \\ \qquad\qquad\qquad \textit{tc-simple-expression}(\text{Tenv}', \text{Simple-expression}''))))\end{array}}$$

$$[7]\quad \frac{\begin{array}{c}\textit{simple-expression-type}(\text{Tenv}', \text{Simple-expression}') = \text{Par-var-type}', \\ \textit{simple-expression-type}(\text{Tenv}', \text{Simple-expression}'') = \text{Par-var-type}'', \\ \textit{type-equivalent}(\text{Par-var-type}', \text{Par-var-type}'') = \textit{false}\end{array}}{\begin{array}{l}\textit{tc-test-expression}(\text{Tenv}', \text{Simple-expression}' \diamond \text{Simple-expression}'') = \\ \textit{join-errors}(\textit{illegal types in Test-expression with lhs} : \text{Par-var-type}' \\ \qquad \textit{rhs} : \text{Par-var-type}'', \\ \qquad \textit{join-errors}(\textit{integer-expressions-expected}, \\ \qquad\qquad \textit{join-errors}(\textit{tc-simple-expression}(\text{Tenv}', \text{Simple-expression}'), \\ \qquad\qquad\qquad \textit{tc-simple-expression}(\text{Tenv}', \text{Simple-expression}''))))\end{array}}$$

$$[8]\quad \frac{\begin{array}{c}\mathit{simple\text{-}expression\text{-}type}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}') = \mathit{Par\text{-}var\text{-}type}',\\ \mathit{simple\text{-}expression\text{-}type}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}'') = \mathit{Par\text{-}var\text{-}type}'',\\ \mathit{test\text{-}compatible}(\mathit{Par\text{-}var\text{-}type}', \mathit{Par\text{-}var\text{-}type}'') = \mathit{false}\end{array}}{\begin{array}{l}\mathit{tc\text{-}test\text{-}expression}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}' < \mathit{Simple\text{-}expression}'') =\\ \mathit{join\text{-}errors}(\mathit{illegal\ types\ in\ Test\text{-}expression\ with\ lhs} : \mathit{Par\text{-}var\text{-}type}'\\ \qquad\qquad \mathit{rhs} : \mathit{Par\text{-}var\text{-}type}'',\\ \qquad\qquad \mathit{join\text{-}errors}(\mathit{integer\text{-}expressions\text{-}expected},\\ \qquad\qquad\qquad\qquad \mathit{join\text{-}errors}(\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}'),\\ \qquad\qquad\qquad\qquad\qquad\qquad \mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}''))))\end{array}}$$

$$[9]\quad \frac{\begin{array}{c}\mathit{simple\text{-}expression\text{-}type}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}') = \mathit{Par\text{-}var\text{-}type}',\\ \mathit{simple\text{-}expression\text{-}type}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}'') = \mathit{Par\text{-}var\text{-}type}'',\\ \mathit{test\text{-}compatible}(\mathit{Par\text{-}var\text{-}type}', \mathit{Par\text{-}var\text{-}type}'') = \mathit{false}\end{array}}{\begin{array}{l}\mathit{tc\text{-}test\text{-}expression}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}' > \mathit{Simple\text{-}expression}'') =\\ \mathit{join\text{-}errors}(\mathit{illegal\ types\ in\ Test\text{-}expression\ with\ lhs} : \mathit{Par\text{-}var\text{-}type}'\\ \qquad\qquad \mathit{rhs} : \mathit{Par\text{-}var\text{-}type}'',\\ \qquad\qquad \mathit{join\text{-}errors}(\mathit{integer\text{-}expressions\text{-}expected},\\ \qquad\qquad\qquad\qquad \mathit{join\text{-}errors}(\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}'),\\ \qquad\qquad\qquad\qquad\qquad\qquad \mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}''))))\end{array}}$$

$$[10]\quad \frac{\begin{array}{c}\mathit{simple\text{-}expression\text{-}type}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}') = \mathit{Par\text{-}var\text{-}type}',\\ \mathit{simple\text{-}expression\text{-}type}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}'') = \mathit{Par\text{-}var\text{-}type}'',\\ \mathit{test\text{-}compatible}(\mathit{Par\text{-}var\text{-}type}', \mathit{Par\text{-}var\text{-}type}'') = \mathit{false}\end{array}}{\begin{array}{l}\mathit{tc\text{-}test\text{-}expression}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}' \leq \mathit{Simple\text{-}expression}'') =\\ \mathit{join\text{-}errors}(\mathit{illegal\ types\ in\ Test\text{-}expression\ with\ lhs} : \mathit{Par\text{-}var\text{-}type}'\\ \qquad\qquad \mathit{rhs} : \mathit{Par\text{-}var\text{-}type}'',\\ \qquad\qquad \mathit{join\text{-}errors}(\mathit{integer\text{-}expressions\text{-}expected},\\ \qquad\qquad\qquad\qquad \mathit{join\text{-}errors}(\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}'),\\ \qquad\qquad\qquad\qquad\qquad\qquad \mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}''))))\end{array}}$$

$$[11]\quad \frac{\begin{array}{c}\mathit{simple\text{-}expression\text{-}type}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}') = \mathit{Par\text{-}var\text{-}type}',\\ \mathit{simple\text{-}expression\text{-}type}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}'') = \mathit{Par\text{-}var\text{-}type}'',\\ \mathit{test\text{-}compatible}(\mathit{Par\text{-}var\text{-}type}', \mathit{Par\text{-}var\text{-}type}'') = \mathit{false}\end{array}}{\begin{array}{l}\mathit{tc\text{-}test\text{-}expression}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}' \geq \mathit{Simple\text{-}expression}'') =\\ \mathit{join\text{-}errors}(\mathit{illegal\ types\ in\ Test\text{-}expression\ with\ lhs} : \mathit{Par\text{-}var\text{-}type}'\\ \qquad\qquad \mathit{rhs} : \mathit{Par\text{-}var\text{-}type}'',\\ \qquad\qquad \mathit{join\text{-}errors}(\mathit{integer\text{-}expressions\text{-}expected},\\ \qquad\qquad\qquad\qquad \mathit{join\text{-}errors}(\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}'),\\ \qquad\qquad\qquad\qquad\qquad\qquad \mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}', \mathit{Simple\text{-}expression}''))))\end{array}}$$

$$[12]\quad \frac{\begin{array}{c}\mathit{check\text{-}Dir\text{-}list}(\mathit{Dir\text{-}list}') = \mathit{Error\text{-}list}',\\ \mathit{Error\text{-}list}' =\end{array}}{\mathit{tc\text{-}test\text{-}expression}(\mathrm{Tenv}', \% = \mathit{Dir\text{-}list}') =}$$

$$[13]\quad \frac{\begin{array}{c}\mathit{check\text{-}Dir\text{-}list}(\mathit{Dir\text{-}list}') = \mathit{Error\text{-}list}',\\ \mathit{Error\text{-}list}' \neq ,\\ \mathit{Error\text{-}list}' = \mathit{Error\text{-}list}'\end{array}}{\begin{array}{l}\mathit{tc\text{-}test\text{-}expression}(\mathrm{Tenv}', \% = \mathit{Dir\text{-}list}') =\\ \mathit{join\text{-}errors}(\mathit{direction\ list\ error\ in\ test\text{-}expression}\ \%_{\sqcup}=,\\ \qquad\qquad \mathit{Error\text{-}list}')\end{array}}$$

[14] $\textit{tc-test-expression}(\textit{Tenv}', \textit{Test-expression}') = \qquad \textbf{otherwise}$

Error report section of simple expressions

[15]
$$\frac{\textit{effective-type}(\textit{simple-expression-type}(\textit{Tenv}', \textit{Simple-expression}')) = \textit{INT\_TYPE}}{\begin{array}{l}\textit{tc-simple-expression}(\textit{Tenv}', -\ \textit{Simple-expression}') = \\ \textit{tc-simple-expression}(\textit{Tenv}', \textit{Simple-expression}')\end{array}}$$

[16]
$$\frac{\begin{array}{l}\textit{simple-expression-type}(\textit{Tenv}', \\ \qquad\qquad \textit{Simple-expression}') = \textit{Par-var-type}'\end{array}}{\begin{array}{l}\textit{tc-simple-expression}(\textit{Tenv}', -\ \textit{Simple-expression}') = \\ \textit{join-errors}(\textit{integer expression expected after monadic operator} -, \\ \qquad \textit{join-errors}(\textit{instead of}\ \textit{Par-var-type}', \\ \qquad\qquad \textit{tc-simple-expression}(\textit{Tenv}', \textit{Simple-expression}')))\end{array}} \quad \textbf{otherwise}$$

[17]
$$\frac{\begin{array}{l}\textit{effective-type}(\textit{simple-expression-type}(\textit{Tenv}', \textit{Simple-expression}')) = \textit{INT\_TYPE}, \\ \textit{effective-type}(\textit{simple-expression-type}(\textit{Tenv}', \textit{Simple-expression}'')) = \textit{INT\_TYPE}\end{array}}{\begin{array}{l}\textit{tc-simple-expression}(\textit{Tenv}', \\ \qquad\qquad \textit{Simple-expression}' + \textit{Simple-expression}'') = \end{array}}$$

[18]
$$\frac{\begin{array}{l}\textit{simple-expression-type}(\textit{Tenv}', \textit{Simple-expression}') = \textit{Par-var-type}', \\ \textit{simple-expression-type}(\textit{Tenv}', \textit{Simple-expression}'') = \textit{Par-var-type}''\end{array}}{\begin{array}{l}\textit{tc-simple-expression}(\textit{Tenv}', \\ \qquad\qquad \textit{Simple-expression}' + \textit{Simple-expression}'') = \\ \textit{join-errors}(\textit{integer expressions expected for the binary} + \textit{operator}, \\ \qquad \textit{join-errors}(\textit{instead of lhs} : \textit{Par-var-type}' \\ \qquad\qquad \textit{rhs} : \textit{Par-var-type}'', \\ \qquad\qquad \textit{join-errors}(\textit{tc-simple-expression}(\textit{Tenv}', \textit{Simple-expression}'), \\ \qquad\qquad\qquad \textit{tc-simple-expression}(\textit{Tenv}', \textit{Simple-expression}''))))\end{array}} \quad \textbf{otherwise}$$

[19]
$$\frac{\begin{array}{l}\textit{effective-type}(\textit{simple-expression-type}(\textit{Tenv}', \textit{Simple-expression}')) = \textit{INT\_TYPE}, \\ \textit{effective-type}(\textit{simple-expression-type}(\textit{Tenv}', \textit{Simple-expression}'')) = \textit{INT\_TYPE}\end{array}}{\begin{array}{l}\textit{tc-simple-expression}(\textit{Tenv}', \\ \qquad\qquad \textit{Simple-expression}' - \textit{Simple-expression}'') = \end{array}}$$

[20]
$$\frac{\begin{array}{l}\textit{simple-expression-type}(\textit{Tenv}', \textit{Simple-expression}') = \textit{Par-var-type}', \\ \textit{simple-expression-type}(\textit{Tenv}', \textit{Simple-expression}'') = \textit{Par-var-type}''\end{array}}{\begin{array}{l}\textit{tc-simple-expression}(\textit{Tenv}', \\ \qquad\qquad \textit{Simple-expression}' - \textit{Simple-expression}'') = \\ \textit{join-errors}(\textit{integer expressions expected for the binary} - \textit{operator}, \\ \qquad \textit{join-errors}(\textit{instead of lhs} : \textit{Par-var-type}' \\ \qquad\qquad \textit{rhs} : \textit{Par-var-type}'', \\ \qquad\qquad \textit{join-errors}(\textit{tc-simple-expression}(\textit{Tenv}', \textit{Simple-expression}'), \\ \qquad\qquad\qquad \textit{tc-simple-expression}(\textit{Tenv}', \textit{Simple-expression}''))))\end{array}} \quad \textbf{otherwise}$$

[21]
$$\frac{\begin{array}{l}\textit{effective-type}(\textit{simple-expression-type}(\textit{Tenv}', \textit{Simple-expression}')) = \textit{INT\_TYPE}, \\ \textit{effective-type}(\textit{simple-expression-type}(\textit{Tenv}', \textit{Simple-expression}'')) = \textit{INT\_TYPE}\end{array}}{\begin{array}{l}\textit{tc-simple-expression}(\textit{Tenv}', \\ \qquad\qquad \textit{Simple-expression}' * \textit{Simple-expression}'') = \end{array}}$$

$$[22]\ \frac{\begin{array}{c}\textit{simple-expression-type}(\mathrm{Tenv}', \textit{Simple-expression}') = \textit{Par-var-type}', \\ \textit{simple-expression-type}(\mathrm{Tenv}', \textit{Simple-expression}'') = \textit{Par-var-type}''\end{array}}{\begin{array}{l}\textit{tc-simple-expression}(\mathrm{Tenv}', \\ \qquad\qquad \textit{Simple-expression}' * \textit{Simple-expression}'') = \\ \textit{join-errors}(\textit{integer expressions expected for the binary} * \textit{operator}, \\ \qquad \textit{join-errors}(\textit{instead of lhs} : \textit{Par-var-type}' \\ \qquad\qquad \textit{rhs} : \textit{Par-var-type}'', \\ \qquad\qquad \textit{join-errors}(\textit{tc-simple-expression}(\mathrm{Tenv}', \textit{Simple-expression}'), \\ \qquad\qquad\qquad \textit{tc-simple-expression}(\mathrm{Tenv}', \textit{Simple-expression}''))))\end{array}}\quad \textbf{otherwise}$$

$$[23]\ \textit{tc-simple-expression}(\mathrm{Tenv}', \textit{Number}') =$$

$$[24]\ \begin{array}{l}\textit{tc-simple-expression}(\mathrm{Tenv}', \textit{Telegram-data-name}') = \\ \quad \textit{exist-telegram-data}(\textit{Telegram-data-name}', \\ \qquad\qquad \textit{telegram-data-declarations}(\mathrm{Tenv}'))\end{array}$$

$$[25]\ \frac{\textit{is-integer-expression}(\mathrm{Tenv}', \textit{Simple-expression}') = \textit{true}}{\textit{tc-simple-expression}(\mathrm{Tenv}', \mid \textit{Simple-expression}' \mid) =}$$

$$[26]\ \frac{\textit{simple-expression-type}(\mathrm{Tenv}', \textit{Simple-expression}') = \textit{Par-var-type}'}{\begin{array}{l}\textit{tc-simple-expression}(\mathrm{Tenv}', \mid \textit{Simple-expression}' \mid) = \\ \textit{join-errors}(\textit{integer expression expected for modulus operator}, \\ \qquad \textit{join-errors}(\textit{instead of} \mid \textit{Simple-expression}' \mid \textit{of Par-var-type}', \\ \qquad\qquad \textit{tc-simple-expression}(\mathrm{Tenv}', \textit{Simple-expression}')))\end{array}}\quad \textbf{otherwise}$$

Par-var-exp $->$ Simple-expression type evaluation

$$[27]\ \begin{array}{l}\textit{par-var-exp-type}(\mathrm{Tenv}', \textit{Par-var-exp}') = \\ \quad \textit{simple-expression-type}(\mathrm{Tenv}', \textit{Par-var-exp}')\end{array}$$

$$[28]\ \begin{array}{l}\textit{tc-par-var-exp}(\mathrm{Tenv}', \textit{Par-var-exp}') = \\ \quad \textit{tc-simple-expression}(\mathrm{Tenv}', \textit{Par-var-exp}')\end{array}$$

$$[29]\ \begin{array}{l}\textit{tc-simple-expression}(\mathrm{Tenv}', \textit{Identifier}') = \\ \quad \textit{exist-par-var}(\mathrm{Tenv}', \textit{Identifier}')\end{array}$$

$$[30]\ \begin{array}{l}\textit{tc-simple-expression}(\mathrm{Tenv}', \textit{Identifier}' \\ \qquad\qquad (\textit{Var-attribute}')) = \\ \quad \textit{tc-var-attr}(\mathrm{Tenv}', \textit{Identifier}', \textit{Var-attribute}')\end{array}$$

$$[31]\ \begin{array}{l}\textit{tc-simple-expression}(\mathrm{Tenv}', \textit{Identifier}'\ \{*\}) = \\ \quad \textit{tc-central-id}(\mathrm{Tenv}', \textit{Identifier}')\end{array}$$

$$[32]\ \begin{array}{l}\textit{tc-simple-expression}(\mathrm{Tenv}', \textit{Identifier}'\ \{\#\}) = \\ \quad \textit{tc-central-id}(\mathrm{Tenv}', \textit{Identifier}')\end{array}$$

$$[33]\ \begin{array}{l}\textit{tc-simple-expression}(\mathrm{Tenv}', \textit{Identifier}' \\ \qquad\qquad (\textit{Var-attribute}')[\textit{Simple-expression}']) = \\ \quad \textit{join-errors}(\textit{tc-list-access}(\mathrm{Tenv}', \textit{Identifier}'[\textit{Simple-expression}']), \\ \qquad\qquad \textit{tc-var-attr}(\mathrm{Tenv}', \textit{Identifier}', \textit{Var-attribute}'))\end{array}$$

$$[34]\ \begin{array}{l}\textit{tc-simple-expression}(\mathrm{Tenv}', \textit{Identifier}'[\textit{Simple-expression}']) = \\ \quad \textit{tc-list-access}(\mathrm{Tenv}', \textit{Identifier}'[\textit{Simple-expression}'])\end{array}$$

[35] $$\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}',\ P.\ \mathrm{Identifier}') = \mathit{exist\text{-}par}(\mathrm{Identifier}',\ \mathit{par\text{-}declarations}(\mathrm{Tenv}'))$$

[36] $$\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}',\ P.\ \mathrm{Identifier}'\ (\mathrm{Var\text{-}attribute}')) = \mathit{illegal\ Var\text{-}attributes}(\mathrm{Var\text{-}attribute}')\ \mathit{after\ Parameter}_{\sqcup}P.\ \mathrm{Identifier}'$$

[37] $$\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}',\ P.\ \mathrm{Identifier}'\ \{*\}) = \mathit{illegal\ central\text{-}parameter\text{-}operator}\ \{*\}\ \mathit{for\ Parameter}_{\sqcup}P.\ \mathrm{Identifier}'$$

[38] $$\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}',\ \mathrm{Identifier}'\ \{\#\}) = \mathit{illegal\ central\text{-}parameter\text{-}operator}\ \{\#\}\ \mathit{for\ Parameter}_{\sqcup}P.\ \mathrm{Identifier}'$$

[39] $$\begin{aligned}&\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}',\ P.\ \mathrm{Identifier}'\ (\mathrm{Var\text{-}attribute}')[\mathrm{Simple\text{-}expression}']) = \\ &\mathit{join\text{-}errors}(\mathit{illegal\ Var\text{-}attributes}(\mathrm{Var\text{-}attribute}')\ \mathit{after\ Parameter}_{\sqcup}P.\ \mathrm{Identifier}', \\ &\qquad \mathit{tc\text{-}list\text{-}access}(\mathrm{Tenv}',\ \mathrm{Identifier}'[\mathrm{Simple\text{-}expression}']))\end{aligned}$$

[40] $$\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}',\ P.\ \mathrm{Identifier}'\ [\mathrm{Simple\text{-}expression}']) = \mathit{tc\text{-}list\text{-}access}(\mathrm{Tenv}',\ \mathrm{Identifier}'[\mathrm{Simple\text{-}expression}'])$$

[41] $$\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}',\ V.\ \mathrm{Identifier}') = \mathit{exist\text{-}var}(\mathrm{Identifier}',\ \mathit{var\text{-}declarations}(\mathrm{Tenv}'))$$

[42] $$\begin{aligned}&\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}',\ V.\ \mathrm{Identifier}'\ (\mathrm{Var\text{-}attribute}')) = \\ &\mathit{join\text{-}errors}(\mathit{exist\text{-}var}(\mathrm{Identifier}',\ \mathit{var\text{-}declarations}(\mathrm{Tenv}')), \\ &\qquad \mathit{tc\text{-}var\text{-}attr}(\mathrm{Tenv}',\ \mathrm{Identifier}',\ \mathrm{Var\text{-}attribute}'))\end{aligned}$$

[43] $$\frac{\mathit{is\text{-}central\text{-}type}(\mathit{lookup\text{-}var}(\mathrm{Identifier}',\ \mathit{var\text{-}declarations}(\mathrm{Tenv}'))) = \mathit{true}}{\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}',\ V.\ \mathrm{Identifier}'\ \{*\}) =}$$

[44] $$\frac{\mathit{is\text{-}central\text{-}type}(\mathit{lookup\text{-}var}(\mathrm{Identifier}',\ \mathit{var\text{-}declarations}(\mathrm{Tenv}'))) \neq \mathit{true}}{\begin{aligned}&\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}',\ V.\ \mathrm{Identifier}'\ \{*\}) = \\ &\mathit{join\text{-}errors}(\mathit{exist\text{-}var}(\mathrm{Identifier}',\ \mathit{var\text{-}declarations}(\mathrm{Tenv}')), \\ &\qquad \mathit{join\text{-}errors}(\mathit{illegal\ central\text{-}parameter\text{-}operator}\ \{*\}, \\ &\qquad\qquad \mathit{for\ non\ central\ variable}\ V.\ \mathrm{Identifier}'))\end{aligned}}$$

[45] $$\frac{\mathit{is\text{-}central\text{-}type}(\mathit{lookup\text{-}var}(\mathrm{Identifier}',\ \mathit{var\text{-}declarations}(\mathrm{Tenv}'))) = \mathit{true}}{\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}',\ V.\ \mathrm{Identifier}'\ \{\#\}) =}$$

[46] $$\frac{\mathit{is\text{-}central\text{-}type}(\mathit{lookup\text{-}var}(\mathrm{Identifier}',\ \mathit{var\text{-}declarations}(\mathrm{Tenv}'))) \neq \mathit{true}}{\begin{aligned}&\mathit{tc\text{-}simple\text{-}expression}(\mathrm{Tenv}',\ V.\ \mathrm{Identifier}'\ \{\#\}) = \\ &\mathit{join\text{-}errors}(\mathit{exist\text{-}var}(\mathrm{Identifier}',\ \mathit{var\text{-}declarations}(\mathrm{Tenv}')), \\ &\qquad \mathit{join\text{-}errors}(\mathit{illegal\ central\text{-}parameter\text{-}operator}\ \{\#\}, \\ &\qquad\qquad \mathit{for\ non\ central\ variable}\ V.\ \mathrm{Identifier}'))\end{aligned}}$$

[47] $$\textit{tc-simple-expression}(\mathit{Tenv}',\ \mathit{V.\ Identifier}'(\mathit{Var\text{-}attribute}')[\mathit{Simple\text{-}expression}']) =$$
$$\textit{join-errors}(\textit{tc-list-access}(\mathit{Tenv}', \mathit{Identifier}'[\mathit{Simple\text{-}expression}']), \textit{tc-var-attr}(\mathit{Tenv}', \mathit{Identifier}', \mathit{Var\text{-}attribute}'))$$

[48] $$\textit{tc-simple-expression}(\mathit{Tenv}',\ \mathit{V.\ Identifier}'[\mathit{Simple\text{-}expression}']) =$$
$$\textit{tc-list-access}(\mathit{Tenv}', \mathit{Identifier}'[\mathit{Simple\text{-}expression}'])$$

Test-expressions Type evaluation =<><><=>=

[49] $$\frac{\textit{type-equivalent}(\textit{simple-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}'), \textit{simple-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}'')) = \mathit{true}}{\textit{test-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}' = \mathit{Simple\text{-}expression}'') = \mathit{BOOL\_TYPE}}$$

[50] $$\frac{\textit{type-equivalent}(\textit{simple-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}'), \textit{simple-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}'')) = \mathit{true}}{\textit{test-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}' \diamond \mathit{Simple\text{-}expression}'') = \mathit{BOOL\_TYPE}}$$

[51] $$\frac{\textit{test-compatible}(\textit{simple-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}'), \textit{simple-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}'')) = \mathit{true}}{\textit{test-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}' < \mathit{Simple\text{-}expression}'') = \mathit{BOOL\_TYPE}}$$

[52] $$\frac{\textit{test-compatible}(\textit{simple-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}'), \textit{simple-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}'')) = \mathit{true}}{\textit{test-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}' > \mathit{Simple\text{-}expression}'') = \mathit{BOOL\_TYPE}}$$

[53] $$\frac{\textit{test-compatible}(\textit{simple-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}'), \textit{simple-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}'')) = \mathit{true}}{\textit{test-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}' \leq \mathit{Simple\text{-}expression}'') = \mathit{BOOL\_TYPE}}$$

[54] $$\frac{\textit{test-compatible}(\textit{simple-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}'), \textit{simple-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}'')) = \mathit{true}}{\textit{test-expression-type}(\mathit{Tenv}', \mathit{Simple\text{-}expression}' \geq \mathit{Simple\text{-}expression}'') = \mathit{BOOL\_TYPE}}$$

[55] $\textit{test-expression-type}(\mathit{Tenv}', \% = \mathit{Dir\text{-}list}') = \mathit{BOOL\_TYPE}$

[56] $\textit{test-expression-type}(\mathit{Tenv}', \mathit{Test\text{-}expression}') =$
$\mathit{Unresolved\text{-}test\text{-}expression}$ **otherwise**

Simple-expression Type evaluation

[57] $$\frac{\textit{effective-type}(\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Simple\text{-}expression}')) = \textit{INT\_TYPE}}{\textit{simple-expression-type}(\mathrm{Tenv}', -\ \mathrm{Simple\text{-}expression}') = \textit{INT\_TYPE}}$$

[58] $\textit{simple-expression-type}(\mathrm{Tenv}', -\ \mathrm{Simple\text{-}expression}') =$
$\textit{Type-mismatch}$ **otherwise**

[59] $$\frac{\begin{array}{l}\textit{test-compatible}(\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Simple\text{-}expression}'),\\ \qquad\qquad \textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Simple\text{-}expression}'')) = \textit{true}\end{array}}{\begin{array}{l}\textit{simple-expression-type}(\mathrm{Tenv}',\\ \qquad\qquad \mathrm{Simple\text{-}expression}' + \mathrm{Simple\text{-}expression}'') = \textit{INT\_TYPE}\end{array}}$$

[60] $$\frac{\begin{array}{l}\textit{test-compatible}(\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Simple\text{-}expression}'),\\ \qquad\qquad \textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Simple\text{-}expression}'')) = \textit{true}\end{array}}{\begin{array}{l}\textit{simple-expression-type}(\mathrm{Tenv}',\\ \qquad\qquad \mathrm{Simple\text{-}expression}' - \mathrm{Simple\text{-}expression}'') = \textit{INT\_TYPE}\end{array}}$$

[61] $$\frac{\begin{array}{l}\textit{test-compatible}(\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Simple\text{-}expression}'),\\ \qquad\qquad \textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Simple\text{-}expression}'')) = \textit{true}\end{array}}{\begin{array}{l}\textit{simple-expression-type}(\mathrm{Tenv}',\\ \qquad\qquad \mathrm{Simple\text{-}expression}' * \mathrm{Simple\text{-}expression}'') = \textit{INT\_TYPE}\end{array}}$$

[62] $\textit{simple-expression-type}(\mathrm{Tenv}', 0) = \textit{BOOLEAN\_CONSTRUCTOR}$

[63] $\textit{simple-expression-type}(\mathrm{Tenv}', 1) = \textit{BOOLEAN\_CONSTRUCTOR}$

[64] $$\frac{\begin{array}{c}\texttt{Number} \neq 0,\\ \texttt{Number} \neq 1\end{array}}{\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Number}') = \textit{NON\_BOOLEAN\_INT}}$$

[65] $$\frac{\textit{is-integer-expression}(\mathrm{Tenv}', \mathrm{Simple\text{-}expression}') = \textit{true}}{\textit{simple-expression-type}(\mathrm{Tenv}', \mid \mathrm{Simple\text{-}expression}' \mid) = \textit{INT\_TYPE}}$$

[66] $$\frac{\begin{array}{c}\textit{is-list-id}(\mathrm{Tenv}', \mathrm{Identifier}') = \textit{true},\\ \textit{is-integer-expression}(\mathrm{Tenv}', \mathrm{Simple\text{-}expression}') = \textit{true}\end{array}}{\begin{array}{l}\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Identifier}'[\mathrm{Simple\text{-}expression}']) =\\ \textit{list-element-type}(\textit{lookup-par-var}(\mathrm{Tenv}', \mathrm{Identifier}'))\end{array}}$$

Par-var-exp $->$ Simple-expression type evaluation

[67] $\textit{par-var-exp-type}(\mathrm{Tenv}', \mathrm{Par\text{-}var\text{-}exp}') =$
$\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Par\text{-}var\text{-}exp}')$

[68] $\textit{tc-par-var-exp}(\mathrm{Tenv}', \mathrm{Par\text{-}var\text{-}exp}') =$
$\textit{tc-simple-expression}(\mathrm{Tenv}', \mathrm{Par\text{-}var\text{-}exp}')$

[69] $\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Telegram\text{-}data\text{-}name}') =$
$\textit{lookup-telegram-data}(\mathrm{Telegram\text{-}data\text{-}name}', \textit{telegram-data-declarations}(\mathrm{Tenv}'))$

[70] $$\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Identifier}') = \textit{lookup-par-var}(\mathrm{Tenv}', \mathrm{Identifier}')$$

[71] $$\frac{\textit{is-var}(\mathrm{Tenv}', \mathrm{Identifier}') = \textit{true}}{\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Identifier}'\ (\mathrm{Var\text{-}attribute}')) = \textit{lookup-var}(\mathrm{Identifier}', \textit{var-declarations}(\mathrm{Tenv}'))}$$

[72] $$\frac{\textit{is-central-id}(\mathrm{Tenv}', \mathrm{Identifier}') = \textit{true}}{\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Identifier}'\ \{*\}) = \textit{lookup-var}(\mathrm{Identifier}', \textit{var-declarations}(\mathrm{Tenv}'))}$$

[73] $$\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Identifier}') = \textit{No-central-type} \quad \textbf{otherwise}$$

[74] $$\frac{\textit{is-central-id}(\mathrm{Tenv}', \mathrm{Identifier}') = \textit{true}}{\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Identifier}'\ \{\#\}) = \textit{lookup-var}(\mathrm{Identifier}', \textit{var-declarations}(\mathrm{Tenv}'))}$$

[75] $$\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Identifier}'\ \{\#\}) = \textit{No-central-type} \quad \textbf{otherwise}$$

[76] $$\frac{\begin{array}{c}\textit{is-integer-expression}(\mathrm{Tenv}', \mathrm{Simple\text{-}expression}') = \textit{true},\\ \textit{is-list-id}(\mathrm{Tenv}', \mathrm{Identifier}') = \textit{true},\\ \textit{is-var}(\mathrm{Tenv}', \mathrm{Identifier}') = \textit{true}\end{array}}{\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Identifier}'\ (\mathrm{Var\text{-}attribute}')[\mathrm{Simple\text{-}expression}']) = \textit{list-element-type}(\textit{lookup-var}(\mathrm{Identifier}', \textit{var-declarations}(\mathrm{Tenv}')))}$$

[77] $$\frac{\begin{array}{c}\textit{is-integer-expression}(\mathrm{Tenv}', \mathrm{Simple\text{-}expression}') = \textit{true},\\ \textit{is-list-id}(\mathrm{Tenv}', \mathrm{Identifier}') = \textit{true}\end{array}}{\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{Identifier}'\ [\mathrm{Simple\text{-}expression}']) = \textit{list-element-type}(\textit{lookup-var}(\mathrm{Identifier}', \textit{var-declarations}(\mathrm{Tenv}')))}$$

[78] $$\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{P.\ Identifier}') = \textit{lookup-par}(\mathrm{Identifier}', \textit{par-declarations}(\mathrm{Tenv}'))$$

[79] $$\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{P.\ Identifier}'\ (\mathrm{Var\text{-}attribute}')) = \textit{Non-var-type}$$

[80] $$\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{P.\ Identifier}'\ \{*\}) = \textit{No-central-type}$$

[81] $$\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{P.\ Identifier}'\ \{\#\}) = \textit{No-central-type}$$

[82] $$\frac{\begin{array}{c}\textit{is-integer-expression}(\mathrm{Tenv}', \mathrm{Simple\text{-}expression}') = \textit{true},\\ \textit{is-list-id}(\mathrm{Tenv}', \mathrm{Identifier}') = \textit{true}\end{array}}{\textit{simple-expression-type}(\mathrm{Tenv}', \mathrm{P.\ Identifier}'\ (\mathrm{Var\text{-}attribute}')[\mathrm{Simple\text{-}expression}']) = \textit{Non-var-type}}$$

[83] $$\frac{\begin{array}{c}\textit{is-integer-expression}(\mathit{Tenv}', \text{Simple-expression}') = \textit{true},\\ \textit{is-list-id}(\mathit{Tenv}', \text{Identifier}') = \textit{true}\end{array}}{\begin{array}{l}\textit{simple-expression-type}(\mathit{Tenv}', \text{P. Identifier}'\\ \qquad\qquad\qquad [\text{Simple-expression}']) =\\ \textit{list-element-type}(\textit{lookup-par}(\text{Identifier}',\\ \qquad\qquad\qquad \textit{par-declarations}(\mathit{Tenv}')))\end{array}}$$

[84] $\textit{simple-expression-type}(\mathit{Tenv}',\ \text{V. Identifier}') =$
$\textit{lookup-var}(\text{Identifier}', \textit{var-declarations}(\mathit{Tenv}'))$

[85] $$\begin{array}{l}\textit{simple-expression-type}(\mathit{Tenv}',\ \text{V. Identifier}'\\ \qquad\qquad\qquad (\text{Var-attribute}')) =\\ \textit{lookup-var}(\text{Identifier}', \textit{var-declarations}(\mathit{Tenv}'))\end{array}$$

[86] $$\frac{\textit{is-central-id}(\mathit{Tenv}', \text{Identifier}') = \textit{true}}{\begin{array}{l}\textit{simple-expression-type}(\mathit{Tenv}',\ \text{V. Identifier}'\ \{*\}) =\\ \textit{lookup-var}(\text{Identifier}', \textit{var-declarations}(\mathit{Tenv}'))\end{array}}$$

[87] $\textit{simple-expression-type}(\mathit{Tenv}',\ \text{V. Identifier}'\ \{*\}) =$
$\textit{No-central-type}$ **otherwise**

[88] $$\frac{\textit{is-central-id}(\mathit{Tenv}', \text{Identifier}') = \textit{true}}{\begin{array}{l}\textit{simple-expression-type}(\mathit{Tenv}',\ \text{V. Identifier}'\ \{\#\}) =\\ \textit{lookup-var}(\text{Identifier}', \textit{var-declarations}(\mathit{Tenv}'))\end{array}}$$

[89] $\textit{simple-expression-type}(\mathit{Tenv}',\ \text{V. Identifier}'\ \{\#\}) =$
$\textit{No-central-type}$ **otherwise**

[90] $$\frac{\begin{array}{c}\textit{is-integer-expression}(\mathit{Tenv}', \text{Simple-expression}') = \textit{true},\\ \textit{is-list-id}(\mathit{Tenv}', \text{Identifier}') = \textit{true}\end{array}}{\begin{array}{l}\textit{simple-expression-type}(\mathit{Tenv}',\ \text{V. Identifier}'\\ \qquad\qquad\qquad (\text{Var-attribute}')[\text{Simple-expression}']) =\\ \textit{list-element-type}(\textit{lookup-var}(\text{Identifier}',\\ \qquad\qquad\qquad \textit{var-declarations}(\mathit{Tenv}')))\end{array}}$$

[91] $$\frac{\begin{array}{c}\textit{is-integer-expression}(\mathit{Tenv}', \text{Simple-expression}') = \textit{true},\\ \textit{is-list-id}(\mathit{Tenv}', \text{Identifier}') = \textit{true}\end{array}}{\begin{array}{l}\textit{simple-expression-type}(\mathit{Tenv}', \text{Identifier}'\\ \qquad\qquad\qquad [\text{Simple-expression}']) =\\ \textit{list-element-type}(\textit{lookup-var}(\text{Identifier}',\\ \qquad\qquad\qquad \textit{var-declarations}(\mathit{Tenv}')))\end{array}}$$

[92] $\textit{simple-expression-type}(\mathit{Tenv}', \text{Simple-expression}') =$
$\textit{Type-mismatch}$ **otherwise**

[93] $\textit{tc-list-access}(\mathit{Tenv}', \text{Identifier}'[\text{Simple-expression}']) =$
$\textit{join-errors}(\textit{tc-list-id}(\mathit{Tenv}', \text{Identifier}'),$
$\qquad \textit{tc-list-index}(\mathit{Tenv}', \text{Simple-expression}'))$

[94] $$\frac{\textit{is-list-id}(\mathit{Tenv}', \text{Identifier}') = \textit{true}}{\textit{tc-list-id}(\mathit{Tenv}', \text{Identifier}') =}$$

[95] $$\frac{\textit{lookup-par-var}(\text{Tenv}', \text{Identifier}') = \textit{Par-var-type}'}{\begin{array}{l}\textit{tc-list-id}(\text{Tenv}', \text{Identifier}') = \\ \textit{join-errors}(\textit{exist-par-var}(\text{Tenv}', \text{Identifier}'), \\ \qquad\qquad \textit{illegal list access of nonlist } \text{Identifier}' \\ \qquad\qquad \textit{of type } \text{Par-var-type}')\end{array}} \quad \textbf{otherwise}$$

[96] $$\frac{\textit{is-integer-expression}(\text{Tenv}', \text{Simple-expression}') = \textit{true}}{\textit{tc-list-index}(\text{Tenv}', \text{Simple-expression}') =}$$

[97] $$\frac{\textit{simple-expression-type}(\text{Tenv}', \text{Simple-expression}') = \textit{Par-var-type}'}{\begin{array}{l}\textit{tc-list-index}(\text{Tenv}', \text{Simple-expression}') = \\ \textit{join-errors}(\textit{integer expression expected for list access function}, \\ \qquad\qquad \textit{join-errors}(\textit{instead of}[\text{Simple-expression}'] \\ \qquad\qquad\qquad\qquad\qquad \textit{of type } \text{Par-var-type}', \\ \qquad\qquad\qquad\qquad \textit{tc-simple-expression}(\text{Tenv}', \text{Simple-expression}')))\end{array}} \quad \textbf{otherwise}$$

[98] $$\frac{\textit{is-central-id}(\text{Tenv}', \text{Identifier}') = \textit{true}}{\textit{tc-central-id}(\text{Tenv}', \text{Identifier}') =}$$

[99] $$\begin{array}{l}\textit{tc-central-id}(\text{Tenv}', \text{Identifier}') = \\ \quad \textit{join-errors}(\textit{illegal usage central parameter operator } \{*\}/\{\#\}, \qquad \textbf{otherwise} \\ \qquad\qquad \textit{join-errors}(\textit{after } (\textit{non}_\sqcup\textit{var}) \textit{ non central } \text{Identifier}' \\ \qquad\qquad\qquad\qquad \textit{of type lookup-var}(\text{Identifier}', \\ \qquad\qquad\qquad\qquad\qquad\qquad \textit{var-declarations}(\text{Tenv}')), \\ \qquad\qquad\qquad\qquad \textit{exist-var}(\text{Identifier}', \textit{var-declarations}(\text{Tenv}'))))\end{array}$$

[100] $$\frac{\textit{is-var}(\text{Tenv}', \text{Identifier}') = \textit{true}}{\textit{tc-var-attr}(\text{Tenv}', \text{Identifier}', \text{Var-attribute}') =}$$

[101] $$\begin{array}{l}\textit{tc-var-attr}(\text{Tenv}', \text{Identifier}', \text{Var-attribute}') = \\ \quad \textit{join-errors}(\textit{illegal use of Var-attribute}_\sqcup(\text{ Var-attribute}'), \qquad \textbf{otherwise} \\ \qquad\qquad \textit{join-errors}(\textit{after non var } \text{Identifier}', \\ \qquad\qquad\qquad\qquad \textit{exist-var}(\text{Identifier}', \textit{var-declarations}(\text{Tenv}'))))\end{array}$$

# E Check-fb-sends

**check-sends** checks whether a send statements in a flow-block, is the last statement in that block. A similar check should be performed on continue at statements for 'Split blocks'. See recommendations (cf. Section 7).

**imports** LSC-mappings$^{(Q)}$ Errors$^{(J)}$
**exports**
**context-free syntax**
*check-sends*(Flow-blocks) $\rightarrow$ ERROR-LIST
*check-send-in-fb*(Flow-block) $\rightarrow$ ERROR-LIST

*is-send-stm*(Statement) $\rightarrow$ BOOL *contains-send-stmt*(Statement∗) → BOOL

*flow block* Flow-initiator *contains send statement* which is not the last statement → ERROR

*only last statement may be send in* Statements ";" Flow-initiator → ERROR

**equations**

[1] $\textit{check-sends}(\textit{Flow-block}'\ \textit{Flow-blocks}') =$
$\textit{join-errors}(\textit{check-send-in-fb}(\textit{Flow-block}'),\ \textit{check-sends}(\textit{Flow-blocks}'))$

[2] $\textit{check-sends}() =$

[3] $\textit{check-send-in-fb}(\textit{LSC\_FLOW Flow-initiator}';\ \textit{Statements}'\ \textit{LSC\_FLOW\_END};) =$
$\textit{only last statement may be send in Statements}';\ \textit{Flow-initiator}'$

[4] $\textit{check-send-in-fb}(\textit{LSC\_CNT\_FLOW Flow-initiator}'\ \textit{Central-parameter-list}';\ \textit{Statements}'\ \textit{LSC\_CNT\_FLOW\_END};) =$
$\textit{only last statement may be send in Statements}';\ \textit{Flow-initiator}'$

[5] $\textit{is-send-stm}(\textit{LSC\_SEND Identifier}' : \textit{Telegram-output-type}';) = \textit{true}$

[6] $\textit{is-send-stm}(\textit{LSC\_CNT\_SEND Identifier}' : \textit{Telegram-output-type}'\ \textit{Central-parameter-list}';) = \textit{true}$

[7] $$\frac{\textit{is-send-stm}(\textit{Statement}') = \textit{true}}{\textit{contains-send-stmt}(\textit{Statements}'\ \textit{Statement}'\ \textit{Statements}'') = \textit{true}}$$

[8] $$\frac{\textit{contains-send-stmt}(\textit{Statements}'\ \textit{Statement}') \neq \textit{true}}{\textit{only last statement may be send in Statements}'\ \textit{Statement}';\ \textit{Flow-initiator}' =}$$

[9] $$\frac{\textit{is-send-stm}(\textit{Statement}') = \textit{true},\quad \textit{contains-send-stmt}(\textit{Statements}') \neq \textit{true}}{\textit{only last statement may be send in Statements}'\ \textit{Statement}';\ \textit{Flow-initiator}' =}$$

[10] $$\frac{\textit{is-send-stm}(\textit{Statement}') \neq \textit{true},\quad \textit{contains-send-stmt}(\textit{Statements}') = \textit{true}}{\textit{only last statement may be send in Statements}'\ \textit{Statement}';\ \textit{Flow-initiator}' = \textit{flow block Flow-initiator}'\ \textit{contains send statement which is not the last statement}}$$

[11] $$\frac{\textit{is-send-stm}(\textit{Statement}') = \textit{true},\quad \textit{contains-send-stmt}(\textit{Statements}') = \textit{true}}{\textit{only last statement may be send in Statements}'\ \textit{Statement}';\ \textit{Flow-initiator}' = \textit{flow block Flow-initiator}'\ \textit{contains send statement which is not the last statement}}$$

# F Check-identifiers

**exist** -functions report errors when the identifier to be looked for is not known to the Type Environment, i.e., they are undeclared.

**imports** LSC-Type-environment(P)
**exports**
**context-free syntax**

*exist-telegram*(TELENV, Identifier) → ERROR-LIST
*exist-telegram-data*(Telegram-data-name,
Telegram-data-declarations) → ERROR-LIST
*exist-par*(Identifier, Par-declarations) → ERROR-LIST
*exist-var*(Identifier, Var-declarations) → ERROR-LIST
*exist-par-var*(TENV, Identifier) → ERROR-LIST

*undeclared* "*Identifier*" Identifier → ERROR

"*Telegram-data*" Telegram-data-name *yet*
undefined in telegram-side-file → ERROR

"*Identifier*" Identifier *yet*
undefined in telegram-side-file → ERROR

**equations**

*exist-telegram*([*IdealTelegramDeclarations′*
*Identifier′* : *Telegram-type′*;
*IdealTelegramDeclarations″*],
[1] *Identifier′*) =

[2] *exist-telegram*(*Telenv′*, *Identifier′*) =
*Identifier Identifier′ yet undefined in telegram-side-file* **otherwise**

*exist-par*(*Identifier′*,
*Par-declarations′*
*Identifier′* : *Par-type′*;
[3] *Par-declarations″*) =

*exist-par*(*Identifier′*,
*Par-declarations′*
*Identifier′*(*Dir-list′*) : *Par-type′*;
[4] *Par-declarations″*) =

*exist-par*(*Identifier′*,
[5] *Par-declarations′*) =
*undeclared Identifier Identifier′* **otherwise**

*exist-var*(*Identifier′*,
*Var-declarations′*
*Identifier′* : *Par-var-type′*;
[6] *Var-declarations″*) =

$$\mathit{exist\text{-}var}(\text{Identifier}', \quad \mathit{Var\text{-}declarations}' \;\; \mathit{Identifier}' \; \mathit{Dir\text{-}ini\text{-}list}' : \mathit{Par\text{-}var\text{-}type}'; \;\; \mathit{Var\text{-}declarations}'') = \qquad [7]$$

$$\mathit{exist\text{-}var}(\text{Identifier}', \quad \mathit{Var\text{-}declarations}') = \qquad [8]$$
$$\mathit{undeclared\ Identifier}\ \text{Identifier}' \quad \textbf{otherwise}$$

$$\mathit{exist\text{-}telegram\text{-}data}(\mathit{Telegram\text{-}data\text{-}name}', \quad \mathit{Telegram\text{-}data\text{-}declarations}' \;\; \mathit{Telegram\text{-}data\text{-}name}' : \mathit{Range}'; \;\; \mathit{Telegram\text{-}data\text{-}declarations}'') = \qquad [9]$$

$$\mathit{exist\text{-}telegram\text{-}data}(\mathit{Telegram\text{-}data\text{-}name}', \quad \mathit{Telegram\text{-}data\text{-}declarations}') = \qquad [10]$$
$$\mathit{Telegram\text{-}data}\ \mathit{Telegram\text{-}data\text{-}name}'\ \mathit{yet\ undefined\ in\ telegram\text{-}side\text{-}file} \quad \textbf{otherwise}$$

$$[11] \quad \frac{\mathit{lookup\text{-}par}(\text{Identifier}', \mathit{par\text{-}declarations}(\mathit{Tenv}')) = \mathit{Undefined\text{-}par\text{-}type}}{\mathit{exist\text{-}par\text{-}var}(\mathit{Tenv}', \text{Identifier}') = \mathit{exist\text{-}var}(\text{Identifier}', \mathit{var\text{-}declarations}(\mathit{Tenv}'))}$$

$$[12] \quad \mathit{exist\text{-}par\text{-}var}(\mathit{Tenv}', \text{Identifier}') = \mathit{exist\text{-}par}(\text{Identifier}', \mathit{par\text{-}declarations}(\mathit{Tenv}')) \quad \textbf{otherwise}$$

# G Check-labels

**check-labels** checks whether all merges corresponds with continue at statements. Merge labels must be unique. There may not be merge labels with no continue at label referring to ( dangling merges ). There may not be continue at statements without a merge block with the same label ( jump to nowhere ).

**imports** LSC-mappings(Q) Errors(J)
**exports**
  **sorts** Label-env Labels
  **context-free syntax**

| | |
|---|---|
| *check-labels*(Ideal-program) | → ERROR-LIST |
| *check-labels*(Function-blocks) | → ERROR-LIST |
| *check-labels*(Merge-blocks, Flow-blocks) | → ERROR-LIST |
| *make-merge-and-continue*(Function-block) | → Label-env |
| "*continue-labels*:" Labels "*&*" "*merge-labels*:" Labels | → Label-env |
| *get-fb-continues*(Function-block) | → Labels |
| *get-fb-merges*(Function-block) | → Labels |
| *get-stmts-continues*(Statements) | → Labels |
| *get-stmt-continue*(Statement) | → Labels |

*get-stmts-merges*(Statements) → Labels
*get-mbs-merges*(Merge-blocks) → Labels
*get-mbs-continues*(Merge-blocks) → Labels
*get-flbs-continues*(Flow-blocks) → Labels
*get-mb-merges*(Merge-block) → Labels
*get-mb-continues*(Merge-block) → Labels
*get-flb-continues*(Flow-block) → Labels

{Label “,”}* → Labels
*join*(Labels, Labels) → Labels

*check-merge-and-continue* “(” Label-env “;” Identifier “)” → ERROR-LIST
*number-in-number-list*(Label, Labels) → BOOL

*all-numbers-in* Labels *should be in* Labels “;” Identifier → ERROR-LIST

*function block* Identifier *refers to non-existing*
function merge block “:” Label → ERROR

*all-merge-block-labels-must-be-unique*
“(” Labels “;” Identifier “)” → ERROR-LIST
*function block* Identifier *contains*
function merge blocks with duplicate label “:” Label → ERROR

*every number in* Labels *must be in* Labels “;” Merge-block → ERROR-LIST
*every number in* Labels *must be in* Labels “;” Flow-block → ERROR-LIST

*merge block labelled* Label *refers to*
non-existing merge block Label → ERROR
*flow block labelled* Flow-initiator *refers to*
non-existing merge block Label → ERROR

*check-labels-in-merge-blocks*
“(” “*merge-labels*:” Labels “,” Merge-blocks “)” → ERROR-LIST
*check-labels-in-merge-block*
“(” “*merge-labels*:” Labels “,” Merge-block “)” → ERROR-LIST
*check-jumps-from-flbs*
“(” “*merge-labels*:” Labels “,” Flow-blocks “)” → ERROR-LIST
*check-jumps-from-flb*
“(” “*merge-labels*:” Labels “,” Flow-blocks “)” → ERROR-LIST

*all-merge-block-labels-must-be-unique*(Labels) → ERROR-LIST
*multiple defined merge block* “*label*:” Label → ERROR

**equations**

[1] *check-labels*(*Ideal-program′*) =
*join-errors*(*check-labels*(*function-blocks*(*Ideal-program′*)),
*check-labels*(*merge-blocks*(*Ideal-program′*),
*flow-blocks*(*Ideal-program′*)))

[2] *check-labels*(*Function-block′ Function-blocks′*) =
*join-errors*(*check-merge-and-continue*(*make-merge-and-continue*(*Function-block′*);
*name*(*Function-block′*)),
*check-labels*(*Function-blocks′*))

[3] *check-labels*() =

[4] *make-merge-and-continue*(Function-block$'$) =
*continue-labels*: *get-fb-continues*(Function-block$'$)
*& merge-labels*: *get-fb-merges*(Function-block$'$)

[5] *get-fb-continues*(*LSC_FUNCTION* Identifier$'$;
Statements$'$
Merge-blocks$'$ *LSC_FUNCTION_END*;) =
*join*(*get-stmts-continues*(Statements$'$),
*get-mbs-continues*(Merge-blocks$'$))

[6] *get-mbs-continues*(Merge-block$'$ Merge-blocks$'$) =
*join*(*get-mb-continues*(Merge-block$'$),
*get-mbs-continues*(Merge-blocks$'$))

[7] *get-mbs-continues*() =

[8] *get-stmts-continues*(Statement$'$ Statements$'$) =
*join*(*get-stmt-continue*(Statement$'$),
*get-stmts-continues*(Statements$'$))

[9] *get-stmts-continues*() =

[10] *get-stmt-continue*(*LSC_SET* Set-expression$'$;) =

[11] *get-stmt-continue*(*LSC_SPLIT* Test-expression$'$;
Statements$'$ *LSC_SPLIT_END*;) =
*get-stmts-continues*(Statements$'$)

[12] *get-stmt-continue*(*LSC_TEST* Test-expression$'$;) =

[13] *get-stmt-continue*(*LSC_SEND* Identifier$'$ : Telegram-output-type$'$;) =

[14] *get-stmt-continue*(*LSC_CNT_SEND* Identifier$'$
: Telegram-output-type$'$ Central-parameter-list$'$;) =

[15] *get-stmt-continue*(*LSC_CONT_AT LABEL_* Label$'$;) = *Label*$'$

[16] *get-fb-merges*(*LSC_FUNCTION* Identifier$'$;
Statements$'$
Merge-blocks$'$ *LSC_FUNCTION_END*;) =
*join*(*get-stmts-merges*(Statements$'$),
*get-mbs-merges*(Merge-blocks$'$))

[17] $\textit{get-stmts-merges}(\text{Statements}') \; = $

[18] $$\frac{\text{Labels}''' = \text{Labels}', \text{Labels}''}{\textit{join}(\text{Labels}', \text{Labels}'') = \text{Labels}'''}$$

[19] $$\begin{array}{l} \textit{check-merge-and-continue}(\textit{continue-labels}\colon \text{Labels}' \;\&\; \textit{merge-labels}\colon \text{Labels}''; \\ \qquad\qquad\qquad\qquad \text{Identifier}') \qquad\qquad\qquad\qquad = \\ \textit{join-errors}(\textit{all-merge-block-labels-must-be-unique}(\text{Labels}''; \text{Identifier}'), \\ \qquad\qquad \textit{all-numbers-in}\ \text{Labels}'\ \textit{should be in}\ \text{Labels}''; \text{Identifier}') \end{array}$$

[20] $\textit{number-in-number-list}(\text{Label}', \text{Labels}', \text{Label}', \text{Labels}'') \; = \; \textit{true}$

[21] $$\frac{\textit{number-in-number-list}(\text{Label}', \text{Labels}'') = \textit{true}}{\begin{array}{l}\textit{all-numbers-in}\ \text{Label}', \text{Labels}'\ \textit{should be in}\ \text{Labels}''; \text{Identifier}' = \\ \textit{all-numbers-in}\ \text{Labels}'\ \textit{should be in}\ \text{Labels}''; \text{Identifier}'\end{array}}$$

[22] $$\frac{\textit{number-in-number-list}(\text{Label}', \text{Labels}'') \neq \textit{true}}{\begin{array}{l}\textit{all-numbers-in}\ \text{Label}', \text{Labels}'\ \textit{should be in}\ \text{Labels}''; \text{Identifier}' = \\ \textit{join-errors}(\textit{function block}\ \text{Identifier}' \\ \qquad\qquad \textit{refers to non-existing function merge block} : \text{Label}', \\ \qquad\qquad \textit{all-numbers-in}\ \text{Labels}'\ \textit{should be in}\ \text{Labels}''; \text{Identifier}')\end{array}}$$

[23] $\textit{all-numbers-in}\ \ \textit{should be in}\ \text{Labels}''; \text{Identifier}' \; =$

[24] $$\frac{\textit{number-in-number-list}(\text{Label}', \text{Labels}') = \textit{true}}{\begin{array}{l}\textit{all-merge-block-labels-must-be-unique}(\text{Label}', \text{Labels}'; \text{Identifier}') = \\ \textit{join-errors}(\textit{function block}\ \text{Identifier}' \\ \qquad\qquad \textit{contains function merge blocks with duplicate label} : \text{Label}', \\ \qquad\qquad \textit{all-merge-block-labels-must-be-unique}(\text{Labels}'; \text{Identifier}'))\end{array}}$$

[25] $$\frac{\textit{number-in-number-list}(\text{Label}', \text{Labels}') \neq \textit{true}}{\begin{array}{l}\textit{all-merge-block-labels-must-be-unique}(\text{Label}', \text{Labels}'; \text{Identifier}') = \\ \textit{all-merge-block-labels-must-be-unique}(\text{Labels}'; \text{Identifier}')\end{array}}$$

[26] $\textit{all-merge-block-labels-must-be-unique}(; \text{Identifier}') \; =$

[27] $$\frac{\text{Labels}' = \textit{get-mbs-merges}(\text{Merge-blocks}')}{\begin{array}{l}\textit{check-labels}(\text{Merge-blocks}', \text{Flow-blocks}') = \\ \textit{join-errors}(\textit{all-merge-block-labels-must-be-unique}(\text{Labels}'), \\ \qquad\qquad \textit{join-errors}(\textit{check-labels-in-merge-blocks}(\textit{merge-labels}\colon \text{Labels}', \text{Merge-blocks}'), \\ \qquad\qquad\qquad\qquad \textit{check-jumps-from-flbs}(\textit{merge-labels}\colon \text{Labels}', \text{Flow-blocks}')))\end{array}}$$

[28] $$\begin{array}{l}\textit{get-mbs-merges}(\text{Merge-block}'\ \text{Merge-blocks}') = \\ \textit{join}(\textit{get-mb-merges}(\text{Merge-block}'), \\ \qquad \textit{get-mbs-merges}(\text{Merge-blocks}'))\end{array}$$

[29] $\mathit{get\text{-}mbs\text{-}merges}() \ =$

[30]
$$\begin{array}{l}\mathit{get\text{-}mb\text{-}merges}(\mathit{LSC\_MERGE\_LABEL\_}\ \text{Label}';\\ \qquad\qquad\qquad\qquad\qquad \text{Statements}'\ \mathit{LSC\_MERGE\_END};) =\\ \text{Label}'\end{array}$$

[31]
$$\begin{array}{l}\mathit{get\text{-}mb\text{-}continues}(\mathit{LSC\_MERGE\_LABEL\_}\ \text{Label}';\\ \qquad\qquad\qquad\qquad\qquad \text{Statements}'\ \mathit{LSC\_MERGE\_END};) =\\ \mathit{get\text{-}stmts\text{-}continues}(\text{Statements}')\end{array}$$

[32]
$$\begin{array}{l}\mathit{get\text{-}flb\text{-}continues}(\mathit{LSC\_FLOW}\ \text{Flow-initiator}';\\ \qquad\qquad\qquad \text{Statements}'\ \mathit{LSC\_FLOW\_END};) =\\ \mathit{get\text{-}stmts\text{-}continues}(\text{Statements}')\end{array}$$

[33]
$$\begin{array}{l}\mathit{get\text{-}flb\text{-}continues}(\mathit{LSC\_CNT\_FLOW}\ \text{Flow-initiator}'\\ \qquad\qquad\qquad\qquad \text{Central-parameter-list}';\\ \qquad\qquad\qquad\qquad \text{Statements}'\ \mathit{LSC\_CNT\_FLOW\_END};) =\\ \mathit{get\text{-}stmts\text{-}continues}(\text{Statements}')\end{array}$$

(almost) top-level checks

[34]
$$\begin{array}{l}\mathit{check\text{-}labels\text{-}in\text{-}merge\text{-}blocks}(\mathit{merge\text{-}labels}\text{: Labels}',\ \text{Merge-block}'\ \text{Merge-blocks}') =\\ \mathit{join\text{-}errors}(\mathit{check\text{-}labels\text{-}in\text{-}merge\text{-}block}(\mathit{merge\text{-}labels}\text{: Labels}',\ \text{Merge-block}'),\\ \qquad \mathit{check\text{-}labels\text{-}in\text{-}merge\text{-}blocks}(\mathit{merge\text{-}labels}\text{: Labels}',\ \text{Merge-blocks}'))\end{array}$$

[35] $\mathit{check\text{-}labels\text{-}in\text{-}merge\text{-}blocks}(\mathit{merge\text{-}labels}\text{: Labels}',\ ) \ =$

[36]
$$\frac{\text{Labels}'' = \mathit{get\text{-}mb\text{-}continues}(\text{Merge-block}')}{\begin{array}{l}\mathit{check\text{-}labels\text{-}in\text{-}merge\text{-}block}(\mathit{merge\text{-}labels}\text{: Labels}',\ \text{Merge-block}') =\\ \mathit{every\ number\ in}\ \text{Labels}''\ \mathit{must\ be\ in}\ \text{Labels}';\ \text{Merge-block}'\end{array}}$$

[37]
$$\begin{array}{l}\mathit{check\text{-}jumps\text{-}from\text{-}flbs}(\mathit{merge\text{-}labels}\text{: Labels}',\ \text{Flow-block}'\ \text{Flow-blocks}') =\\ \mathit{join\text{-}errors}(\mathit{check\text{-}jumps\text{-}from\text{-}flb}(\mathit{merge\text{-}labels}\text{: Labels}',\ \text{Flow-block}'),\\ \qquad \mathit{check\text{-}jumps\text{-}from\text{-}flbs}(\mathit{merge\text{-}labels}\text{: Labels}',\ \text{Flow-blocks}'))\end{array}$$

[38] $\mathit{check\text{-}jumps\text{-}from\text{-}flbs}(\mathit{merge\text{-}labels}\text{: Labels}',\ ) \ =$

[39]
$$\frac{\text{Labels}'' = \mathit{get\text{-}flb\text{-}continues}(\text{Flow-block}')}{\begin{array}{l}\mathit{check\text{-}jumps\text{-}from\text{-}flb}(\mathit{merge\text{-}labels}\text{: Labels}',\ \text{Flow-block}') =\\ \mathit{every\ number\ in}\ \text{Labels}''\ \mathit{must\ be\ in}\ \text{Labels}';\ \text{Flow-block}'\end{array}}$$

[40]
$$\frac{\mathit{number\text{-}in\text{-}number\text{-}list}(\text{Label}',\ \text{Labels}') = \mathit{true}}{\begin{array}{l}\mathit{every\ number\ in}\ \text{Label}',\ \text{Labels}''\ \mathit{must\ be\ in}\ \text{Labels}';\ \text{Merge-block}' =\\ \mathit{every\ number\ in}\ \text{Labels}''\ \mathit{must\ be\ in}\ \text{Labels}';\ \text{Merge-block}'\end{array}}$$

[41]
$$\frac{\mathit{number\text{-}in\text{-}number\text{-}list}(\text{Label}',\ \text{Labels}') = \mathit{true}}{\begin{array}{l}\mathit{every\ number\ in}\ \text{Label}',\ \text{Labels}''\ \mathit{must\ be\ in}\ \text{Labels}';\ \text{Flow-block}' =\\ \mathit{every\ number\ in}\ \text{Labels}''\ \mathit{must\ be\ in}\ \text{Labels}';\ \text{Flow-block}'\end{array}}$$

[42] $$\frac{\textit{number-in-number-list}(\text{Label}', \text{Labels}') \neq \textit{true}}{\begin{array}{l}\textit{every number in } \text{Label}', \text{Labels}'' \textit{ must be in } \text{Labels}'; \textit{Merge-block}' = \\ \textit{join-errors}(\textit{merge block labelled number}(\text{Merge-block}') \\ \qquad\qquad \textit{refers to non-existing merge block } \text{Label}', \\ \qquad\qquad \textit{every number in } \text{Labels}'' \textit{ must be in } \text{Labels}'; \textit{Merge-block}')\end{array}}$$

[43] $$\frac{\textit{number-in-number-list}(\text{Label}', \text{Labels}') \neq \textit{true}}{\begin{array}{l}\textit{every number in } \text{Label}', \text{Labels}'' \textit{ must be in } \text{Labels}'; \textit{Flow-block}' = \\ \textit{join-errors}(\textit{flow block labelled initiator}(\text{Flow-block}') \\ \qquad\qquad \textit{refers to non-existing merge block } \text{Label}', \\ \qquad\qquad \textit{every number in } \text{Labels}'' \textit{ must be in } \text{Labels}'; \textit{Flow-block}')\end{array}}$$

[44] $\textit{every number in } \ \textit{must be in } \text{Labels}'; \textit{Merge-block}' \ =$

[45] $\textit{every number in } \ \textit{must be in } \text{Labels}'; \textit{Flow-block}' \ =$

[46] $$\frac{\textit{number-in-number-list}(\text{Label}', \text{Labels}') = \textit{true}}{\begin{array}{l}\textit{all-merge-block-labels-must-be-unique}(\text{Label}', \text{Labels}') = \\ \textit{join-errors}(\textit{multiple defined merge block label}\text{: Label}', \\ \qquad\qquad \textit{all-merge-block-labels-must-be-unique}(\text{Labels}'))\end{array}}$$

[47] $$\frac{\textit{number-in-number-list}(\text{Label}', \text{Labels}') \neq \textit{true}}{\begin{array}{l}\textit{all-merge-block-labels-must-be-unique}(\text{Label}', \text{Labels}') = \\ \textit{all-merge-block-labels-must-be-unique}(\text{Labels}')\end{array}}$$

[48] $\textit{all-merge-block-labels-must-be-unique}() \ =$

# H Check-qualifiers

**check-qualifiers** performs additional checks on qualifiers conform the Appendices of the Euris-Simulation Tutorial [Klap94]. Most of requirements of the table has been already been solved syntactically by the IDEAL language designers. The direction checks here are performed on directions in the declaration section and test-expressions. See the module Check-directions for other direction checks.

**imports** LSC-Type-environment$^{(P)}$
**exports**
**context-free syntax**

| | |
|---|---|
| *check-qualifiers*(TENV) | → ERROR-LIST |
| *check-par-qfs*(Par-declarations) | → ERROR-LIST |
| *check-par-qf*(Par-declaration) | → ERROR-LIST |
| *check-qf-vars*(Var-declarations) | → ERROR-LIST |
| *check-qf-var*(Var-declaration) | → ERROR-LIST |
| *check-Dir-list*({Directions ","}+, Par-declaration) | → ERROR-LIST |
| *check-Dir-list*({Directions ","}+, Var-declaration) | → ERROR-LIST |
| *check-Dir-list*({Directions ","}+) | → ERROR-LIST |
| *check-Direction*(Direction, {Directions ","}+) | → ERROR-LIST |

$\textit{check-dirs-in-Dir-list}(\text{Directions, \{Directions ``,''\}+}) \rightarrow \text{ERROR-LIST}$

$\textit{check-Directions}(\text{Directions}) \rightarrow \text{ERROR-LIST}$

$\textit{number-of-directions-in-list}(\text{\{Directions ``,''\}+}) \rightarrow \text{INT}$
$\textit{count-directions}(\text{Directions}) \rightarrow \text{INT}$

*too many directions* "(" Dir-list ")" *after* Identifier
of type Par-var-type in par-declaration Par-declaration → ERROR

*too many directions* "(" Dir-list ")" *after* Identifier
of type Par-var-type in var-declaration Var-declaration → ERROR

*duplicate "Direction(s)" in direction list* → ERROR

"*Illegal␣Directions*" Directions → ERROR
*duplicate "Direction(s)" in direction list*
in var-declaration Var-declaration → ERROR

*duplicate "Direction(s)" in direction list*
in par-declaration Par-declaration → ERROR

**equations**

[1] $\textit{check-qualifiers}(\textit{Tenv}') =$
$\textit{join-errors}(\textit{check-par-qfs}(\textit{par-declarations}(\textit{Tenv}')),$
$\textit{check-qf-vars}(\textit{var-declarations}(\textit{Tenv}')))$

[2] $\textit{check-par-qfs}() \; =$

[3] $\textit{check-par-qfs}(\textit{Par-declaration}' \; \textit{Par-declarations}') =$
$\textit{join-errors}(\textit{check-par-qf}(\textit{Par-declaration}'),$
$\textit{check-par-qfs}(\textit{Par-declarations}'))$

[4] $\textit{check-qf-vars}() \; =$

[5] $\textit{check-qf-vars}(\textit{Var-declaration}' \; \textit{Var-declarations}') =$
$\textit{join-errors}(\textit{check-qf-var}(\textit{Var-declaration}'),$
$\textit{check-qf-vars}(\textit{Var-declarations}'))$

[6] $\textit{check-par-qf}(\textit{Identifier}' : \textit{Par-type}';) \; =$

[7]
$$\frac{\textit{number-of-directions-in-list}(\textit{Dir-list}') = 1}{\begin{array}{l}\textit{check-par-qf}(\textit{Identifier}'(\textit{Dir-list}') \\ \qquad : \textit{Par-type}';) = \\ \textit{check-Dir-list}(\textit{Dir-list}', \textit{Identifier}'(\textit{Dir-list}') \\ \qquad : \textit{Par-type}';)\end{array}}$$

[8]
$$\frac{\begin{array}{c}\textit{value-in-range}(\textit{number-of-directions-in-list}(\text{Dir-list}'),\ [2,\ 4]) = \textit{true},\\ \text{Par-type}' = \textit{BOOL\_LIST\_TYPE}\end{array}}{\begin{array}{l}\textit{check-par-qf}(\text{Identifier}'(\text{Dir-list}')\\ \qquad\qquad : \text{Par-type}';) =\\ \textit{join-errors}(\textit{too many directions}(\text{Dir-list}')\ \textit{after}\ \text{Identifier}'\\ \qquad \textit{of type}\ \text{Par-type}'\\ \qquad \textit{in par-declaration}\ \text{Identifier}'(\text{Dir-list}') : \text{Par-type}'; ,\\ \qquad \textit{check-Dir-list}(\text{Dir-list}',\\ \qquad\qquad \text{Identifier}'(\text{Dir-list}') : \text{Par-type}';))\end{array}}$$

[9]
$$\frac{\begin{array}{c}\textit{value-in-range}(\textit{number-of-directions-in-list}(\text{Dir-list}'),\ [2,\ 4]) = \textit{true},\\ \text{Par-type}' = \textit{INT\_TYPE}\end{array}}{\begin{array}{l}\textit{check-par-qf}(\text{Identifier}'(\text{Dir-list}') : \text{Par-type}';) =\\ \textit{join-errors}(\textit{too many directions}(\text{Dir-list}')\ \textit{after}\ \text{Identifier}'\\ \qquad \textit{of type}\ \text{Par-type}'\\ \qquad \textit{in par-declaration}\ \text{Identifier}'(\text{Dir-list}') : \text{Par-type}'; ,\\ \qquad \textit{check-Dir-list}(\text{Dir-list}',\\ \qquad\qquad \text{Identifier}'(\text{Dir-list}') : \text{Par-type}';))\end{array}}$$

[10]
$$\begin{array}{ll}\textit{check-par-qf}(\text{Identifier}'(\text{Dir-list}') : \text{Par-type}';) = & \\ \quad \textit{check-Dir-list}(\text{Dir-list}', & \textbf{otherwise}\\ \qquad\qquad \text{Identifier}'(\text{Dir-list}') : \text{Par-type}';) & \end{array}$$

[11]
$$\textit{check-qf-var}(\text{Identifier}' : \text{Par-var-type}';)\ =$$

[12]
$$\frac{\begin{array}{c}\textit{value-in-range}(\textit{number-of-directions-in-list}(\text{Dir-list}'),\ [2,\ 4]) = \textit{true},\\ \text{Par-var-type}' = \textit{INT\_TYPE}\end{array}}{\begin{array}{l}\textit{check-qf-var}(\text{Identifier}'\ (\text{Dir-list}', \text{Ini-list}')\\ \qquad\qquad : \text{Par-var-type}';) \qquad =\\ \textit{join-errors}(\textit{too many directions}(\text{Dir-list}')\ \textit{after}\ \text{Identifier}'\\ \qquad \textit{of type}\ \text{Par-var-type}'\\ \qquad \textit{in var-declaration}\ \text{Identifier}'\ (\text{Dir-list}', \text{Ini-list}')\\ \qquad\qquad\qquad : \text{Par-var-type}'; ,\\ \qquad \textit{check-Dir-list}(\text{Dir-list}',\\ \qquad\qquad \text{Identifier}'\ (\text{Dir-list}', \text{Ini-list}')\\ \qquad\qquad : \text{Par-var-type}';))\end{array}}$$

[13]
$$\frac{\begin{array}{c}\textit{value-in-range}(\textit{number-of-directions-in-list}(\text{Dir-list}'),\ [2,\ 4]) = \textit{true},\\ \text{Par-var-type}' = \textit{BOOL\_LIST\_TYPE}\end{array}}{\begin{array}{l}\textit{check-qf-var}(\text{Identifier}'\ (\text{Dir-list}', \text{Ini-list}')\\ \qquad\qquad : \text{Par-var-type}';) \qquad =\\ \textit{join-errors}(\textit{too many directions}(\text{Dir-list}')\ \textit{after}\ \text{Identifier}'\\ \qquad \textit{of type}\ \text{Par-var-type}'\\ \qquad \textit{in var-declaration}\ \text{Identifier}'\ (\text{Dir-list}', \text{Ini-list}')\\ \qquad\qquad\qquad : \text{Par-var-type}'; ,\\ \qquad \textit{check-Dir-list}(\text{Dir-list}',\\ \qquad\qquad \text{Identifier}'\ (\text{Dir-list}', \text{Ini-list}')\\ \qquad\qquad : \text{Par-var-type}';))\end{array}}$$

$$[14]\quad \frac{\begin{array}{c}\mathit{value\text{-}in\text{-}range}(\mathit{number\text{-}of\text{-}directions\text{-}in\text{-}list}(\text{Dir-list}'), [2, 4]) = \mathit{true},\\ \text{Par-var-type}' = \mathit{INT\_CNT\_TYPE}\ \text{Central-parameter-list}'\end{array}}{\begin{array}{l}\mathit{check\text{-}qf\text{-}var}(\text{Identifier}'\ (\text{Dir-list}', \text{Ini-list}')\\ \qquad\qquad : \text{Par-var-type}';) \qquad\qquad =\\ \mathit{join\text{-}errors}(\mathit{too\ many\ directions}(\text{Dir-list}')\ \mathit{after}\ \text{Identifier}'\\ \qquad \mathit{of\ type}\ \text{Par-var-type}'\\ \qquad \mathit{in\ var\text{-}declaration}\ \text{Identifier}'\ (\text{Dir-list}', \text{Ini-list}')\\ \qquad\qquad\qquad : \text{Par-var-type}';\ ,\\ \qquad \mathit{check\text{-}Dir\text{-}list}(\text{Dir-list}', \text{Identifier}'\ (\text{Dir-list}',\\ \qquad\qquad\qquad\qquad\qquad \text{Ini-list}')\\ \qquad\qquad\qquad\qquad : \text{Par-var-type}';))\end{array}}$$

$$[15]\quad \begin{array}{l}\mathit{check\text{-}qf\text{-}var}(\text{Identifier}'\ (\text{Ini-list}')\\ \qquad\qquad : \text{Par-var-type}';) \quad =\end{array}$$

$$[16]\quad \begin{array}{l}\mathit{check\text{-}qf\text{-}var}(\text{Identifier}'\ (\text{Dir-list}', \text{Ini-list}')\\ \qquad\qquad : \text{Par-var-type}';) \qquad\qquad =\\ \mathit{check\text{-}Dir\text{-}list}(\text{Dir-list}', \qquad\qquad\qquad\qquad \textbf{otherwise}\\ \qquad\qquad \text{Identifier}'\ (\text{Dir-list}', \text{Ini-list}')\\ \qquad\qquad : \text{Par-var-type}';)\end{array}$$

$$[17]\quad \mathit{number\text{-}of\text{-}directions\text{-}in\text{-}list}(\text{Directions}') = \mathit{count\text{-}directions}(\text{Directions}')$$

$$[18]\quad \begin{array}{l}\mathit{number\text{-}of\text{-}directions\text{-}in\text{-}list}(\text{Directions}', \text{Dir-list}') =\\ \mathit{count\text{-}directions}(\text{Directions}')\\ \_{+}\ \mathit{number\text{-}of\text{-}directions\text{-}in\text{-}list}(\text{Dir-list}')\end{array}$$

$$[19]\quad \mathit{count\text{-}directions}(\text{Direction}') = 1$$

$$[20]\quad \begin{array}{l}\mathit{count\text{-}directions}(\text{Direction}'\ \text{Directions}') =\\ \mathit{count\text{-}directions}(\text{Direction}')\ \_{+}\ \mathit{count\text{-}directions}(\text{Directions}')\end{array}$$

$$[21]\quad \frac{\begin{array}{c}\mathit{check\text{-}Dir\text{-}list}(\text{Dir-list}', \text{Par-declaration}') =\ ,\\ \mathit{check\text{-}dirs\text{-}in\text{-}Dir\text{-}list}(\text{Directions}', \text{Dir-list}') =\end{array}}{\begin{array}{l}\mathit{check\text{-}Dir\text{-}list}(\text{Directions}', \text{Dir-list}', \text{Par-declaration}') =\\ \mathit{join\text{-}errors}(\mathit{check\text{-}dirs\text{-}in\text{-}Dir\text{-}list}(\text{Directions}', \text{Dir-list}'),\\ \qquad\qquad \mathit{check\text{-}Dir\text{-}list}(\text{Dir-list}', \text{Par-declaration}'))\end{array}}$$

$$[22]\quad \begin{array}{l}\mathit{check\text{-}Dir\text{-}list}(\text{Directions}', \text{Dir-list}', \text{Par-declaration}') =\\ \mathit{join\text{-}errors}(\mathit{duplicate\ Direction(s)\ in\ direction\ list\ in\ par\text{-}declaration}\ \text{Par-declaration}', \quad \textbf{otherwise}\\ \qquad\qquad \mathit{join\text{-}errors}(\mathit{check\text{-}dirs\text{-}in\text{-}Dir\text{-}list}(\text{Directions}', \text{Dir-list}'),\\ \qquad\qquad\qquad\qquad \mathit{check\text{-}Dir\text{-}list}(\text{Dir-list}', \text{Par-declaration}')))\end{array}$$

$$[23]\quad \frac{\begin{array}{c}\mathit{check\text{-}Dir\text{-}list}(\text{Dir-list}', \text{Var-declaration}') =\ ,\\ \mathit{check\text{-}dirs\text{-}in\text{-}Dir\text{-}list}(\text{Directions}', \text{Dir-list}') =\end{array}}{\begin{array}{l}\mathit{check\text{-}Dir\text{-}list}(\text{Directions}', \text{Dir-list}', \text{Var-declaration}') =\\ \mathit{join\text{-}errors}(\mathit{check\text{-}dirs\text{-}in\text{-}Dir\text{-}list}(\text{Directions}', \text{Dir-list}'),\\ \qquad\qquad \mathit{check\text{-}Dir\text{-}list}(\text{Dir-list}', \text{Var-declaration}'))\end{array}}$$

[24] $$\begin{array}{l} \textit{check-Dir-list}(\text{Directions}', \text{Dir-list}', \text{Var-declaration}') = \\ \quad \textit{join-errors}(\textit{duplicate Direction}(s) \textit{ in direction list in var-declaration } \text{Var-declaration}', \quad \textbf{otherwise} \\ \qquad\qquad \textit{join-errors}(\textit{check-dirs-in-Dir-list}(\text{Directions}', \text{Dir-list}'), \\ \qquad\qquad\qquad \textit{check-Dir-list}(\text{Dir-list}', \text{Var-declaration}'))) \end{array}$$

[25] $$\frac{\textit{check-Directions}(\text{Directions}') = }{\textit{check-Dir-list}(\text{Directions}', \text{Par-declaration}') = \textit{check-Directions}(\text{Directions}')}$$

[26] $$\frac{\textit{check-Directions}(\text{Directions}') \neq }{\begin{array}{l} \textit{check-Dir-list}(\text{Directions}', \text{Par-declaration}') = \\ \textit{join-errors}(\textit{duplicate Direction}(s) \textit{ in direction list in par-declaration } \text{Par-declaration}', \\ \qquad \textit{check-Directions}(\text{Directions}')) \end{array}}$$

[27] $$\frac{\textit{check-Directions}(\text{Directions}') = }{\begin{array}{l} \textit{check-Dir-list}(\text{Directions}', \text{Var-declaration}') = \\ \textit{check-Directions}(\text{Directions}') \end{array}}$$

[28] $$\frac{\textit{check-Directions}(\text{Directions}') \neq }{\begin{array}{l} \textit{check-Dir-list}(\text{Directions}', \text{Var-declaration}') = \\ \textit{join-errors}(\textit{duplicate Direction}(s) \textit{ in direction list in var-declaration } \text{Var-declaration}', \\ \qquad \textit{check-Directions}(\text{Directions}')) \end{array}}$$

[29] $$\frac{\begin{array}{c} \textit{check-Dir-list}(\text{Dir-list}') = , \\ \textit{check-dirs-in-Dir-list}(\text{Directions}', \text{Dir-list}') = \end{array}}{\begin{array}{l} \textit{check-Dir-list}(\text{Directions}', \text{Dir-list}') = \\ \textit{join-errors}(\textit{check-dirs-in-Dir-list}(\text{Directions}', \text{Dir-list}'), \\ \qquad \textit{check-Dir-list}(\text{Dir-list}')) \end{array}}$$

[30] $$\begin{array}{l} \textit{check-Dir-list}(\text{Directions}', \text{Dir-list}') = \\ \quad \textit{join-errors}(\textit{duplicate Direction}(s) \textit{ in direction list}, \quad \textbf{otherwise} \\ \qquad\qquad \textit{join-errors}(\textit{check-dirs-in-Dir-list}(\text{Directions}', \text{Dir-list}'), \\ \qquad\qquad\qquad \textit{check-Dir-list}(\text{Dir-list}'))) \end{array}$$

[31] $$\frac{\textit{check-Directions}(\text{Directions}') = }{\textit{check-Dir-list}(\text{Directions}') = }$$

[32] $$\frac{\textit{check-Directions}(\text{Directions}') \neq }{\begin{array}{l} \textit{check-Dir-list}(\text{Directions}') = \\ \textit{join-errors}(\textit{duplicate Direction}(s) \textit{ in direction list}, \\ \qquad \textit{check-Directions}(\text{Directions}')) \end{array}}$$

[33] $$\begin{array}{l} \textit{check-dirs-in-Dir-list}(\text{Direction}'\ \text{Directions}', \text{Dir-list}') = \\ \quad \textit{join-errors}(\textit{check-Directions}(\text{Direction}'\ \text{Directions}'), \\ \qquad\qquad \textit{join-errors}(\textit{check-Direction}(\text{Direction}', \text{Dir-list}'), \\ \qquad\qquad\qquad \textit{check-dirs-in-Dir-list}(\text{Directions}', \text{Dir-list}'))) \end{array}$$

[34] $$\begin{array}{l} \textit{check-dirs-in-Dir-list}(\text{Direction}', \text{Dir-list}') = \\ \quad \textit{check-Direction}(\text{Direction}', \text{Dir-list}') \end{array}$$

NB. Number of directions is always within the range [1..4]

[35] $\textit{check-Directions}(\textit{Direction}') \;=$

$$[36]\quad \frac{\textit{Direction}' \neq \textit{Direction}''}{\textit{check-Directions}(\textit{Direction}'\ \textit{Direction}'') =}$$

$$[37]\quad \frac{\begin{array}{c}\textit{Direction}' \neq \textit{Direction}'',\\ \textit{Direction}'' \neq \textit{Direction}''',\\ \textit{Direction}''' \neq \textit{Direction}'\end{array}}{\textit{check-Directions}(\textit{Direction}'\ \textit{Direction}''\ \textit{Direction}''') =}$$

$$[38]\quad \frac{\begin{array}{c}\textit{Direction}' \neq \textit{Direction}'',\\ \textit{Direction}'' \neq \textit{Direction}''',\\ \textit{Direction}''' \neq \textit{Direction}'''',\\ \textit{Direction}'''' \neq \textit{Direction}',\\ \textit{Direction}'''' \neq \textit{Direction}'',\\ \textit{Direction}''' \neq \textit{Direction}'\end{array}}{\begin{array}{l}\textit{check-Directions}(\textit{Direction}'\ \textit{Direction}''\ \textit{Direction}'''\\ \qquad\qquad \textit{Direction}'''') \qquad\qquad =\end{array}}$$

[39] $\textit{check-Directions}(\textit{Directions}') =$
$\quad\textit{Illegal}\text{␣}\textit{Directions}\ \textit{Directions}'$ **otherwise**

[40] $\textit{check-Direction}(\textit{Direction}', \textit{Directions}', \textit{Dir-list}') =$
$\quad\textit{join-errors}(\textit{check-Directions}(\textit{Direction}'\ \textit{Directions}'),$
$\qquad\qquad \textit{check-Direction}(\textit{Direction}', \textit{Dir-list}'))$

[41] $\textit{check-Direction}(\textit{Direction}', \textit{Directions}') =$
$\quad\textit{check-Directions}(\textit{Direction}'\ \textit{Directions}')$

[42] $\textit{check-Direction}(\textit{Direction}', \textit{Directions}', \textit{Dir-list}') =$
$\quad\textit{check-Direction}(\textit{Direction}', \textit{Dir-list}')$ **otherwise**

$$[43]\quad \frac{\begin{array}{c}\textit{value-in-range}(\textit{number-of-directions-in-list}(\textit{Dir-list}'), [1, 4]) = \textit{true},\\ \textit{Par-var-type}' = \textit{INT\_TYPE}\end{array}}{\begin{array}{l}\textit{check-qf-var}(\textit{Identifier}'\ (\textit{Dir-list}')\\ \qquad\qquad : \textit{Par-var-type}';) \quad =\\ \textit{join-errors}(\textit{too many directions}(\textit{Dir-list}')\ \textit{after}\ \textit{Identifier}'\ \textit{of type}\ \textit{Par-var-type}'\\ \qquad\qquad \textit{in var-declaration}\ \textit{Identifier}'\ (\textit{Dir-list}')\\ \qquad\qquad\qquad\qquad : \textit{Par-var-type}'; ,\\ \qquad\qquad \textit{check-Dir-list}(\textit{Dir-list}', \textit{Identifier}'\ (\textit{Dir-list}')\\ \qquad\qquad\qquad\qquad\quad : \textit{Par-var-type}';))\end{array}}$$

$$[44]\quad \frac{\begin{array}{c}\textit{value-in-range}(\textit{number-of-directions-in-list}(\textit{Dir-list}'), [1, 4]) = \textit{true},\\ \textit{Par-var-type}' = \textit{BOOL\_LIST\_TYPE}\end{array}}{\begin{array}{l}\textit{check-qf-var}(\textit{Identifier}'\ (\textit{Dir-list}')\\ \qquad\qquad : \textit{Par-var-type}';) \quad =\\ \textit{join-errors}(\textit{too many directions}(\textit{Dir-list}')\ \textit{after}\ \textit{Identifier}'\ \textit{of type}\ \textit{Par-var-type}'\\ \qquad\qquad \textit{in var-declaration}\ \textit{Identifier}'\ (\textit{Dir-list}')\\ \qquad\qquad\qquad\qquad : \textit{Par-var-type}'; ,\\ \qquad\qquad \textit{check-Dir-list}(\textit{Dir-list}', \textit{Identifier}'\ (\textit{Dir-list}')\\ \qquad\qquad\qquad\qquad\quad : \textit{Par-var-type}';))\end{array}}$$

$$
[45]\quad \frac{\begin{array}{c}\textit{value-in-range}(\textit{number-of-directions-in-list}(\textit{Dir-list}'), [1, 4]) = \textit{true},\\ \textit{Par-var-type}' = \textit{INT\_CNT\_TYPE Central-parameter-list}'\end{array}}{\begin{array}{l}\textit{check-qf-var}(\textit{Identifier}'\ (\textit{Dir-list}')\\ \qquad\qquad : \textit{Par-var-type}';) \quad = \\ \textit{join-errors}(\textit{too many directions}(\textit{Dir-list}')\ \textit{after Identifier}'\ \textit{of type Par-var-type}'\\ \qquad\qquad \textit{in var-declaration Identifier}'\ (\textit{Dir-list}')\\ \qquad\qquad\qquad\qquad\qquad : \textit{Par-var-type}';\ ,\\ \qquad\qquad \textit{check-Dir-list}(\textit{Dir-list}', \textit{Identifier}'\ (\textit{Dir-list}')\\ \qquad\qquad\qquad\qquad\qquad\qquad : \textit{Par-var-type}';))\end{array}}
$$

$$
[46]\quad \begin{array}{l}\textit{check-qf-var}(\textit{Identifier}'\ (\textit{Dir-list}')\\ \qquad\qquad : \textit{Par-var-type}';) \quad = \\ \textit{check-Dir-list}(\textit{Dir-list}', \textit{Identifier}'\ (\textit{Dir-list}') \quad \textbf{otherwise}\\ \qquad\qquad\qquad : \textit{Par-var-type}';)\end{array}
$$

# I Check-types

**tc** -functions performs the 'Classical' typecheck on IDEAL LSC Programs. All the sides files and the Type Environment itself are double checked on duplication of variables. Other checks are performed which are more exhaustively listed in the static semantics section of this document (cf. Section 6).

**imports** Check-identifiers$^{(F)}$ Check-expressions$^{(D)}$ Parameter-side-file$^{(S)}$
Variable-side-file$^{(W)}$ Check-qualifiers$^{(H)}$
**exports**
**context-free syntax**

| | |
|---|---|
| *check-types*(Ideal-program) | → ERROR-LIST |
| *check-declarations*(TENV, Ideal-program) | → ERROR-LIST |
| *check-calls*(TENV, Ideal-program) | → ERROR-LIST |
| *tc-program-start-block*(Program-start-block) | → ERROR-LIST |
| *tc-configuration-block*(Configuration-block) | → ERROR-LIST |
| *tc-type-environment*(TENV) | → ERROR-LIST |
| *tc-vars*(TENV) | → ERROR-LIST |
| *tc-pars*(TENV) | → ERROR-LIST |
| *tc-function-blocks*(TENV, Function-blocks) | → ERROR-LIST |
| *tc-function-block*(TENV, Function-block) | → ERROR-LIST |
| *tc-merge-blocks*(TENV, Merge-blocks) | → ERROR-LIST |
| *tc-merge-block*(TENV, Merge-block) | → ERROR-LIST |
| *tc-flow-blocks*(TENV, Flow-blocks) | → ERROR-LIST |
| *tc-flow-block*(TENV, Flow-block) | → ERROR-LIST |
| *tc-statements*(TENV, Statement*) | → ERROR-LIST |
| *tc-statement*(TENV, Statement) | → ERROR-LIST |
| *tc-set-expression*(TENV, Set-expression) | → ERROR-LIST |
| *check-lhs*(Par-var-exp) | → ERROR-LIST |
| *check-par-unicity*(Par-declarations) | → ERROR-LIST |
| *check-var-unicity*(Var-declarations) | → ERROR-LIST |

$\mathit{check\text{-}var\text{-}central\text{-}types}$(Var-declarations) → ERROR-LIST
$\mathit{check\text{-}var\text{-}central\text{-}type}$(Par-var-type) → ERROR-LIST

$\mathit{check\text{-}normal\text{-}flow}$(TENV, Flow-initiator) → ERROR-LIST
$\mathit{check\text{-}central\text{-}flow}$(TENV, Flow-initiator, Central-parameter-list) → ERROR-LIST
$\mathit{check\text{-}normal\text{-}send}$(TELENV, Identifier, Telegram-in-type) → ERROR-LIST
$\mathit{check\text{-}central\text{-}send}$(TELENV, Identifier, Telegram-in-type,
Central-parameter-list) → ERROR-LIST
$\mathit{check\text{-}parameter\text{-}list}$(Central-parameter-list) → ERROR-LIST

$\mathit{is\text{-}valid\text{-}lhs}$(Par-var-exp) → BOOL

*redeclaration of* Identifier *within*
"*Parameter*" declaration block → ERROR

*redeclaration of* Identifier *within* "*Variable*"
declaration block → ERROR

*incompatible lhs* ":" Telegram-data-name *assigned with*
rhs ":" Simple-expression → ERROR

*incompatible lhs* ":" Par-var-exp *assigned with*
rhs ":" Simple-expression → ERROR

*test expression* Test-expression "*is-not-boolean*" → ERROR

*illegal telegram-output-type* ":" Telegram-in-type → ERROR
*in central telegram send statement for* Identifier → ERROR
*in normal telegram send statement for* Identifier → ERROR

*duplicate identifier-s in* "*Central-parameter-list*"
Central-parameter-list → ERROR

*illegal use of telegram* Identifier *as* Telegram-in-type → ERROR
Telegram-type *expected* → ERROR

"$*\sqcup*\sqcup*\sqcup$ *TYPE-error(s)−in-function-block*$\sqcup$*FUNCTION*" Identifier → ERROR
"$*\sqcup*\sqcup*\sqcup$ *TYPE-error(s)−in-merge-block*$\sqcup$*MERGE*" Label → ERROR
"$*\sqcup*\sqcup*\sqcup$ *TYPE-error(s)−in-flow-block*$\sqcup$*FLOW*" Flow-initiator → ERROR

*incompatible lhs* ":" Telegram-data-name *assigned with*
rhs ":" Test-expression → ERROR
*incompatible lhs* ":" Par-var-exp *assigned with*
rhs ":" Test-expression → ERROR

**equations**

$$[1]\quad \frac{\begin{array}{l}[\mathit{IdealTelegramData},\\ \mathit{parameter\text{-}declarations}(\mathit{parameter\text{-}block}(\text{Ideal-program}')),\\ \mathit{variable\text{-}declarations}(\mathit{variable\text{-}block}(\text{Ideal-program}'))] \quad = \mathit{Tenv}'\end{array}}{\begin{array}{l}\mathit{check\text{-}types}(\text{Ideal-program}') =\\ \mathit{join\text{-}errors}(\mathit{check\text{-}declarations}(\text{Tenv}', \text{Ideal-program}'),\\ \qquad \mathit{check\text{-}calls}(\text{Tenv}', \text{Ideal-program}'))\end{array}}$$

[2] $\mathit{check\text{-}declarations}(\text{Tenv}', \text{Ideal-program}') =$

$$
\begin{aligned}
&\textit{join-errors}(\textit{check-telegram-unicity}(\textit{IdealTelegrams}),\\
&\quad\quad \textit{join-errors}(\textit{tc-program-start-block}(\textit{program-start-block}(\text{Ideal-program}'))),\\
&\quad\quad\quad\quad \textit{join-errors}(\textit{tc-configuration-block}(\textit{configuration-block}(\text{Ideal-program}')),\\
&\quad\quad\quad\quad\quad\quad \textit{tc-type-environment}(\text{Tenv}'))))
\end{aligned}
$$

[3] $\textit{check-calls}(\text{Tenv}', \text{Ideal-program}') =$

$$
\begin{aligned}
&\textit{join-errors}(\textit{tc-function-blocks}(\text{Tenv}', \textit{function-blocks}(\text{Ideal-program}')),\\
&\quad\quad \textit{join-errors}(\textit{tc-merge-blocks}(\text{Tenv}', \textit{merge-blocks}(\text{Ideal-program}')),\\
&\quad\quad\quad\quad \textit{tc-flow-blocks}(\text{Tenv}', \textit{flow-blocks}(\text{Ideal-program}'))))
\end{aligned}
$$

[4] $\textit{check-var-central-types}() =$

$\textit{check-var-central-types}(\text{Identifier}'$
$\quad\quad : \textit{Par-var-type}';$

[5] $\quad\quad \textit{Var-declarations}') =$

$$
\begin{aligned}
&\textit{join-errors}(\textit{check-var-central-type}(\textit{Par-var-type}'),\\
&\quad\quad \textit{check-var-central-types}(\text{Var-declarations}'))
\end{aligned}
$$

[6] $\textit{check-var-central-types}(\text{Identifier}'\ \text{Dir-ini-list}' : \textit{Par-var-type}';\ \text{Var-declarations}') =$

$$
\begin{aligned}
&\textit{join-errors}(\textit{check-var-central-type}(\textit{Par-var-type}'),\\
&\quad\quad \textit{check-var-central-types}(\text{Var-declarations}'))
\end{aligned}
$$

[7] $\textit{check-var-central-type}(\textit{BOOL\_CNT\_TYPE}\ \text{Central-parameter-list}') =$
$\quad \textit{check-parameter-list}(\text{Central-parameter-list}')$

[8] $\textit{check-var-central-type}(\textit{INT\_CNT\_TYPE}\ \text{Central-parameter-list}') =$
$\quad \textit{check-parameter-list}(\text{Central-parameter-list}')$

[9] $\textit{check-var-central-type}(\textit{Par-var-type}') =$ **otherwise**

[10] $\textit{tc-program-start-block}(\textit{LSC\_PROGRAM}\ \text{Identifier}';) =$

[11] $\textit{tc-program-start-block}(\textit{LSC\_PROGRAM}\ \text{Identifier}'\ \text{Central-parameter-list}';) =$
$\quad \textit{check-parameter-list}(\text{Central-parameter-list}')$

[12] $$\frac{\text{Identifier}' \neq \text{Identifier}''}{\textit{check-parameter-list}(\{\text{Identifier}', \text{Identifier}''\}) =}$$

[13] $$\frac{\begin{array}{c}\text{Identifier}' \neq \text{Identifier}'',\\ \text{Identifier}'' \neq \text{Identifier}''',\\ \text{Identifier}''' \neq \text{Identifier}'\end{array}}{\textit{check-parameter-list}(\{\text{Identifier}', \text{Identifier}'', \text{Identifier}'''\}) =}$$

[14] $\textit{check-parameter-list}(\text{Central-parameter-list}') =$
$\quad$ *duplicate identifier-s in Central-parameter-list* Central-parameter-list$'$ **otherwise**

[15] $\textit{tc-configuration-block}(\text{Configuration-block}') =$

The operational semantics of the Configuration-block is too unclear for us to perform test on it. We expect test on the TARGET field to be reasonable.

[16] $\textit{tc-type-environment}(\text{Tenv}') = \textit{join-errors}(\textit{tc-pars}(\text{Tenv}'), \textit{tc-vars}(\text{Tenv}'))$

[17] $tc\text{-}pars(Tenv') =$

$$join\text{-}errors(double\text{-}check\text{-}par\text{-}declarations(IdealParameters,\ par\text{-}declarations(Tenv')),\ check\text{-}par\text{-}unicity(par\text{-}declarations(Tenv')))$$

[18] $tc\text{-}vars(Tenv') =$

$$join\text{-}errors(check\text{-}var\text{-}central\text{-}types(var\text{-}declarations(Tenv')),\ join\text{-}errors(double\text{-}check\text{-}var\text{-}declarations(IdealVariables,\ var\text{-}declarations(Tenv')),\ join\text{-}errors(check\text{-}var\text{-}unicity(var\text{-}declarations(Tenv')),\ check\text{-}qualifiers(Tenv'))))$$

[19] $check\text{-}par\text{-}unicity(\text{Par-declarations}'\ \text{Identifier}' : \text{Par-type}';\ \text{Par-declarations}''\ \text{Identifier}' : \text{Par-type}'';\ \text{Par-declarations}''') =$

*redeclaration of* Identifier$'$ *within Parameter declaration block*

[20] $check\text{-}par\text{-}unicity(\text{Par-declarations}') =$ **otherwise**

[21] $check\text{-}var\text{-}unicity(\text{Var-declarations}'\ \text{Identifier}' : \text{Par-var-type}';\ \text{Var-declarations}''\ \text{Identifier}' : \text{Par-var-type}'';\ \text{Var-declarations}''') =$

*redeclaration of* Identifier$'$ *within Variable declaration block*

[22] $check\text{-}var\text{-}unicity(\text{Var-declarations}') =$ **otherwise**

[23] $tc\text{-}function\text{-}blocks(Tenv', ) =$

[24]
$$\frac{tc\text{-}function\text{-}block(Tenv', \text{Function-block}') =}{tc\text{-}function\text{-}blocks(Tenv', \text{Function-block}'\ \text{Function-blocks}') = tc\text{-}function\text{-}blocks(Tenv', \text{Function-blocks}')}$$

[25]
$$\frac{tc\text{-}function\text{-}block(Tenv', \text{Function-block}') = \text{Error-list}',\quad \text{Error-list}' \neq ,\quad \text{Error-list}' = \text{Error-list}'}{\begin{array}{l}tc\text{-}function\text{-}blocks(Tenv', \text{Function-block}'\ \text{Function-blocks}') =\\ join\text{-}errors(*\sqcup*\sqcup*\sqcup TYPE\text{-}error(s){-}in\text{-}function\text{-}block\sqcup FUNCTION\ name(\text{Function-block}'),\\ \quad join\text{-}errors(\text{Error-list}',\ tc\text{-}function\text{-}blocks(Tenv', \text{Function-blocks}')))\end{array}}$$

[26] $tc\text{-}merge\text{-}blocks(Tenv', ) =$

[27]
$$\frac{tc\text{-}merge\text{-}block(Tenv', \text{Merge-block}') =}{tc\text{-}merge\text{-}blocks(Tenv', \text{Merge-block}'\ \text{Merge-blocks}') = tc\text{-}merge\text{-}blocks(Tenv', \text{Merge-blocks}')}$$

$$
[28]\quad \frac{\begin{array}{c} \textit{tc-merge-block}(\mathrm{Tenv}', \textit{Merge-block}') = \textit{Error-list}', \\ \textit{Error-list}' \neq \,, \\ \textit{Error-list}' = \textit{Error-list}' \end{array}}{\begin{array}{l} \textit{tc-merge-blocks}(\mathrm{Tenv}', \textit{Merge-block}'\ \textit{Merge-blocks}') = \\ \textit{join-errors}(*\text{\textvisiblespace}*\text{\textvisiblespace}*\text{\textvisiblespace}\,\textit{TYPE-error}(s){-}\textit{in-merge-block}\text{\textvisiblespace}\textit{MERGE}\ \textit{number}(\textit{Merge-block}'), \\ \qquad\qquad \textit{join-errors}(\textit{Error-list}', \\ \qquad\qquad\qquad\qquad \textit{tc-merge-blocks}(\mathrm{Tenv}', \textit{Merge-blocks}'))) \end{array}}
$$

$$
[29]\quad \textit{tc-flow-blocks}(\mathrm{Tenv}', \,) =
$$

$$
[30]\quad \frac{\textit{tc-flow-block}(\mathrm{Tenv}', \textit{Flow-block}') = }{\begin{array}{l} \textit{tc-flow-blocks}(\mathrm{Tenv}', \textit{Flow-block}'\ \textit{Flow-blocks}') = \\ \textit{tc-flow-blocks}(\mathrm{Tenv}', \textit{Flow-blocks}') \end{array}}
$$

$$
[31]\quad \frac{\begin{array}{c} \textit{tc-flow-block}(\mathrm{Tenv}', \textit{Flow-block}') = \textit{Error-list}', \\ \textit{Error-list}' \neq \,, \\ \textit{Error-list}' = \textit{Error-list}' \end{array}}{\begin{array}{l} \textit{tc-flow-blocks}(\mathrm{Tenv}', \textit{Flow-block}'\ \textit{Flow-blocks}') = \\ \textit{join-errors}(*\text{\textvisiblespace}*\text{\textvisiblespace}*\text{\textvisiblespace}\,\textit{TYPE-error}(s){-}\textit{in-flow-block}\text{\textvisiblespace}\textit{FLOW}\ \textit{initiator}(\textit{Flow-block}'), \\ \qquad\qquad \textit{join-errors}(\textit{Error-list}', \\ \qquad\qquad\qquad\qquad \textit{tc-flow-blocks}(\mathrm{Tenv}', \textit{Flow-blocks}'))) \end{array}}
$$

$$
[32]\quad \begin{array}{l} \textit{tc-function-block}(\mathrm{Tenv}', \\ \qquad\qquad \textit{LSC\_FUNCTION}\ \mathrm{Identifier}'; \\ \qquad\qquad\qquad\qquad \textit{Statements}' \\ \qquad\qquad\qquad\qquad \textit{Merge-blocks}'\ \textit{LSC\_FUNCTION\_END};) = \\ \textit{join-errors}(\textit{exist-var}(\mathrm{Identifier}', \textit{var-declarations}(\mathrm{Tenv}')), \\ \qquad\qquad \textit{join-errors}(\textit{tc-statements}(\mathrm{Tenv}', \textit{Statements}'), \\ \qquad\qquad\qquad\qquad \textit{tc-merge-blocks}(\mathrm{Tenv}', \textit{Merge-blocks}'))) \end{array}
$$

$$
[33]\quad \begin{array}{l} \textit{tc-merge-block}(\mathrm{Tenv}', \\ \qquad\qquad \textit{LSC\_MERGE}\text{\textvisiblespace}\textit{LABEL\_}\ \textit{Label}'; \\ \qquad\qquad\qquad\qquad \textit{Statements}'\ \textit{LSC\_MERGE\_END};) = \\ \textit{tc-statements}(\mathrm{Tenv}', \textit{Statements}') \end{array}
$$

$$
[34]\quad \begin{array}{l} \textit{tc-flow-block}(\mathrm{Tenv}', \\ \qquad\qquad \textit{LSC\_FLOW}\ \mathrm{Flow\text{-}initiator}'; \\ \qquad\qquad\qquad\qquad \textit{Statements}'\ \textit{LSC\_FLOW\_END};) = \\ \textit{join-errors}(\textit{check-normal-flow}(\mathrm{Tenv}', \textit{Flow-initiator}'), \\ \qquad\qquad \textit{tc-statements}(\mathrm{Tenv}', \textit{Statements}')) \end{array}
$$

$$
[35]\quad \begin{array}{l} \textit{tc-flow-block}(\mathrm{Tenv}', \\ \qquad\qquad \textit{LSC\_CNT\_FLOW}\ \mathrm{Flow\text{-}initiator}' \\ \qquad\qquad\qquad\qquad \textit{Central-parameter-list}'; \\ \qquad\qquad\qquad\qquad \textit{Statements}'\ \textit{LSC\_CNT\_FLOW\_END};) = \\ \textit{join-errors}(\textit{check-central-flow}(\mathrm{Tenv}', \textit{Flow-initiator}', \\ \qquad\qquad\qquad\qquad \textit{Central-parameter-list}'), \\ \qquad\qquad \textit{tc-statements}(\mathrm{Tenv}', \textit{Statements}')) \end{array}
$$

[36] $\textit{tc-statements}(\text{Tenv}', ) =$

[37] $\textit{tc-statements}(\text{Tenv}', \text{Statement}' \ \text{Statements}') =$

$\quad \textit{join-errors}(\textit{tc-statement}(\text{Tenv}', \text{Statement}'),$
$\qquad \textit{tc-statements}(\text{Tenv}', \text{Statements}'))$

[38]
$$\frac{\textit{test-expression-type}(\text{Tenv}', \text{Test-expression}') = \mathit{BOOL\_TYPE}}{\begin{array}{l}\textit{tc-statement}(\text{Tenv}', \\ \qquad \mathit{LSC\_SPLIT}\ \text{Test-expression}'; \\ \qquad\qquad \text{Statements}'\ \mathit{LSC\_SPLIT\_END};) = \\ \textit{join-errors}(\textit{tc-test-expression}(\text{Tenv}', \text{Test-expression}'), \\ \qquad \textit{tc-statements}(\text{Tenv}', \text{Statements}'))\end{array}}$$

$\textit{tc-statement}(\text{Tenv}',$
$\qquad \mathit{LSC\_SPLIT}\ \text{Test-expression}';$

[39] $\qquad\qquad \text{Statements}'\ \mathit{LSC\_SPLIT\_END};) =$

$\textit{join-errors}(\textit{test expression}\ \text{Test-expression}'\ \textit{is-not-boolean},$ **otherwise**
$\qquad \textit{join-errors}(\textit{tc-test-expression}(\text{Tenv}', \text{Test-expression}'),$
$\qquad\qquad \textit{tc-statements}(\text{Tenv}', \text{Statements}')))$

[40] $\textit{tc-statement}(\text{Tenv}', \mathit{LSC\_SET}\ \text{Set-expression}';) =$

$\quad \textit{tc-set-expression}(\text{Tenv}', \text{Set-expression}')$

$\textit{tc-statement}(\text{Tenv}',$

[41] $\qquad \mathit{LSC\_SEND}\ \text{Identifier}' : \text{Telegram-output-type}';) =$

$\textit{check-normal-send}(\mathit{IdealTelegrams},$
$\qquad \text{Identifier}', \text{Telegram-output-type}')$

$\textit{tc-statement}(\text{Tenv}',$
$\qquad \mathit{LSC\_CNT\_SEND}\ \text{Identifier}'$
$\qquad : \text{Telegram-output-type}'$

[42] $\qquad \text{Central-parameter-list}';) \quad =$

$\textit{check-central-send}(\mathit{IdealTelegrams},$
$\qquad \text{Identifier}', \text{Telegram-output-type}',$
$\qquad\qquad \text{Central-parameter-list}')$

$\textit{tc-statement}(\text{Tenv}',$

[43] $\qquad \mathit{LSC\_TEST}\ \text{Test-expression}';) =$

$\textit{tc-test-expression}(\text{Tenv}', \text{Test-expression}')$

[44] $\textit{tc-statement}(\text{Tenv}', \mathit{LSC\_CONT\_AT}\ \mathit{LABEL\_}\ \text{Label}';) =$

[45]
$$\frac{\begin{array}{c}\text{Telegram-in-type}' \neq \$, \\ \text{Telegram-in-type}' \neq @, \\ \text{Telegram-in-type}' \neq P, \\ \text{Telegram-in-type}' \neq \&\end{array}}{\begin{array}{l}\textit{check-normal-flow}(\text{Tenv}', \text{Identifier}' : \text{Telegram-in-type}') = \\ \textit{check-normal-send}(\mathit{IdealTelegrams}, \text{Identifier}', \\ \qquad\qquad \text{Telegram-in-type}')\end{array}}$$

[46] $\mathit{check\text{-}normal\text{-}flow}(\mathrm{Tenv}', \mathrm{Identifier}' : \mathit{Telegram\text{-}in\text{-}type}'(\mathit{Dir\text{-}list}')) =$

$\quad \mathit{join\text{-}errors}(\mathit{exist\text{-}var}(\mathrm{Identifier}', \mathit{var\text{-}declarations}(\mathrm{Tenv}')),$
$\qquad \mathit{check\text{-}Dir\text{-}list}(\mathit{Dir\text{-}list}'))$

[47] $\mathit{check\text{-}normal\text{-}flow}(\mathrm{Tenv}', \mathrm{Identifier}' : \mathit{Telegram\text{-}in\text{-}type}') =$

$\quad \mathit{exist\text{-}var}(\mathrm{Identifier}', \mathit{var\text{-}declarations}(\mathrm{Tenv}'))$ **otherwise**

Telegram-in-type should have been checked with its declaration.

[48]
$$\frac{\begin{array}{c}\mathit{Telegram\text{-}in\text{-}type}' \neq \$, \\ \mathit{Telegram\text{-}in\text{-}type}' \neq @, \\ \mathit{Telegram\text{-}in\text{-}type}' \neq P, \\ \mathit{Telegram\text{-}in\text{-}type}' \neq \& \end{array}}{\begin{array}{l}\mathit{check\text{-}central\text{-}flow}(\mathrm{Tenv}', \mathrm{Identifier}' : \mathit{Telegram\text{-}in\text{-}type}', \\ \qquad \mathrm{Central\text{-}parameter\text{-}list}') \quad = \\ \mathit{join\text{-}errors}(\mathit{check\text{-}central\text{-}send}(\mathit{IdealTelegrams}, \\ \qquad \mathrm{Identifier}', \mathit{Telegram\text{-}in\text{-}type}', \\ \qquad \quad \mathrm{Central\text{-}parameter\text{-}list}'), \\ \quad \mathit{check\text{-}parameter\text{-}list}(\mathrm{Central\text{-}parameter\text{-}list}'))\end{array}}$$

[49]
$$\begin{array}{l}\mathit{check\text{-}central\text{-}flow}(\mathrm{Tenv}', \\ \qquad \mathrm{Identifier}' : \mathit{Telegram\text{-}in\text{-}type}'(\mathit{Dir\text{-}list}'), \\ \qquad \mathrm{Central\text{-}parameter\text{-}list}') \quad = \\ \mathit{join\text{-}errors}(\mathit{exist\text{-}var}(\mathrm{Identifier}', \mathit{var\text{-}declarations}(\mathrm{Tenv}')), \\ \quad \mathit{join\text{-}errors}(\mathit{check\text{-}Dir\text{-}list}(\mathit{Dir\text{-}list}'), \\ \qquad \mathit{check\text{-}parameter\text{-}list}(\mathrm{Central\text{-}parameter\text{-}list}')))\end{array}$$

[50]
$$\begin{array}{l}\mathit{check\text{-}central\text{-}flow}(\mathrm{Tenv}', \mathrm{Identifier}' : \mathit{Telegram\text{-}in\text{-}type}', \\ \qquad \mathrm{Central\text{-}parameter\text{-}list}') \quad = \\ \mathit{join\text{-}errors}(\mathit{exist\text{-}var}(\mathrm{Identifier}', \mathit{var\text{-}declarations}(\mathrm{Tenv}')), \quad \textbf{otherwise} \\ \quad \mathit{check\text{-}parameter\text{-}list}(\mathrm{Central\text{-}parameter\text{-}list}'))\end{array}$$

[51]
$$\frac{\begin{array}{c}\mathit{lookup\text{-}telegram}(\mathrm{Telenv}', \mathrm{Identifier}') = \mathit{Telegram\text{-}type}', \\ \mathit{telegram\text{-}type}(\mathit{Telegram\text{-}in\text{-}type}') = \mathit{Telegram\text{-}type}'', \\ \mathit{Telegram\text{-}type}' = \mathit{Telegram\text{-}type}''\end{array}}{\mathit{check\text{-}normal\text{-}send}(\mathrm{Telenv}', \mathrm{Identifier}', \mathit{Telegram\text{-}in\text{-}type}') =}$$

[52]
$$\frac{\begin{array}{c}\mathit{lookup\text{-}telegram}(\mathrm{Telenv}', \mathrm{Identifier}') = \mathit{Telegram\text{-}type}'', \\ \mathit{telegram\text{-}type}(\mathit{Telegram\text{-}in\text{-}type}') = \mathit{Telegram\text{-}type}', \\ \mathit{Telegram\text{-}type}'' \neq \mathit{Telegram\text{-}type}'\end{array}}{\begin{array}{l}\mathit{check\text{-}normal\text{-}send}(\mathrm{Telenv}', \mathrm{Identifier}', \mathit{Telegram\text{-}in\text{-}type}') = \\ \mathit{join\text{-}errors}(\mathit{illegal\ use\ of\ telegram}\ \mathrm{Identifier}'\ \mathit{as}\ \mathit{Telegram\text{-}in\text{-}type}', \\ \quad \mathit{lookup\text{-}telegram}(\mathrm{Telenv}', \mathrm{Identifier}')\ \mathit{expected})\end{array}}$$

[53]
$$\frac{\begin{array}{c}\mathit{lookup\text{-}telegram}(\mathrm{Telenv}', \mathrm{Identifier}') = \mathit{CENTRAL}, \\ \mathit{telegram\text{-}type}(\mathit{Telegram\text{-}in\text{-}type}') = \mathit{CENTRAL}\end{array}}{\begin{array}{l}\mathit{check\text{-}central\text{-}send}(\mathrm{Telenv}', \mathrm{Identifier}', \mathit{Telegram\text{-}in\text{-}type}', \\ \qquad \mathrm{Central\text{-}parameter\text{-}list}') =\end{array}}$$

[54]
$$\begin{array}{l}\mathit{check\text{-}central\text{-}send}(\mathrm{Telenv}', \mathrm{Identifier}', \mathit{Telegram\text{-}in\text{-}type}', \\ \qquad \mathrm{Central\text{-}parameter\text{-}list}') = \\ \mathit{join\text{-}errors}(\mathit{illegal\ use\ of\ telegram}\ \mathrm{Identifier}'\ \mathit{as}\ \mathit{Telegram\text{-}in\text{-}type}', \quad \textbf{otherwise} \\ \quad \mathit{lookup\text{-}telegram}(\mathrm{Telenv}', \mathrm{Identifier}')\ \mathit{expected})\end{array}$$

Set-expressions

$$[55]\quad \frac{\begin{array}{l} \textit{set-compatible}(\textit{lookup-telegram-data}(\text{Telegram-data-name}', \\ \qquad\qquad \textit{telegram-data-declarations}(\text{Tenv}')), \\ \qquad \textit{test-expression-type}(\text{Tenv}', \text{Test-expression}')) \qquad = \textit{true} \end{array}}{\begin{array}{l} \textit{tc-set-expression}(\text{Tenv}', \text{Telegram-data-name}' : \text{Test-expression}') = \\ \textit{exist-telegram-data}(\text{Telegram-data-name}', \\ \qquad \textit{telegram-data-declarations}(\text{Tenv}')) \end{array}}$$

No check on range since Test-expression has evaluated to a Boolean value [0..1] and telegram-data is int by default

$$[56]\quad \begin{array}{l} \textit{tc-set-expression}(\text{Tenv}', \text{Telegram-data-name}' : \text{Test-expression}') = \\ \quad \textit{join-errors}(\textit{exist-telegram-data}(\text{Telegram-data-name}', \qquad\qquad \textbf{otherwise} \\ \qquad\qquad \textit{telegram-data-declarations}(\text{Tenv}')), \\ \qquad \textit{join-errors}(\textit{incompatible lhs} : \text{Telegram-data-name}' \\ \qquad\qquad \textit{assigned with rhs} : \text{Test-expression}', \\ \qquad\qquad \textit{tc-test-expression}(\text{Tenv}', \text{Test-expression}'))) \end{array}$$

$$[57]\quad \frac{\begin{array}{l} \textit{is-valid-lhs}(\text{Par-var-exp}') = \textit{true}, \\ \textit{set-compatible}(\textit{par-var-exp-type}(\text{Tenv}', \text{Par-var-exp}'), \\ \qquad \textit{test-expression-type}(\text{Tenv}', \text{Test-expression}')) = \textit{true} \end{array}}{\textit{tc-set-expression}(\text{Tenv}', \text{Par-var-exp}' : \text{Test-expression}') =}$$

$$[58]\quad \begin{array}{l} \textit{tc-set-expression}(\text{Tenv}', \text{Par-var-exp}' : \text{Test-expression}') = \\ \quad \textit{join-errors}(\textit{check-lhs}(\text{Par-var-exp}'), \qquad\qquad \textbf{otherwise} \\ \qquad \textit{join-errors}(\textit{tc-par-var-exp}(\text{Tenv}', \text{Par-var-exp}'), \\ \qquad\quad \textit{join-errors}(\textit{incompatible lhs} : \text{Par-var-exp}' \\ \qquad\qquad \textit{assigned with rhs} : \text{Test-expression}', \\ \qquad\qquad \textit{tc-test-expression}(\text{Tenv}', \text{Test-expression}')))) \end{array}$$

$$[59]\quad \frac{\begin{array}{l} \textit{set-compatible}(\textit{lookup-telegram-data}(\text{Telegram-data-name}', \\ \qquad\qquad \textit{telegram-data-declarations}(\text{Tenv}')), \\ \qquad \textit{simple-expression-type}(\text{Tenv}', \text{Simple-expression}')) \qquad = \textit{true} \end{array}}{\begin{array}{l} \textit{tc-set-expression}(\text{Tenv}', \text{Telegram-data-name}' : \text{Simple-expression}') = \\ \textit{join-errors}(\textit{exist-telegram-data}(\text{Telegram-data-name}', \\ \qquad\qquad \textit{telegram-data-declarations}(\text{Tenv}')), \\ \qquad \textit{check-tel-dat-range}(\textit{telegram-data-declarations}(\text{Tenv}'), \text{Telegram-data-name}', \\ \qquad\qquad \text{Simple-expression}')) \end{array}}$$

$$[60]\quad \frac{\begin{array}{l} \textit{set-compatible}(\textit{lookup-telegram-data}(\text{Telegram-data-name}', \\ \qquad\qquad \textit{telegram-data-declarations}(\text{Tenv}')), \\ \qquad \textit{simple-expression-type}(\text{Tenv}', \text{Simple-expression}')) \qquad = \textit{false} \end{array}}{\begin{array}{l} \textit{tc-set-expression}(\text{Tenv}', \text{Telegram-data-name}' : \text{Simple-expression}') = \\ \textit{join-errors}(\textit{exist-telegram-data}(\text{Telegram-data-name}', \\ \qquad\qquad \textit{telegram-data-declarations}(\text{Tenv}')), \\ \qquad \textit{join-errors}(\textit{incompatible lhs} : \text{Telegram-data-name}' \\ \qquad\qquad \textit{assigned with rhs} : \text{Simple-expression}', \\ \qquad\qquad \textit{join-errors}(\textit{check-tel-dat-range}(\textit{telegram-data-declarations}(\text{Tenv}'), \\ \qquad\qquad\qquad \text{Telegram-data-name}', \\ \qquad\qquad\qquad \text{Simple-expression}'), \\ \qquad\qquad\quad \textit{tc-simple-expression}(\text{Tenv}', \text{Simple-expression}')))) \end{array}}$$

$$[61]\quad \frac{\begin{array}{c} \textit{is-valid-lhs}(\text{Par-var-exp}') = \textit{true}, \\ \textit{set-compatible}(\textit{par-var-exp-type}(\text{Tenv}', \text{Par-var-exp}'), \\ \textit{simple-expression-type}(\text{Tenv}', \text{Simple-expression}')) = \textit{true} \end{array}}{\textit{tc-set-expression}(\text{Tenv}', \text{Par-var-exp}' : \text{Simple-expression}') = }$$

[62] $\textit{tc-set-expression}(\text{Tenv}', \text{Par-var-exp}' : \text{Simple-expression}') =$
$\textit{join-errors}(\textit{check-lhs}(\text{Par-var-exp}'),$ **otherwise**
$\textit{join-errors}(\textit{tc-par-var-exp}(\text{Tenv}', \text{Par-var-exp}'),$
$\textit{join-errors}(\textit{incompatible lhs} : \text{Par-var-exp}'$
$\textit{assigned with rhs} : \text{Simple-expression}',$
$\textit{tc-simple-expression}(\text{Tenv}', \text{Simple-expression}'))))$

[63] $\textit{is-valid-lhs}(P.\ \text{Identifier}') = \textit{false}$
[64] $\textit{is-valid-lhs}(P.\ \text{Identifier}'(\text{Var-attribute}')) = \textit{false}$
[65] $\textit{is-valid-lhs}(P.\ \text{Identifier}'\ \{*\}) = \textit{false}$
[66] $\textit{is-valid-lhs}(P.\ \text{Identifier}'\ \{\#\}) = \textit{false}$

[67] $\textit{is-valid-lhs}(P.\ \text{Identifier}'(\text{Var-attribute}')[\text{Simple-expression}']) = \textit{false}$
[68] $\textit{is-valid-lhs}(P.\ \text{Identifier}'[\text{Simple-expression}']) =$
$\textit{false}$
[69] $\textit{is-valid-lhs}(\text{Par-var-exp}') = \textit{true}$ **otherwise**

[70] $\textit{check-lhs}(P.\ \text{Identifier}') =$
$\textit{parameter}\ \text{Identifier}'\ \textit{in}_\sqcup\textit{lhs of assignment}$
[71] $\textit{check-lhs}(P.\ \text{Identifier}'(\text{Var-attribute}')) =$
$\textit{parameter}\ \text{Identifier}'\ \textit{in}_\sqcup\textit{lhs of assignment}$
[72] $\textit{check-lhs}(P.\ \text{Identifier}'\ \{*\}) =$
$\textit{parameter}\ \text{Identifier}'\ \textit{in}_\sqcup\textit{lhs of assignment}$
[73] $\textit{check-lhs}(P.\ \text{Identifier}'\ \{\#\}) =$
$\textit{parameter}\ \text{Identifier}'\ \textit{in}_\sqcup\textit{lhs of assignment}$
[74] $\textit{check-lhs}(P.\ \text{Identifier}'(\text{Var-attribute}')[\text{Simple-expression}']) =$
$\textit{parameter}\ \text{Identifier}'\ \textit{in}_\sqcup\textit{lhs of assignment}$
[75] $\textit{check-lhs}(P.\ \text{Identifier}'[\text{Simple-expression}']) =$
$\textit{parameter}\ \text{Identifier}'\ \textit{in}_\sqcup\textit{lhs of assignment}$
[76] $\textit{check-lhs}(\text{Par-var-exp}') =$ **otherwise**

# J Errors

**Error Reporting** This module defines a data type for reporting Errors. It is generic and can be used for all kinds of error reporting.

**imports** Layout$^{(R)}$
**exports**
  **sorts** ERROR ERROR-LIST
  **context-free syntax**
    {ERROR ";"}* → ERROR-LIST
    *join-errors*(ERROR-LIST, ERROR-LIST) → ERROR-LIST
  **variables**
    "*Error*"[']+ → ERROR
    "*Error-list*"[']+→ {ERROR ";"}*
**equations**

[1] $\textit{join-errors}(\textit{Error-list}', \textit{Error-list}'') = \textit{Error-list}'; \textit{Error-list}''$

# K Fundamental-elements

**Lexical Notions** The module Fundamental-elements defines the IDEAL identifiers and other lexical notions used to define terms. Note that we have treated all identifiers (Identifiers, Par-var-identifier, and Telegram-name) to be equal. See recommendations on this topic (cf. Section 7.

**imports** Range$^{(T)}$
**exports**

**Labels** in the current version of the IDEAL compiler, labels are normalized not to have leading zeros. This means that "003" is equal with the label "3". Since we have not defined the semantics of Labels, within this specification we still consider them to be different. This choice is arbitrary. The separation of Labels and Numbers, however, should be made.

  **sorts** DIGIT Letter Letter-or-digit Label String Number Version-field Version
  **lexical syntax**
    [*0-9*] → DIGIT
    DIGIT+ → Label
    DIGIT+ → Number
    [*a-zA-Z*] → Letter
    [*a-zA-Z0-9*] → Letter-or-digit
    "\""~["]*"\"" → String

    DIGIT → Version-field
    DIGIT DIGIT → Version-field
    Version-field Version-field → Version

**Identifiers** The Par-var tags are added to smoothen Identifier syntax.

  **sorts** Par-tag Var-tag Par-var-tag Identifier Telegram-data-name
  **lexical syntax**
    Letter Letter-or-digit Letter-or-digit+ → Identifier
    Letter → Telegram-data-name

Letter Letter-or-digit → Telegram-data-name

**context-free syntax**

"*V.*" → Var-tag
"*P.*" → Par-tag
Var-tag → Par-var-tag
Par-tag → Par-var-tag

**Directions** It would actually have been better to define Directions as a 'star'-list and to perform a static check on these list for them to contain at least one node. Now the Check-directions and Check-qualifiers modules have been unneccessarily cluttered with awkward test functions.

**sorts** Direction Directions Dir-list

**lexical syntax**

[*a-dA-D*] → Direction

**context-free syntax**

Direction+ → Directions
{Directions ","}+ → Dir-list

**Telegram Types and Qualifiers** Subclassification of telegrams into central and normal telegrams are added for ease of typechecking.

**sorts** Initiator Ini-list Var-attribute Central-telegram Normal-telegram Telegram-in-type Telegram-output-type

**context-free syntax**

Normal-telegram → Telegram-output-type
Central-telegram → Telegram-output-type

Telegram-output-type → Telegram-in-type
"*P*" → Telegram-in-type
"*&*" → Telegram-in-type
"@" → Telegram-in-type
"*$*" → Telegram-in-type

Telegram-in-type → Initiator
"*I*" → Initiator
{Initiator ","}+ → Ini-list

Directions "," Initiator → Var-attribute
Directions → Var-attribute
Initiator → Var-attribute

**Additional List Types** Directions and Initiator lists are used within IDEAL Programs in the declarations section. Var-attributes, however, are used in the the Flow-blocks, see the modules Ideal-LSC and Ideal-expressions respectively.

**sorts** Dir-ini-list

**context-free syntax**

"(" Dir-list ")" → Dir-ini-list
"(" Ini-list ")" → Dir-ini-list
"(" Dir-list "," Ini-list ")" → Dir-ini-list

**Flow-initiator** is added to conform to the latest change of the IDEAL syntax. This change has been done after the Guide & Ref. version 0.3 had been issued.

**sorts** Flow-initiator
**context-free syntax**
Identifier ":" Telegram-in-type
"(" Dir-list ")" → Flow-initiator
Identifier ":" Telegram-in-type → Flow-initiator

**Central Element Parametrization** The usage of central parameters can be compared with a parametrized program or function call in Pascal, see module Ideal-LSC.

**sorts** Central-parameter-list
**context-free syntax**
"{" {Identifier ","}+ "}" → Central-parameter-list
**variables**
*Central-parameter-list* [′]+→ Central-parameter-list

# L Ideal-LSC

**Control Flow and Keywords** This module contains the syntax of the Control Flow KeyWords and IDEAL LSC Program constructs.

**Case Sensitivity** Unlike IDEAL, this SDF definition of IDEAL is case sensitive with respect to IDEAL keywords, which are expected to be in UPPER CASE. Mixed cased input is trivially solved by using the Emacs GNU editor `<upper-case-region>` command as a preprocessor.

**Statements and Merge-blocks** The subclassification of Function-statements and Flow-statements at the syntactical level has been abandoned. This used to cause syntactical ambiguities throughout the specification. The test on the scope for which Function-statements should be used is now implemented by checks at the static semantical level. An analoguous argumentation can be held for Function-merge-blocks. They have also disappeared at the syntactical level.

The test on the presence of a Flow-block has also been moved to the static semantic level.

For each sort that occurs in plural auxiliary sorts have been introduced for ease of writing equations.

**imports** Ideal-Types$^{(M)}$
**exports**
**sorts** Ideal-program Program-start-block Configuration-block
Parameter-block Variable-block Function-block Merge-block
Flow-block Program-end-block Par-declaration Var-declaration
Config-declaration Statement Statements Function-blocks
Merge-blocks Flow-blocks Par-declarations Var-declarations
Config-declarations
**context-free syntax**
Function-block∗ → Function-blocks
Merge-block∗ → Merge-blocks

| | |
|---|---|
| Flow-block∗ | → Flow-blocks |
| Statement∗ | → Statements |
| Par-declaration∗ | → Par-declarations |
| Var-declaration∗ | → Var-declarations |
| Config-declaration∗ | → Config-declarations |
| Program-start-block<br>Configuration-block<br>Parameter-block<br>Variable-block<br>Function-blocks<br>Merge-blocks<br>Flow-blocks<br>Program-end-block | → Ideal-program |
| "*LSC_PROGRAM*" Identifier ";" | → Program-start-block |
| "*LSC_PROGRAM*" Identifier<br>Central-parameter-list ";" | → Program-start-block |
| "*LSC_CONFIG*"<br>Config-declarations | → Configuration-block |
| | → Configuration-block |
| "*LSC_VERSION*" ":" Version ";" | → Config-declaration |
| "*IDEAL_TARGET*" ":" String ";" | → Config-declaration |
| "*IDEAL_COMPANY*" ":" String ";" | → Config-declaration |
| "*LSC_DATE*" ":" String ";" | → Config-declaration |
| "*LSC_AUTHOR*" ":" String ";" | → Config-declaration |
| "*LSC_VERSION*" ":" String ";" | → Config-declaration |
| "*LSC_PAR*" Par-declarations | → Parameter-block |
| | → Parameter-block |
| "*LSC_VAR*" Var-declarations | → Variable-block |
| | → Variable-block |
| "*LSC_FUNCTION*" Identifier ";"<br>Statements<br>Merge-blocks<br>"*LSC_FUNCTION_END*" ";" | → Function-block |
| "*LSC_MERGE␣LABEL_*" Label ";"<br>Statements<br>"*LSC_MERGE_END*" ";" | → Merge-block |
| "*LSC_FLOW*" Flow-initiator ";"<br>Statements<br>"*LSC_FLOW_END*" ";" | → Flow-block |
| "*LSC_CNT_FLOW*" Flow-initiator<br>Central-parameter-list ";"<br>Statements<br>"*LSC_CNT_FLOW_END*" ";" | → Flow-block |
| "*LSC_PROGRAM_END*" ";" | → Program-end-block |
| "*LSC_SET*" Set-expression ";" | → Statement |

| | |
|---|---|
| "*LSC_TEST*" Test-expression ";" | → Statement |
| "*LSC_SPLIT*" Test-expression ";"<br>Statements<br>"*LSC_SPLIT_END*" ";" | → Statement |
| "*LSC_SEND*" Identifier ":"<br>Telegram-output-type ";" | → Statement |
| "*LSC_CNT_SEND*" Identifier ":"<br>Telegram-output-type<br>Central-parameter-list ";" | → Statement |
| "*LSC_CONT_AT*" "*LABEL_*" Label ";" | → Statement |
| Identifier ":" Par-type ";" | → Par-declaration |
| Identifier "(" Dir-list ")" ":"<br>Par-type ";" | → Par-declaration |
| Identifier ":" Par-var-type ";" | → Var-declaration |
| Identifier Dir-ini-list ":"<br>Par-var-type ";" | → Var-declaration |

# M Ideal-Types

**Disambiguations of 0 and 1** The Sorts Boolean-constructor and Non-boolean-int are auxiliary sorts to distinguish "0" "1" from other Numbers. The alternative would be to make the distinction by range checking, but this would automatically mean that all IDEAL Types should be modeled in this way if we want to preserve the symmetry of the language.

**Type Summary** The SDF definition knows only of two basic IDEAL Sorts: ENUM_TYPE and Numbers. Other 'Types' are derived from these sorts at the static semantic level or using program constructs.

**Type Conversion** The sort Number is converted to the Standard Integers (without min max int) for computations within equations. The semantics of IDEAL Integers is beyond the scope of this study.

**Type Compatibility** We make a distinction of three classes of Type Compatibility

1. compatibility for the assignment operator
2. compatibility for ordering and arithmetic operators
3. compatibility for the equivalence operator

**imports** Ideal-expressions$^{(N)}$
**exports**
  **sorts** Par-type Par-var-type Boolean-constructor Non-boolean-int
    ENUM_TYPE
  **context-free syntax**

| | |
|---|---|
| Letter | → ENUM_TYPE |
| “*BOOL_LIST_TYPE*” | → Par-type |
| “*INT_LIST_TYPE*” | → Par-type |
| “*ENUM_TYPE*” | → Par-type |
| “*BOOL_TYPE*” | → Par-type |
| “*INT_TYPE*” | → Par-type |
| Par-type | → Par-var-type |
| “*BOOL_CNT_TYPE*” Central-parameter-list | → Par-var-type |
| “*INT_CNT_TYPE*” Central-parameter-list | → Par-var-type |
| “*BOOL_FUNC_TYPE*” | → Par-var-type |
| “*INT_FUNC_TYPE*” | → Par-var-type |
| “*BOOLEAN_CONSTRUCTOR*” | → Par-var-type |
| “*NON_BOOLEAN_INT*” | → Par-var-type |
| “LALR” | → Telegram-output-type |
| “LBLR” | → Telegram-output-type |
| “LCLR” | → Telegram-output-type |
| “LDLR” | → Telegram-output-type |
| “LARL” | → Telegram-output-type |
| “LBRL” | → Telegram-output-type |
| “LCRL” | → Telegram-output-type |
| “LDRL” | → Telegram-output-type |
| “MMI” | → Telegram-output-type |
| “TRK” | → Telegram-output-type |
| “*CNT_UP*” | → Telegram-output-type |
| “*CNT_DOWN*” | → Telegram-output-type |
| ENUM_TYPE “=” ENUM_TYPE | → BOOL |
| “MAXINT” | → INT |
| “MININT” | → INT |
| *int-to-simple-expression*(INT) | → Simple-expression |
| *simple-expression-to-int*(Simple-expression) | → INT |
| *is-list-type*(Par-var-type) | → BOOL |
| *is-central-type*(Par-var-type) | → BOOL |
| *range-of-type*(Par-var-type) | → Range |
| *effective-type*(Par-var-type) | → Par-var-type |
| *set-compatible*(Par-var-type, Par-var-type) | → BOOL |
| *set-effective-compatibility*(Par-var-type, Par-var-type) | → BOOL |
| *test-compatibility*(Par-var-type, Par-var-type) | → BOOL |
| *test-compatible*(Par-var-type, Par-var-type) | → BOOL |
| *type-equivalence*(Par-var-type, Par-var-type) | → BOOL |
| *type-equivalent*(Par-var-type, Par-var-type) | → BOOL |

**variables**

| | |
|---|---|
| *Enum* [′]+ | → ENUM_TYPE |
| *Par-type* [′]+ | → Par-type |
| *Par-var-type* [′]+ | → Par-var-type |
| *Boolean-constructor* [′]+ | → Boolean-constructor |

$\textit{Non-boolean-int}\ [']+ \quad \rightarrow \text{Non-boolean-int}$

**hiddens**

**variables**

$y[\textit{0-9}]* \rightarrow \text{CHAR+}$

**equations**

[1] $MININT \;=\; \_-2147483648$

[2] $MAXINT \;=\; 2147483647$

[3] $\textit{range-of-type}(BOOL\_TYPE) = [0, 1]$

[4] $\textit{range-of-type}(BOOL\_CNT\_TYPE\ \text{Central-parameter-list}') = [0, 1]$

[5] $\textit{range-of-type}(BOOL\_FUNC\_TYPE) = [0, 1]$

[6] $\textit{range-of-type}(INT\_TYPE) = [MININT, MAXINT]$

[7] $\textit{range-of-type}(INT\_CNT\_TYPE\ \text{Central-parameter-list}') = [MININT, MAXINT]$

[8] $\textit{range-of-type}(INT\_FUNC\_TYPE) = [MININT, MAXINT]$

[9] $\textit{is-list-type}(INT\_LIST\_TYPE) = \textit{true}$

[10] $\textit{is-list-type}(BOOL\_LIST\_TYPE) = \textit{true}$

[11] $\textit{is-list-type}(\text{Par-var-type}') = \textit{false}$ **otherwise**

[12] $\textit{is-central-type}(BOOL\_CNT\_TYPE\ \text{Central-parameter-list}') = \textit{true}$

[13] $\textit{is-central-type}(INT\_CNT\_TYPE\ \text{Central-parameter-list}') = \textit{true}$

[14] $\textit{is-central-type}(\text{Par-var-type}') = \textit{false}$ **otherwise**

[15] $\textit{effective-type}(INT\_FUNC\_TYPE) = INT\_TYPE$

[16] $\textit{effective-type}(INT\_CNT\_TYPE\ \text{Central-parameter-list}') = INT\_TYPE$

[17] $\textit{effective-type}(BOOL\_FUNC\_TYPE) = BOOL\_TYPE$

[18] $\textit{effective-type}(BOOL\_CNT\_TYPE\ \text{Central-parameter-list}') = BOOL\_TYPE$

[19] $\textit{effective-type}(\text{Par-var-type}') = \text{Par-var-type}'$ **otherwise**

Relational operator compatibility and arithmetic holds only for Numbers and INT_TYPE.

[20] $\textit{test-compatibility}(INT\_TYPE, BOOLEAN\_CONSTRUCTOR) = \textit{true}$

[21] $\textit{test-compatibility}(BOOLEAN\_CONSTRUCTOR, INT\_TYPE) = \textit{true}$

[22] $\textit{test-compatibility}(INT\_TYPE, INT\_TYPE) = \textit{true}$

[23] $\textit{test-compatibility}(INT\_TYPE, NON\_BOOLEAN\_INT) = \textit{true}$

[24] $\textit{test-compatibility}(NON\_BOOLEAN\_INT, INT\_TYPE) = \textit{true}$

[25] $\textit{test-compatibility}(NON\_BOOLEAN\_INT, NON\_BOOLEAN\_INT) = \textit{true}$

[26] $\textit{test-compatibility}(BOOLEAN\_CONSTRUCTOR, BOOLEAN\_CONSTRUCTOR) = \textit{true}$

[27] $\textit{test-compatibility}(\text{Par-var-type}', \text{Par-var-type}'') = \textit{false}$ **otherwise**

Equivalence operator compatibility for Numbers.

[28] $\textit{type-equivalence}(\textit{BOOLEAN\_CONSTRUCTOR}, \textit{BOOL\_TYPE}) = \textit{true}$

[29] $\textit{type-equivalence}(\textit{BOOL\_TYPE}, \textit{BOOLEAN\_CONSTRUCTOR}) = \textit{true}$

[30] $\textit{type-equivalence}(\textit{INT\_TYPE}, \textit{BOOLEAN\_CONSTRUCTOR}) = \textit{true}$

[31] $\textit{type-equivalence}(\textit{BOOLEAN\_CONSTRUCTOR}, \textit{INT\_TYPE}) = \textit{true}$

[32] $\textit{type-equivalence}(\textit{NON\_BOOLEAN\_INT}, \textit{INT\_TYPE}) = \textit{true}$

[33] $\textit{type-equivalence}(\textit{INT\_TYPE}, \textit{NON\_BOOLEAN\_INT}) = \textit{true}$

[34] $\textit{type-equivalence}(\text{Par-var-}\textit{type}', \text{Par-var-}\textit{type}') = \textit{true}$

[35] $\textit{type-equivalence}(\text{Par-var-}\textit{type}', \text{Par-var-}\textit{type}'') = \textit{false}$ **otherwise**

[36] $\textit{test-compatible}(\text{Par-var-}\textit{type}', \text{Par-var-}\textit{type}'') =$
$\textit{test-compatibility}(\textit{effective-type}(\text{Par-var-}\textit{type}'),$
$\textit{effective-type}(\text{Par-var-}\textit{type}''))$

[37] $\textit{type-equivalent}(\text{Par-var-}\textit{type}', \text{Par-var-}\textit{type}'') =$
$\textit{type-equivalence}(\textit{effective-type}(\text{Par-var-}\textit{type}'),$
$\textit{effective-type}(\text{Par-var-}\textit{type}''))$

Note that lhs has to be a variable (syntax) and that functions in Ideal *CAN!* be assigned a value. Functions, however may not be recursively be called. Recursion is not tested here.

[38] $\textit{set-effective-compatibility}(\textit{INT\_TYPE}, \textit{NON\_BOOLEAN\_INT}) = \textit{true}$

[39] $\textit{set-effective-compatibility}(\textit{INT\_TYPE}, \textit{BOOLEAN\_CONSTRUCTOR}) = \textit{true}$

[40] $\textit{set-effective-compatibility}(\textit{BOOL\_TYPE}, \textit{BOOLEAN\_CONSTRUCTOR}) = \textit{true}$

[41] $\textit{set-effective-compatibility}(\text{Par-var-}\textit{type}', \text{Par-var-}\textit{type}') = \textit{true}$

[42] $\textit{set-effective-compatibility}(\text{Par-var-}\textit{type}', \text{Par-var-}\textit{type}'') = \textit{false}$ **otherwise**

[43] $\textit{set-compatible}(\text{Par-var-}\textit{type}', \text{Par-var-}\textit{type}'') =$
$\textit{set-effective-compatibility}(\textit{effective-type}(\text{Par-var-}\textit{type}'),$
$\textit{effective-type}(\text{Par-var-}\textit{type}''))$

The type conversion functions make use of ASF+SDF built in functions. The functions 'nat-con' and 'number' are automatically declared and defined by the ASF+SDF system. The technique involves manipulation of the string representation of the sorts to be typecasted.

[44] $\textit{int-to-simple-expression}(\_- \ \textit{nat-con}(y)) \ = \ - \ \textit{number}(y)$

[45] $\textit{int-to-simple-expression}(\textit{nat-con}(y)) \ = \ \textit{number}(y)$

[46] $\textit{simple-expression-to-int}(- \ \textit{number}(y)) \ = \ \_- \ \textit{nat-con}(y)$

[47] $\textit{simple-expression-to-int}(\textit{number}(y)) \ = \ \textit{nat-con}(y)$

# N Ideal-expressions

**Par-var-exp** (ressions) are traditionally better known as L-value of assignments.

**R-value** is an auxiliary sort to smoothen set-expression syntax. It is actually better to restrict R-value to (very) Simple-expressions and to exclude Test-expression from occuring on the right-hand-side.

**Simplifications** The sorts: add_terms, term, mult_factors, mult_factor, add_op, mult_op, and factor have been eliminated from this IDEAL specification to keep the syntax as flat as possible. The resulting specification is however equivalent upto the associativity of relational operators to the original. The IDEAL auxiliary operator class sorts ( add_op and mult_op ) have been ommitted to guarantee proper disambiguation.

**Absolute Value and List Access** The expression $e$ in $|e|$ , should return an integer. Since Test-expressions use Simple-expressions, we use that type. The SDF definition here is thus different from IDEAL. This is solved at the static semantical level. An analoguous argument holds for the list access operator $[e]$.

**Guard for Assignments of Constants** Note that contrary to 4.5.1. of IDEAL0.3 the assignment to Parameters is prohibited by a static check. The syntax of set_expression in Ideal (the V. part) is ambiguous and introduces redundancy.

**Associativity of Relational Operators** In IDEAL relational operators are varyadic. We feel this to be undesireable. See the section on recommendations (cf. Section 7) for arguments. In this specification relational operators are binary and non-associative.

**imports** Fundamental-elements$^{(\mathrm{K})}$
**exports**
  **sorts** Par-var-exp R-value Test-expression Set-expression
    Simple-expression
  **context-free syntax**

| | |
|---|---|
| Identifier | → Par-var-exp |
| Identifier "(" Var-attribute ")" | → Par-var-exp |
| Identifier "{∗}" | → Par-var-exp |
| Identifier "{#}" | → Par-var-exp |
| Identifier "(" Var-attribute ")" "[" Simple-expression "]" | → Par-var-exp |
| Identifier "[" Simple-expression "]" | → Par-var-exp |
| Par-var-tag Identifier | → Par-var-exp |
| Par-var-tag Identifier "(" Var-attribute ")" | → Par-var-exp |
| Par-var-tag Identifier "{∗}" | → Par-var-exp |
| Par-var-tag Identifier "{#}" | → Par-var-exp |
| Par-var-tag Identifier "(" Var-attribute ")" "[" Simple-expression "]" | → Par-var-exp |
| Par-var-tag Identifier "[" Simple-expression "]" | → Par-var-exp |
| Par-var-exp | → Simple-expression |

Simple-expression → R-value
Test-expression → R-value

Telegram-data-name “:” R-value → Set-expression
Par-var-exp “:” R-value → Set-expression

Simple-expression “=” Simple-expression → Test-expression
Simple-expression “◇” Simple-expression → Test-expression
Simple-expression “<” Simple-expression → Test-expression
Simple-expression “>” Simple-expression → Test-expression
Simple-expression “≤” Simple-expression → Test-expression
Simple-expression “≥” Simple-expression → Test-expression

“%” “=” Dir-list → Test-expression

“−” Simple-expression → Simple-expression
Simple-expression “+” Simple-expression → Simple-expression {**left**}
Simple-expression “−” Simple-expression → Simple-expression {**left**}
Simple-expression “*” Simple-expression → Simple-expression {**left**}
“(” Simple-expression “)” → Simple-expression {**bracket**}

“′” Letter “′” → Simple-expression
Number → Simple-expression
Telegram-data-name → Simple-expression

“|” Simple-expression “|” → Simple-expression

Identifier “[” Simple-expression “]” → Simple-expression

**priorities**
“−”Simple-expression → Simple-expression >
Simple-expression “*”Simple-expression → Simple-expression >
Simple-expression “+”Simple-expression → Simple-expression >
Simple-expression “−”Simple-expression → Simple-expression

# O Integers

**Standard Integers** The standard module of Integers has also been used for specifying the SEAL typechecker and compiler [25, 27]. The Integer operators have been disambiguated from the IDEAL Expressions by prefixing them with an underscore _. The semantics of the IDEAL Expressions, which are defined in the module Ideal-expressions, can be derived from the semantics of the standard Integers. Some auxiliary operators have been renamed to prevent clashes with the IDEAL comment convention.

This specification defines Integers of infinite size which means that there is no minimum or maximum value for these Standard Integers. Those constraints are reintroduced in the module Ideal-Types.

The equations basically implement primary school arithmetic.

**imports** Layout$^{(R)}$ Booleans$^{(A)}$
**exports**
**sorts** NAT-CON NAT INT
**lexical syntax**
[*0-9*]+ → NAT-CON
**context-free syntax**
NAT-CON → NAT

NAT "_−|" NAT → NAT {**left**}
"(" NAT ")" → NAT {**bracket**}
NAT → INT
"_+" NAT → INT
"_−" NAT → INT
INT "_+" INT → INT {**left**}
INT "_−" INT → INT {**left**}
INT "_*" INT → INT {**left**}
INT "_>" INT → BOOL
INT "_>=" INT → BOOL
INT "_<" INT → BOOL
INT "_<=" INT → BOOL
"(" INT ")" → INT {**bracket**}

**variables**

*Nat-con* [*0-9'*]*→ NAT-CON
*Nat* [*0-9'*]* → NAT
"*_Int*"[*0-9'*]+ → INT

**hiddens**

**context-free syntax**

*gt* "(" NAT "," NAT ")" → BOOL
NAT "_−||" NAT → NAT

**variables**

*n*[*0-9*]*→ NAT-CON
*c*[*0-9*]*→ CHAR
*x*[*0-9*]*→ CHAR*
*y*[*0-9*]*→ CHAR+

**priorities**

{**left**: INT "_+"INT → INT, INT "_−"INT → INT} <
INT "_*"INT → INT

**equations**

[1] *nat-con*("0" *y*) = *nat-con*(*y*)

– addition (to SEAL)

[2] 0 _+ *Nat* = *Nat*
[3] *Nat* _+ 0 = *Nat*

[4] 1 _+ 1 = 2 [5] 1 _+ 2 = 3 [6] 1 _+ 3 = 4
[7] 1 _+ 4 = 5 [8] 1 _+ 5 = 6 [9] 1 _+ 6 = 7
[10] 1 _+ 7 = 8 [11] 1 _+ 8 = 9 [12] 1 _+ 9 = 10
[13] 2 _+ 1 = 3 [14] 2 _+ 2 = 4 [15] 2 _+ 3 = 5
[16] 2 _+ 4 = 6 [17] 2 _+ 5 = 7 [18] 2 _+ 6 = 8
[19] 2 _+ 7 = 9 [20] 2 _+ 8 = 10 [21] 2 _+ 9 = 11
[22] 3 _+ 1 = 4 [23] 3 _+ 2 = 5 [24] 3 _+ 3 = 6
[25] 3 _+ 4 = 7 [26] 3 _+ 5 = 8 [27] 3 _+ 6 = 9
[28] 3 _+ 7 = 10 [29] 3 _+ 8 = 11 [30] 3 _+ 9 = 12
[31] 4 _+ 1 = 5 [32] 4 _+ 2 = 6 [33] 4 _+ 3 = 7
[34] 4 _+ 4 = 8 [35] 4 _+ 5 = 9 [36] 4 _+ 6 = 10
[37] 4 _+ 7 = 11 [38] 4 _+ 8 = 12 [39] 4 _+ 9 = 13

[40] $5 \mathbin{\_+} 1 = 6$ [41] $5 \mathbin{\_+} 2 = 7$ [42] $5 \mathbin{\_+} 3 = 8$
[43] $5 \mathbin{\_+} 4 = 9$ [44] $5 \mathbin{\_+} 5 = 10$ [45] $5 \mathbin{\_+} 6 = 11$
[46] $5 \mathbin{\_+} 7 = 12$ [47] $5 \mathbin{\_+} 8 = 13$ [48] $5 \mathbin{\_+} 9 = 14$
[49] $6 \mathbin{\_+} 1 = 7$ [50] $6 \mathbin{\_+} 2 = 8$ [51] $6 \mathbin{\_+} 3 = 9$
[52] $6 \mathbin{\_+} 4 = 10$ [53] $6 \mathbin{\_+} 5 = 11$ [54] $6 \mathbin{\_+} 6 = 12$
[55] $6 \mathbin{\_+} 7 = 13$ [56] $6 \mathbin{\_+} 8 = 14$ [57] $6 \mathbin{\_+} 9 = 15$
[58] $7 \mathbin{\_+} 1 = 8$ [59] $7 \mathbin{\_+} 2 = 9$ [60] $7 \mathbin{\_+} 3 = 10$
[61] $7 \mathbin{\_+} 4 = 11$ [62] $7 \mathbin{\_+} 5 = 12$ [63] $7 \mathbin{\_+} 6 = 13$
[64] $7 \mathbin{\_+} 7 = 14$ [65] $7 \mathbin{\_+} 8 = 15$ [66] $7 \mathbin{\_+} 9 = 16$
[67] $8 \mathbin{\_+} 1 = 9$ [68] $8 \mathbin{\_+} 2 = 10$ [69] $8 \mathbin{\_+} 3 = 11$
[70] $8 \mathbin{\_+} 4 = 12$ [71] $8 \mathbin{\_+} 5 = 13$ [72] $8 \mathbin{\_+} 6 = 14$
[73] $8 \mathbin{\_+} 7 = 15$ [74] $8 \mathbin{\_+} 8 = 16$ [75] $8 \mathbin{\_+} 9 = 17$
[76] $9 \mathbin{\_+} 1 = 10$ [77] $9 \mathbin{\_+} 2 = 11$ [78] $9 \mathbin{\_+} 3 = 12$
[79] $9 \mathbin{\_+} 4 = 13$ [80] $9 \mathbin{\_+} 5 = 14$ [81] $9 \mathbin{\_+} 6 = 15$
[82] $9 \mathbin{\_+} 7 = 16$ [83] $9 \mathbin{\_+} 8 = 17$ [84] $9 \mathbin{\_+} 9 = 18$

[85] $$\frac{\begin{array}{c} \textit{nat-con}(c_1) \mathbin{\_+} \textit{nat-con}(c_2) = \textit{nat-con}(x\ c), \\ \textit{nat-con}(y_1) \mathbin{\_+} \textit{nat-con}(y_2) \mathbin{\_+} \textit{nat-con}(\texttt{"0"}\ x) = \textit{nat-con}(y) \end{array}}{\textit{nat-con}(y_1\ c_1) \mathbin{\_+} \textit{nat-con}(y_2\ c_2) = \textit{nat-con}(y\ c)}$$

[86] $$\frac{\begin{array}{c} \textit{nat-con}(c_1) \mathbin{\_+} \textit{nat-con}(c_2) = \textit{nat-con}(x\ c), \\ \textit{nat-con}(\texttt{"0"}\ x_1) \mathbin{\_+} \textit{nat-con}(y_2) \mathbin{\_+} \textit{nat-con}(\texttt{"0"}\ x) = \textit{nat-con}(y) \end{array}}{\textit{nat-con}(x_1\ c_1) \mathbin{\_+} \textit{nat-con}(y_2\ c_2) = \textit{nat-con}(y\ c)}$$

[87] $$\frac{\begin{array}{c} \textit{nat-con}(c_1) \mathbin{\_+} \textit{nat-con}(c_2) = \textit{nat-con}(x\ c), \\ \textit{nat-con}(y_1) \mathbin{\_+} \textit{nat-con}(\texttt{"0"}\ x_2) \mathbin{\_+} \textit{nat-con}(\texttt{"0"}\ x) = \textit{nat-con}(y) \end{array}}{\textit{nat-con}(y_1\ c_1) \mathbin{\_+} \textit{nat-con}(x_2\ c_2) = \textit{nat-con}(y\ c)}$$

– auxiliary function $\_{-}||$ for cut off subtraction

[88] $\textit{nat-con}(c) \mathbin{\_-||} \textit{nat-con}(c) = 0$
[89] $\textit{nat-con}(c) \mathbin{\_-||} 0 = \textit{nat-con}(c)$
[90] $2 \mathbin{\_-||} 1 = 1$
[91] $3 \mathbin{\_-||} 1 = 2$ [92] $3 \mathbin{\_-||} 2 = 1$
[93] $4 \mathbin{\_-||} 1 = 3$ [94] $4 \mathbin{\_-||} 2 = 2$ [95] $4 \mathbin{\_-||} 3 = 1$
[96] $5 \mathbin{\_-||} 1 = 4$ [97] $5 \mathbin{\_-||} 2 = 3$ [98] $5 \mathbin{\_-||} 3 = 2$
[99] $5 \mathbin{\_-||} 4 = 1$
[100] $6 \mathbin{\_-||} 1 = 5$ [101] $6 \mathbin{\_-||} 2 = 4$ [102] $6 \mathbin{\_-||} 3 = 3$
[103] $6 \mathbin{\_-||} 4 = 2$ [104] $6 \mathbin{\_-||} 5 = 1$
[105] $7 \mathbin{\_-||} 1 = 6$ [106] $7 \mathbin{\_-||} 2 = 5$ [107] $7 \mathbin{\_-||} 3 = 4$
[108] $7 \mathbin{\_-||} 4 = 3$ [109] $7 \mathbin{\_-||} 5 = 2$ [110] $7 \mathbin{\_-||} 6 = 1$
[111] $8 \mathbin{\_-||} 1 = 7$ [112] $8 \mathbin{\_-||} 2 = 6$ [113] $8 \mathbin{\_-||} 3 = 5$
[114] $8 \mathbin{\_-||} 4 = 4$ [115] $8 \mathbin{\_-||} 5 = 3$ [116] $8 \mathbin{\_-||} 6 = 2$
[117] $8 \mathbin{\_-||} 7 = 1$
[118] $9 \mathbin{\_-||} 1 = 8$ [119] $9 \mathbin{\_-||} 2 = 7$ [120] $9 \mathbin{\_-||} 3 = 6$

[121] $9 \mathbin{\_-\|} 4 = 5$ [122] $9 \mathbin{\_-\|} 5 = 4$ [123] $9 \mathbin{\_-\|} 6 = 3$

[124] $9 \mathbin{\_-\|} 7 = 2$ [125] $9 \mathbin{\_-\|} 8 = 1$

[126] $10 \mathbin{\_-\|} 1 = 9$ [127] $10 \mathbin{\_-\|} 2 = 8$ [128] $10 \mathbin{\_-\|} 3 = 7$

[129] $10 \mathbin{\_-\|} 4 = 6$ [130] $10 \mathbin{\_-\|} 5 = 5$ [131] $10 \mathbin{\_-\|} 6 = 4$

[132] $10 \mathbin{\_-\|} 7 = 3$ [133] $10 \mathbin{\_-\|} 8 = 2$ [134] $10 \mathbin{\_-\|} 9 = 1$

– auxiliary function gt

[135] $$\frac{\mathit{nat\text{-}con}(c_1) \mathbin{\_-\|} \mathit{nat\text{-}con}(c_2) = \mathit{nat\text{-}con}(c),\ \ \mathit{nat\text{-}con}(c) \neq 0}{\mathit{gt}(\mathit{nat\text{-}con}(c_1),\, \mathit{nat\text{-}con}(c_2)) = \mathit{true}}$$

[136] $\mathit{gt}(\mathit{nat\text{-}con}(y_1\ c_1),\, \mathit{nat\text{-}con}(c_2)) \ = \ \mathit{true}$

[137] $\mathit{gt}(\mathit{nat\text{-}con}(y\ c_1),\, \mathit{nat\text{-}con}(y\ c_2)) \ = \ \mathit{gt}(\mathit{nat\text{-}con}(c_1),\, \mathit{nat\text{-}con}(c_2))$

[138] $$\frac{\mathit{gt}(\mathit{nat\text{-}con}(y_1),\, \mathit{nat\text{-}con}(y_2)) = \mathit{true}}{\mathit{gt}(\mathit{nat\text{-}con}(y_1\ c_1),\, \mathit{nat\text{-}con}(y_2\ c_2)) = \mathit{true}}$$

– cut off subtraction _-|

[139] $0 \mathbin{\_-|} \mathit{Nat} \ = \ 0$

[140] $\mathit{Nat} \mathbin{\_-|} 0 \ = \ \mathit{Nat}$

[141] $$\frac{\begin{array}{c}\mathit{nat\text{-}con}(c_1) \mathbin{\_-\|} \mathit{nat\text{-}con}(c_2) = \mathit{nat\text{-}con}(c), \\ \mathit{nat\text{-}con}(\texttt{"0"}\ x_1) \mathbin{\_-|} \mathit{nat\text{-}con}(\texttt{"0"}\ x_2) = \mathit{nat\text{-}con}(y)\end{array}}{\mathit{nat\text{-}con}(x_1\ c_1) \mathbin{\_-|} \mathit{nat\text{-}con}(x_2\ c_2) = \mathit{nat\text{-}con}(y\ c)}$$

[142] $$\frac{\begin{array}{c}\mathit{nat\text{-}con}(c_2) \mathbin{\_-\|} \mathit{nat\text{-}con}(c_1) = \mathit{nat\text{-}con}(c_3), \\ \mathit{nat\text{-}con}(\texttt{"0"}\ x_2) \mathbin{\_+} 1 = \mathit{Nat}, \\ \mathit{nat\text{-}con}(y_1) \mathbin{\_-|} \mathit{Nat} = \mathit{nat\text{-}con}(y), \\ 10 \mathbin{\_-\|} \mathit{nat\text{-}con}(c_3) = \mathit{nat\text{-}con}(c)\end{array}}{\mathit{nat\text{-}con}(y_1\ c_1) \mathbin{\_-|} \mathit{nat\text{-}con}(x_2\ c_2) = \mathit{nat\text{-}con}(y\ c)}$$

[143] $$\frac{\mathit{gt}(\mathit{Nat}_1,\, \mathit{Nat}_2) \neq \mathit{true}}{\mathit{Nat}_1 \mathbin{\_-|} \mathit{Nat}_2 = 0}$$

– subtraction of naturals

[144] $\mathit{gt}(\mathit{Nat}_1,\, \mathit{Nat}_2) = \mathit{true} \ \Rightarrow \mathit{Nat}_1 \mathbin{\_-} \mathit{Nat}_2 \ = \ \mathit{Nat}_1 \mathbin{\_-|} \mathit{Nat}_2$

[145] $\mathit{gt}(\mathit{Nat}_1,\, \mathit{Nat}_2) \neq \mathit{true} \ \Rightarrow \mathit{Nat}_1 \mathbin{\_-} \mathit{Nat}_2 \ = \ \_- \mathit{Nat}_2 \mathbin{\_-|} \mathit{Nat}_1$

– multiplication of naturals –

[146] $\mathit{Nat} \mathbin{\_*} 0 \ = \ 0$

[147] $\mathit{Nat} \mathbin{\_*} 1 \ = \ \mathit{Nat}$

[148] $$\frac{\mathit{gt}(\mathit{nat\text{-}con}(c),\, 1) = \mathit{true}}{\mathit{Nat} \mathbin{\_*} \mathit{nat\text{-}con}(c) = \mathit{Nat} \mathbin{\_+} \mathit{Nat} \mathbin{\_*} (\mathit{nat\text{-}con}(c) \mathbin{\_-} 1)}$$

[149] $$\frac{nat\text{-}con(y_1) \;\_{*}\; nat\text{-}con(y_2) = nat\text{-}con(y)}{\begin{array}{l} nat\text{-}con(y_1) \;\_{*}\; nat\text{-}con(y_2\ c) = \\ nat\text{-}con(y\ \texttt{"0"}) \;\_{+}\; nat\text{-}con(y_1) \;\_{*}\; nat\text{-}con(c) \end{array}}$$

– addition, subtraction, and multiplication of integers

[150] $\_{+}\ Nat \ = \ Nat$

[151] $Nat_1 \;\_{+}\; \_{-}\ Nat_2 \quad = \quad Nat_1 \;\_{-}\; Nat_2$

[152] $\_{-}\ Nat_1 \;\_{+}\; Nat_2 \quad = \quad Nat_2 \;\_{-}\; Nat_1$

[153] $Nat_1 \;\_{+}\; Nat_2 = Nat \ \Rightarrow \_{-}\ Nat_1 \;\_{+}\; \_{-}\ Nat_2 \ = \quad \_{-}\ Nat$

[154] $Nat_1 \;\_{-}\; \_{-}\ Nat_2 \quad = \quad Nat_1 \;\_{+}\; Nat_2$

[155] $Nat_1 \;\_{+}\; Nat_2 = Nat \ \Rightarrow \_{-}\ Nat_1 \;\_{-}\; Nat_2 \ = \quad \_{-}\ Nat$

[156] $\_{-}\ Nat_1 \;\_{-}\; \_{-}\ Nat_2 \quad = \quad Nat_2 \;\_{-}\; Nat_1$

[157] $Nat_1 \;\_{*}\; Nat_2 = Nat \ \Rightarrow Nat_1 \;\_{*}\; \_{-}\ Nat_2 \ = \quad \_{-}\ Nat$

[158] $Nat_1 \;\_{*}\; Nat_2 = Nat \ \Rightarrow \_{-}\ Nat_1 \;\_{*}\; Nat_2 \ = \quad \_{-}\ Nat$

[159] $\_{-}\ Nat_1 \;\_{*}\; \_{-}\ Nat_2 \quad = \quad Nat_1 \;\_{*}\; Nat_2$

– relational operators

[160] $gt(Nat_1, Nat_2) = true \ \Rightarrow Nat_1 \;\_{>}\; Nat_2 \ = \quad true$

[161] $gt(Nat_1, Nat_2) \neq true \ \Rightarrow Nat_1 \;\_{>}\; Nat_2 \ = \quad false$

[162] $Nat_1 \;\_{>}\; \_{-}\ Nat_2 \quad = \quad true$

[163] $\_{-}\ Nat_1 \;\_{>}\; Nat_2 \quad = \quad false$

[164] $\_{-}\ Nat_1 \;\_{>}\; \_{-}\ Nat_2 \quad = \quad Nat_2 \;\_{>}\; Nat_1$

[165] $\_Int_1 \neq \_Int_2 \ \Rightarrow \_Int_1 \;\_{>=}\; \_Int_2 \ = \ \_Int_1 \;\_{>}\; \_Int_2$

[166] $\_Int_1 \;\_{>=}\; \_Int_1 \quad = \quad true$

[167] $\_Int_1 \;\_{<}\; \_Int_2 \ = \ not(\_Int_1 \;\_{>=}\; \_Int_2)$

[168] $\_Int_1 \;\_{<=}\; \_Int_2 \ = \ not(\_Int_1 \;\_{>}\; \_Int_2)$

[169] $\_{-}\ 0 \ = \ 0$

# P LSC-Type-environment

**TENV** stands for Type ENVironment; this is an auxiliary data structure for typechecking purposes. The type environment is modeled as a ternary tuple holding:

1. the telegram data fields as defined by the example section of the EURIS document. The choice to include this in TENV is made for efficiency reasons. When the Telegram-side-file gets larger, it would better not to include the Telegram-data in the type environment.

2. the parmeters declared in the Parameter-block
3. the variables and functions declared in the Variable-block

**imports** Telegram-side-file$^{(V)}$
**exports**
  **sorts** TENV
  **context-free syntax**

| | |
|---|---|
| "*Undefined-par-var-type*" | → Par-var-type |
| "*Undefined-par-type*" | → Par-type |
| "*Type-mismatch*" | → Par-var-type |
| "*No-list-type*" | → Par-var-type |
| "*No-central-type*" | → Par-var-type |
| "*Non-var-type*" | → Par-var-type |
| "[" Telegram-data-declarations "," Par-declarations "," Var-declarations "]" | → TENV |
| *telegram-data-declarations*(TENV) | → Telegram-data-declarations |
| *par-declarations*(TENV) | → Par-declarations |
| *var-declarations*(TENV) | → Var-declarations |
| *lookup-telegram*(TELENV, Identifier) | → Telegram-type |
| *lookup-telegram-data*(Telegram-data-name, Telegram-data-declarations) | → Par-var-type |
| *lookup-par*(Identifier, Par-declarations) | → Par-type |
| *lookup-var*(Identifier, Var-declarations) | → Par-var-type |
| *lookup-par-var*(TENV, Identifier) | → Par-var-type |
| *lookup*(TENV, Telegram-data-name) | → Par-var-type |
| *lookup-par-directions*(Identifier, Par-declarations) | → Dir-list |
| *lookup-var-directions*(Identifier, Var-declarations) | → Dir-list |
| *lookup-directions*(TENV, Identifier) | → Dir-list |
| *list-element-type*(Par-var-type) | → Par-var-type |

  **variables**
    *Tenv* [′]+→ TENV
**equations**

[1] $\textit{telegram-data-declarations}([\textit{Telegram-data-declarations}', \textit{Par-declarations}', \textit{Var-declarations}']) = \textit{Telegram-data-declarations}'$

[2] $\textit{par-declarations}([\textit{Telegram-data-declarations}', \textit{Par-declarations}', \textit{Var-declarations}']) = \textit{Par-declarations}'$

[3] $\textit{var-declarations}([\textit{Telegram-data-declarations}', \textit{Par-declarations}', \textit{Var-declarations}']) =$

$Var\text{-}declarations'$

[4]
$$lookup\text{-}telegram([IdealTelegramDeclarations' \; Identifier' : Telegram\text{-}type'; \; IdealTelegramDeclarations''], Identifier') = Telegram\text{-}type'$$

[5] $lookup\text{-}telegram(Telenv', Identifier') =$
$Undefined\text{-}telegram\text{-}name$ **otherwise**

[6] $lookup\text{-}telegram\text{-}data(Telegram\text{-}data\text{-}name', Telegram\text{-}data\text{-}declarations') =$
$INT\_TYPE$

[7]
$$lookup\text{-}par(Identifier', Par\text{-}declarations' \; Identifier' : Par\text{-}type'; \; Par\text{-}declarations'') = Par\text{-}type'$$

[8]
$$lookup\text{-}par(Identifier', Par\text{-}declarations' \; Identifier'(Dir\text{-}list') : Par\text{-}type'; \; Par\text{-}declarations'') = Par\text{-}type'$$

[9] $lookup\text{-}par(Identifier', Par\text{-}declarations') =$
$Undefined\text{-}par\text{-}type$ **otherwise**

[10]
$$lookup\text{-}var(Identifier', Var\text{-}declarations' \; Identifier' : Par\text{-}var\text{-}type'; \; Var\text{-}declarations'') = Par\text{-}var\text{-}type'$$

[11]
$$lookup\text{-}var(Identifier', Var\text{-}declarations' \; Identifier' \; Dir\text{-}ini\text{-}list' : Par\text{-}var\text{-}type'; \; Var\text{-}declarations'') = Par\text{-}var\text{-}type'$$

[12]
$lookup\text{-}var(Identifier', Var\text{-}declarations') = Undefined\text{-}par\text{-}var\text{-}type$ **otherwise**

[13]
$$\frac{lookup\text{-}par(Identifier', par\text{-}declarations(Tenv')) = Undefined\text{-}par\text{-}type}{lookup\text{-}par\text{-}var(Tenv', Identifier') = lookup\text{-}var(Identifier', var\text{-}declarations(Tenv'))}$$

[14] $lookup\text{-}par\text{-}var(Tenv', Identifier') =$
$lookup\text{-}par(Identifier', par\text{-}declarations(Tenv'))$ **otherwise**

[15]
$$lookup\text{-}par\text{-}directions(Identifier', Par\text{-}declarations' \; Identifier'(Dir\text{-}list') : Par\text{-}type'; \; Par\text{-}declarations'') =$$
$Dir\text{-}list'$

[16] $lookup\text{-}par\text{-}directions(Identifier', Par\text{-}declarations') =$
$Empty\text{-}directions$ **otherwise**

[17] $\textit{lookup-var-directions}(\textit{Identifier}', \textit{Var-declarations}'\ \textit{Identifier}'\ \textit{Dir-ini-list}' : \textit{Par-var-type}';\ \textit{Var-declarations}'') =$
$\textit{directions}(\textit{Dir-ini-list}')$

[18] $\textit{lookup-var-directions}(\textit{Identifier}', \textit{Var-declarations}') =$
$\textit{Empty-directions}$ **otherwise**

[19] $$\frac{\textit{lookup-par-directions}(\textit{Identifier}', \textit{par-declarations}(\textit{Tenv}')) = \textit{Empty-directions}}{\textit{lookup-directions}(\textit{Tenv}', \textit{Identifier}') = \textit{lookup-var-directions}(\textit{Identifier}', \textit{var-declarations}(\textit{Tenv}'))}$$

[20] $\textit{lookup-directions}(\textit{Tenv}', \textit{Identifier}') =$
$\textit{lookup-par-directions}(\textit{Identifier}', \textit{par-declarations}(\textit{Tenv}'))$ **otherwise**

[21] $\textit{list-element-type}(\textit{BOOL\_LIST\_TYPE}) = \textit{BOOL\_TYPE}$
[22] $\textit{list-element-type}(\textit{INT\_LIST\_TYPE}) = \textit{INT\_TYPE}$
[23] $\textit{list-element-type}(\textit{Par-var-type}') = \textit{No-list-type}$ **otherwise**

# Q LSC-mappings

**OOP** This module mimicks the Object Oriented Programming style. Projections can also be viewed upon as object field access 'methods'.

**imports** Ideal-LSC$^{(\mathrm{L})}$
**exports**
**context-free syntax**
*program-start-block*(Ideal-program) → Program-start-block
*configuration-block*(Ideal-program) → Configuration-block
*parameter-block*(Ideal-program) → Parameter-block
*variable-block*(Ideal-program) → Variable-block
*function-blocks*(Ideal-program) → Function-blocks
*merge-blocks*(Ideal-program) → Merge-blocks
*flow-blocks*(Ideal-program) → Flow-blocks
*program-end-block*(Ideal-program) → Program-end-block

*parameter-declarations*(Parameter-block) → Par-declarations

*variable-declarations*(Variable-block) → Var-declarations

*name*(Function-block) → Identifier
*function-statements*(Function-block) → Statements
*function-merge-blocks*(Function-block) → Merge-blocks

*initiator*(Flow-block) → Flow-initiator
*flow-statements*(Flow-block) → Statements

*number*(Merge-block) → Label

$merge\text{-}statements$(Merge-block) → Statements

$set\text{-}expr$(Statement) → Set-expression

$par\text{-}var\text{-}exp$(Set-expression) → Par-var-exp
$var\text{-}attribute$(Par-var-exp) → Var-attribute
$directions$(Var-attribute) → Directions

$directions$(Dir-ini-list) → Directions
"*Empty-directions*" → Directions

**variables**

*Ideal-program* [′]+ → Ideal-program
*Program-start-block* [′]+ → Program-start-block
*Program-end-block* [′]+ → Program-end-block
*Configuration-block* [′]+ → Configuration-block
*Parameter-block* [′]+ → Parameter-block
*Variable-block* [′]+ → Variable-block

*Function-block* [′]+ → Function-block
*Function-blocks* [′]+ → Function-block∗
*Merge-block* [′]+ → Merge-block
*Merge-blocks* [′]+ → Merge-block∗
*Merge-block* [′]+ → Merge-block
*Merge-blocks* [′]+ → Merge-block∗
*Flow-block* [′]+ → Flow-block
*Flow-blocks* [′]+ → Flow-block∗

*Letter* [′]+ → Letter
*Digit* [′]+ → DIGIT
*Number* [′]+ → Number
*Letter-or-digit* [′]+ → Letter-or-digit
*Label* [′]+ → Label
*Labels* [′]+ → {Label ","}∗
*String* [′]+ → String
*Par-var-exp* [′]+ → Par-var-exp
*Identifier* [′]+ → Identifier

*Simple-expression* [′]+ → Simple-expression
*Var-attribute* [′]+ → Var-attribute
*Direction* [′]+ → Direction
*Directions* [′]+ → Direction+
*Initiator* [′]+ → Initiator
*Flow-initiator* [′]+ → Flow-initiator
*Ini-list* [′]+ → Ini-list
*Dir-list* [′]+ → {Directions ","}+
*Dir-ini-list* [′]+ → Dir-ini-list

*Telegram-in-type* [′]+ → Telegram-in-type
*Telegram-output-type* [′]+ → Telegram-output-type

*Simple-expression* [′]+ → Simple-expression
*Test-expression* [′]+ → Test-expression
*Set-expression* [′]+ → Set-expression

*Par-declaration* [′]+ → Par-declaration
*Var-declaration* [′]+ → Var-declaration

$$
\begin{array}{ll}
\textit{Config-declaration}\ [']+ & \rightarrow \text{Config-declaration} \\
\textit{Par-declarations}\ [']+ & \rightarrow \text{Par-declaration}* \\
\textit{Var-declarations}\ [']+ & \rightarrow \text{Var-declaration}* \\
\textit{Central-parameter-list}\ [']+ & \rightarrow \text{Central-parameter-list} \\
\\
\textit{Statement}\ [']+ & \rightarrow \text{Statement} \\
\textit{Statements}\ [']+ & \rightarrow \text{Statement}* \\
\\
\textit{Par-tag}\ [']+ & \rightarrow \text{Par-tag} \\
\textit{Var-tag}\ [']+ & \rightarrow \text{Var-tag} \\
\textit{Par-var-tag}\ [']+ & \rightarrow \text{Par-var-tag} \\
\\
\textit{R-value}\ [']+ & \rightarrow \text{R-value}
\end{array}
$$

**equations**

$$
\begin{array}{l}
\textit{program-start-block}(\textit{Program-start-block}' \\
\quad \textit{Configuration-block}' \\
\quad \textit{Parameter-block}' \\
\quad \textit{Variable-block}' \\
\quad \textit{Function-blocks}' \\
\quad \textit{Merge-blocks}' \\
\quad \textit{Flow-blocks}' \\
\quad \textit{Program-end-block}') = \textit{Program-start-block}'
\end{array} \qquad [1]
$$

$$
\begin{array}{l}
\textit{configuration-block}(\textit{Program-start-block}' \\
\quad \textit{Configuration-block}' \\
\quad \textit{Parameter-block}' \\
\quad \textit{Variable-block}' \\
\quad \textit{Function-blocks}' \\
\quad \textit{Merge-blocks}' \\
\quad \textit{Flow-blocks}' \\
\quad \textit{Program-end-block}') = \textit{Configuration-block}'
\end{array} \qquad [2]
$$

$$
\begin{array}{l}
\textit{parameter-block}(\textit{Program-start-block}' \\
\quad \textit{Configuration-block}' \\
\quad \textit{Parameter-block}' \\
\quad \textit{Variable-block}' \\
\quad \textit{Function-blocks}' \\
\quad \textit{Merge-blocks}' \\
\quad \textit{Flow-blocks}' \\
\quad \textit{Program-end-block}') = \textit{Parameter-block}'
\end{array} \qquad [3]
$$

$$
\begin{array}{l}
\textit{variable-block}(\textit{Program-start-block}' \\
\quad \textit{Configuration-block}' \\
\quad \textit{Parameter-block}' \\
\quad \textit{Variable-block}' \\
\quad \textit{Function-blocks}' \\
\quad \textit{Merge-blocks}' \\
\quad \textit{Flow-blocks}' \\
\quad \textit{Program-end-block}') = \textit{Variable-block}'
\end{array} \qquad [4]
$$

[5]
$$
\begin{aligned}
&\mathit{function\text{-}blocks}(\mathit{Program\text{-}start\text{-}block}' \\
&\quad \mathit{Configuration\text{-}block}' \\
&\quad \mathit{Parameter\text{-}block}' \\
&\quad \mathit{Variable\text{-}block}' \\
&\quad \mathit{Function\text{-}blocks}' \\
&\quad \mathit{Merge\text{-}blocks}' \\
&\quad \mathit{Flow\text{-}blocks}' \\
&\quad \mathit{Program\text{-}end\text{-}block}') = \mathit{Function\text{-}blocks}'
\end{aligned}
$$

[6]
$$
\begin{aligned}
&\mathit{merge\text{-}blocks}(\mathit{Program\text{-}start\text{-}block}' \\
&\quad \mathit{Configuration\text{-}block}' \\
&\quad \mathit{Parameter\text{-}block}' \\
&\quad \mathit{Variable\text{-}block}' \\
&\quad \mathit{Function\text{-}blocks}' \\
&\quad \mathit{Merge\text{-}blocks}' \\
&\quad \mathit{Flow\text{-}blocks}' \\
&\quad \mathit{Program\text{-}end\text{-}block}') = \mathit{Merge\text{-}blocks}'
\end{aligned}
$$

[7]
$$
\begin{aligned}
&\mathit{flow\text{-}blocks}(\mathit{Program\text{-}start\text{-}block}' \\
&\quad \mathit{Configuration\text{-}block}' \\
&\quad \mathit{Parameter\text{-}block}' \\
&\quad \mathit{Variable\text{-}block}' \\
&\quad \mathit{Function\text{-}blocks}' \\
&\quad \mathit{Merge\text{-}blocks}' \\
&\quad \mathit{Flow\text{-}blocks}' \\
&\quad \mathit{Program\text{-}end\text{-}block}') = \mathit{Flow\text{-}blocks}'
\end{aligned}
$$

[8]
$$
\begin{aligned}
&\mathit{program\text{-}end\text{-}block}(\mathit{Program\text{-}start\text{-}block}' \\
&\quad \mathit{Configuration\text{-}block}' \\
&\quad \mathit{Parameter\text{-}block}' \\
&\quad \mathit{Variable\text{-}block}' \\
&\quad \mathit{Function\text{-}blocks}' \\
&\quad \mathit{Merge\text{-}blocks}' \\
&\quad \mathit{Flow\text{-}blocks}' \\
&\quad \mathit{Program\text{-}end\text{-}block}') = \mathit{Program\text{-}end\text{-}block}'
\end{aligned}
$$

[9] $\mathit{parameter\text{-}declarations}() =$

[10] $\mathit{parameter\text{-}declarations}(\mathit{LSC\_PAR}\ \mathit{Par\text{-}declarations}') = \mathit{Par\text{-}declarations}'$

[11] $\mathit{variable\text{-}declarations}() =$

[12] $\mathit{variable\text{-}declarations}(\mathit{LSC\_VAR}\ \mathit{Var\text{-}declarations}') = \mathit{Var\text{-}declarations}'$

[13]
$$
\begin{aligned}
&\mathit{name}(\mathit{LSC\_FUNCTION}\ \mathit{Identifier}'; \\
&\quad \mathit{Statements}' \\
&\quad \mathit{Merge\text{-}blocks}'\ \mathit{LSC\_FUNCTION\_END};) = \\
&\mathit{Identifier}'
\end{aligned}
$$

[14]
$$
\begin{aligned}
&\mathit{function\text{-}statements}(\mathit{LSC\_FUNCTION}\ \mathit{Identifier}'; \\
&\quad \mathit{Statements}' \\
&\quad \mathit{Merge\text{-}blocks}'\ \mathit{LSC\_FUNCTION\_END};) = \\
&\mathit{Statements}'
\end{aligned}
$$

[15]
$$
\begin{aligned}
&\mathit{function\text{-}merge\text{-}blocks}(\mathit{LSC\_FUNCTION}\ \mathit{Identifier}'; \\
&\quad \mathit{Statements}' \\
&\quad \mathit{Merge\text{-}blocks}'\ \mathit{LSC\_FUNCTION\_END};) =
\end{aligned}
$$

Merge-blocks'

[16] number(LSC_MERGE␣LABEL_ Label';
Statements' LSC_MERGE_END;) =
Label'

[17] merge-statements(LSC_MERGE␣LABEL_ Label';
Statements' LSC_MERGE_END;) =
Statements'

[18] initiator(LSC_FLOW Flow-initiator';
Statements' LSC_FLOW_END;) =
Flow-initiator'

[19] initiator(LSC_CNT_FLOW Flow-initiator'
Central-parameter-list';
Statements' LSC_CNT_FLOW_END;) =
Flow-initiator'

[20] flow-statements(LSC_FLOW Flow-initiator';
Statements' LSC_FLOW_END;) =
Statements'

[21] flow-statements(LSC_CNT_FLOW Flow-initiator'
Central-parameter-list';
Statements' LSC_CNT_FLOW_END;) =
Statements'

[22] set-expr(LSC_SET Set-expression';) = Set-expression'

[23] par-var-exp(Par-var-exp' : R-value') = Par-var-exp'

[24] var-attribute(Identifier'(Var-attribute')) = Var-attribute'

[25] var-attribute(Identifier'(Var-attribute')
[Simple-expression']) = Var-attribute'

[26] var-attribute( V. Identifier'(Var-attribute')) = Var-attribute'

[27] var-attribute( V. Identifier'(Var-attribute')
[Simple-expression']) = Var-attribute'

[28] directions(Directions', Initiator') = Directions'
[29] directions(Directions') = Directions'

[30] directions((Dir-list', Ini-list')) = Dir-list'
[31] directions((Dir-list')) = Dir-list'
[32] directions((Ini-list')) = Empty-directions

# R Layout

**Comment Conventions** This module supports both IDEAL line comment conventions. IDEAL has two nested block comment conventions as well. In ASF+SDF, however, nested comments cannot be specified due to the built-in lexical scanning algorithm. For debugging purposes the same effect can be achieved by either writing a block macro in GNU Emacs that either inserts or extract line-comments or by extensively cut & paste techniques. The percentage line commentary convention is native to SDF.

**exports**
  **lexical syntax**
    [␣\t\n] → LAYOUT
    "%%"~[\n]*[\n] → LAYOUT
    "−−"~[\n]*[\n] → LAYOUT
    "//"~[\n]*[\n] → LAYOUT

# S Parameter-side-file

**The Parameter side file** still contains many gaps since not enough information can be drawn from the EURIS documentation for this purpose.

**imports** LSC-mappings$^{(Q)}$ Errors$^{(J)}$
**exports**
  **sorts** PENV
  **context-free syntax**
    "[" Par-declarations "]" → PENV

    *double-check-par-declarations*(PENV, Par-declarations) → ERROR-LIST
    *double-check-par*(PENV, Par-declaration) → ERROR-LIST

    "*IdealParameters*" → PENV
    "*Parameter-in*" *par-declaration* Par-declaration
    yet undefined in parameter-side-file → ERROR
  **variables**
    *Penv* [′]+→ PENV
**hiddens**
  **context-free syntax**
    *dummy-par-type* → Par-type
**equations**

[1] $\textit{double-check-par-declarations}(\textit{Penv}', ) =$

$\textit{double-check-par-declarations}(\textit{Penv}',$
[2] $\textit{Par-declaration}'\ \textit{Par-declarations}') =$
$\textit{join-errors}(\textit{double-check-par}(\textit{Penv}', \textit{Par-declaration}'),$
$\textit{double-check-par-declarations}(\textit{Penv}', \textit{Par-declarations}'))$

$\textit{double-check-par}([\textit{Par-declarations}'$
$\textit{Identifier}' : \textit{Par-type}';$
$\textit{Par-declarations}''],$
[3] $\textit{Identifier}' : \textit{Par-type}'';) =$

$double\text{-}check\text{-}par([Par\text{-}declarations'$
$Identifier' : Par\text{-}type';$
$Par\text{-}declarations''],$
[4] $Identifier'(Dir\text{-}list') : Par\text{-}type'';) =$

[5] $double\text{-}check\text{-}par(Penv', Par\text{-}declaration') =$
*Parameter-in par-declaration* $Par\text{-}declaration'$ *yet undefined in parameter-side-file* **otherwise**

This side file is filled with dummies since at this stage it is not perfectly clear which type convention should be chosen for.

[6] *IdealParameters* =

[`ASS` : *dummy-par-type*; `CBP` : *dummy-par-type*;
`CPP` : *dummy-par-type*; `CPS` : *dummy-par-type*;
`DT1` : *dummy-par-type*; `DT2` : *dummy-par-type*;
`DT3` : *dummy-par-type*; `FIJ` : *dummy-par-type*;
`FIT` : *dummy-par-type*; `MSL` : *dummy-par-type*;
`MSR` : *dummy-par-type*; `MSS` : *dummy-par-type*;
`MSW1` : *dummy-par-type*; `MSW2` : *dummy-par-type*;
`NIJ` : *dummy-par-type*; `PRI` : *dummy-par-type*;
`PSPS` : *dummy-par-type*; `SNR` : *dummy-par-type*;
`SRR` : *dummy-par-type*; `WNR` : *dummy-par-type*;
`XPN` : *dummy-par-type*;]

# T Range

**Range Checking** IDEAL Types are effectively all of type range. The best way to deal with type checking in IDEAL should be a *range check* rather than the subclassification in artificially introduced types BOOL and ENUM. Here they are only pursued as an experiment.

Note the usage of the Standard Integer Relational operators in the equations rather than the IDEAL relational operators which are defined in the module Ideal-expressions.

**imports** Integers$^{(O)}$
**exports**
**sorts** Range
**context-free syntax**
"[" INT "," INT "]" → Range
*min-value*(Range) → INT
*max-value*(Range) → INT
*value-in-range*(INT, Range) → BOOL
**variables**
*Range* [′]+→ Range
*Int* [′]+ → INT
**equations**

[1] $min\text{-}value([Int', Int'']) \ = \ Int'$
[2] $max\text{-}value([Int', Int'']) \ = \ Int''$

[3] $value\text{-}in\text{-}range(Int', Range') =$
$Int' \ \_\!\!>= min\text{-}value(Range') \ and \ Int' \ \_\!\!<= max\text{-}value(Range')$

# U TC-LSC

**TC-LSC** is the top module of all static semantic checks specified for IDEAL LSC Programs. It returns "No-Errors-found" when it fails to find errors. As mentioned before, the tests are far from complete and only suggest a possible route for proving the correctness of the translation from LSCs to IDEAL.

**imports** Check-types$^{(I)}$ Check-labels$^{(G)}$ Check-fb-sends$^{(E)}$ Check-directions$^{(C)}$
Check-LSC-structure$^{(B)}$

**exports**

**context-free syntax**

$\mathit{check\text{-}all}(\text{Ideal-program}) \quad \rightarrow \text{ERROR-LIST}$

"$*_\sqcup*_\sqcup*_\sqcup \mathit{No\text{-}Errors\text{-}found}_\sqcup{:}\ -)$" $\rightarrow$ ERROR

**equations**

$$[1]\ \mathit{check\text{-}all}(\mathit{Ideal\text{-}program}') = \\ \mathit{join\text{-}errors}(\mathit{check\text{-}lsc\text{-}struct}(\mathit{Ideal\text{-}program}'), \\ \mathit{join\text{-}errors}(\mathit{check\text{-}types}(\mathit{Ideal\text{-}program}'), \\ \mathit{join\text{-}errors}(\mathit{check\text{-}labels}(\mathit{Ideal\text{-}program}'), \\ \mathit{join\text{-}errors}(\mathit{check\text{-}sends}(\mathit{flow\text{-}blocks}(\mathit{Ideal\text{-}program}')), \\ \mathit{join\text{-}errors}(\mathit{check\text{-}directions}(\mathit{Ideal\text{-}program}'), \\ *_\sqcup*_\sqcup*_\sqcup \mathit{No\text{-}Errors\text{-}found}_\sqcup{:}\ -))))))$$

$$[2]\ \frac{\mathit{Error\text{-}list}' = \ , \quad \mathit{Error\text{-}list}' = \mathit{Error\text{-}list}'}{\mathit{join\text{-}errors}(\mathit{Error\text{-}list}', *_\sqcup*_\sqcup*_\sqcup \mathit{No\text{-}Errors\text{-}found}_\sqcup{:}\ -)) = *_\sqcup*_\sqcup*_\sqcup \mathit{No\text{-}Errors\text{-}found}_\sqcup{:}\ -)}$$

$$[3]\ \frac{\mathit{Error\text{-}list}' \neq \ , \quad \mathit{Error\text{-}list}' = \mathit{Error\text{-}list}'}{\mathit{join\text{-}errors}(\mathit{Error\text{-}list}', *_\sqcup*_\sqcup*_\sqcup \mathit{No\text{-}Errors\text{-}found}_\sqcup{:}\ -)) = \mathit{Error\text{-}list}'}$$

# V Telegram-side-file

**The Telegram side file** has been modeled as a specification as to evade the communication overhead when putting the Telegram data in a separate file.

This side file is a first step toward range checking that should be performed on all parameters, variables and telegram data fields.

The sort `TELENV` stands for Telegram Environment.

**imports** LSC-mappings$^{(Q)}$ Errors$^{(J)}$

**exports**

**sorts** Telegram-data-declaration Telegram-data-declarations
Telegram-type IdealTelegramDeclaration
IdealTelegramDeclarations TELENV

**context-free syntax**

| | |
|---|---|
| Telegram-data-name ":" Range ";" | → Telegram-data-declaration |
| Telegram-data-declaration* | → Telegram-data-declarations |
| "*IdealTelegramData*" | → Telegram-data-declarations |
| Identifier ":" Telegram-type ";" | → IdealTelegramDeclaration |
| IdealTelegramDeclaration* | → IdealTelegramDeclarations |
| "[" IdealTelegramDeclarations "]" | → TELENV |
| "*IdealTelegrams*" | → TELENV |
| *check-telegram-unicity*(TELENV) | → ERROR-LIST |
| *check-tel-dat-range*(Telegram-data-declarations, Telegram-data-name, Simple-expression) | → ERROR-LIST |
| *check-tel-dat-int-range*(Telegram-data-declarations, Telegram-data-name, INT) | → ERROR-LIST |
| "CENTRAL" | → Telegram-type |
| "HORIZONTAL" | → Telegram-type |
| "VERTICAL" | → Telegram-type |
| "*dummy-tel-type*" | → Telegram-type |
| "*Undefined-telegram-name*" | → Telegram-type |
| *telegram-type*(Telegram-in-type) | → Telegram-type |
| *range-of-tel-data*(Telegram-data-name, Telegram-data-declarations) | → Range |
| *redeclaration of* "*Telegram*" ":" Identifier in-Telegram-side-file | → ERROR |
| *assignment of* "*Telegram*" *data* Telegram-data-name *with* Simple-expression out of bounds Range | → ERROR |

**variables**

| | |
|---|---|
| *Telenv* [′]+ | → TELENV |
| *Telegram-data-name* [′]+ | → Telegram-data-name |
| *Identifier* [′]+ | → Identifier |
| *Telegram-type* [′]+ | → Telegram-type |
| *Telegram-data-declaration* [′]+ | → Telegram-data-declaration |
| *Telegram-data-declarations* [′]+ | → Telegram-data-declaration* |
| *IdealTelegramDeclaration* [′]+ | → IdealTelegramDeclaration |
| *IdealTelegramDeclarations* [′]+ | → IdealTelegramDeclaration* |

**equations**

[1] *IdealTelegramData* =

`B` : [0, 1]; `AM` : [0, 2]; `C` : [0, 2]; `F` : [0, 1];
`NS` : [0, *MAXINT*]; `NW` : [0, *MAXINT*]; `TP` : [0, 1]; `RL` : [0, 1];
`RT` : [0, 3]; `SA` : [0, 4]; `SC` : [0, 1]; `SI` : [*MININT*, *MAXINT*];
`ST` : [0, 1]; `TD` : [0, 1]; `TS` : [0, 1]; `VA` : [*MININT*, *MAXINT*];
`VP` : [*MININT*, *MAXINT*]; `VT` : [*MININT*, *MAXINT*]; `VW` : [*MININT*, *MAXINT*]; `WD` : [0, 1];
`XT` : [0, 1];

$$
\begin{array}{ll}
 & \mathit{range\text{-}of\text{-}tel\text{-}data}(\mathit{Telegram\text{-}data\text{-}name}', \\
 & \qquad \mathit{Telegram\text{-}data\text{-}declarations}' \\
 & \qquad \mathit{Telegram\text{-}data\text{-}name}' : \mathit{Range}'; \\
\text{[2]} & \qquad \mathit{Telegram\text{-}data\text{-}declarations}'') \quad = \mathit{Range}'
\end{array}
$$

[3] $\mathit{range\text{-}of\text{-}tel\text{-}data}(\mathit{Telegram\text{-}data\text{-}name}', \mathit{Telegram\text{-}data\text{-}declarations}') =$
$[\mathit{MININT}, \mathit{MAXINT}]$ **otherwise**

When range unknown the user gets notified for missing declaration elsewhere

[4] $\mathit{telegram\text{-}type}(\mathit{LALR}) = \mathit{HORIZONTAL}$
[5] $\mathit{telegram\text{-}type}(\mathit{LBLR}) = \mathit{HORIZONTAL}$
[6] $\mathit{telegram\text{-}type}(\mathit{LCLR}) = \mathit{HORIZONTAL}$
[7] $\mathit{telegram\text{-}type}(\mathit{LDLR}) = \mathit{HORIZONTAL}$
[8] $\mathit{telegram\text{-}type}(\mathit{LARL}) = \mathit{HORIZONTAL}$
[9] $\mathit{telegram\text{-}type}(\mathit{LBRL}) = \mathit{HORIZONTAL}$
[10] $\mathit{telegram\text{-}type}(\mathit{LCRL}) = \mathit{HORIZONTAL}$
[11] $\mathit{telegram\text{-}type}(\mathit{LDRL}) = \mathit{HORIZONTAL}$
[12] $\mathit{telegram\text{-}type}(\mathit{MMI}) = \mathit{VERTICAL}$
[13] $\mathit{telegram\text{-}type}(\mathit{TRK}) = \mathit{VERTICAL}$
[14] $\mathit{telegram\text{-}type}(\mathit{CNT\_UP}) = \mathit{CENTRAL}$
[15] $\mathit{telegram\text{-}type}(\mathit{CNT\_DOWN}) = \mathit{CENTRAL}$

[16] $\mathit{IdealTelegrams} =$

$$
\begin{array}{l}
[\mathtt{A01} : \mathit{HORIZONTAL};\ \mathtt{A02} : \mathit{HORIZONTAL};\ \mathtt{A03} : \mathit{HORIZONTAL};\\
\ \mathtt{A04} : \mathit{HORIZONTAL};\ \mathtt{A05} : \mathit{HORIZONTAL};\ \mathtt{A06} : \mathit{HORIZONTAL};\\
\ \mathtt{A07} : \mathit{HORIZONTAL};\ \mathtt{B01} : \mathit{HORIZONTAL};\ \mathtt{B02} : \mathit{HORIZONTAL};\\
\ \mathtt{B03} : \mathit{HORIZONTAL};\ \mathtt{B04} : \mathit{HORIZONTAL};\ \mathtt{B05} : \mathit{HORIZONTAL};\\
\ \mathtt{B06} : \mathit{HORIZONTAL};\ \mathtt{B07} : \mathit{HORIZONTAL};\ \mathtt{C01} : \mathit{HORIZONTAL};\\
\ \mathtt{C02} : \mathit{HORIZONTAL};\ \mathtt{C03} : \mathit{HORIZONTAL};\ \mathtt{D01} : \mathit{HORIZONTAL};\\
\ \mathtt{D02} : \mathit{HORIZONTAL};\ \mathtt{D03} : \mathit{HORIZONTAL};\ \mathtt{D04} : \mathit{HORIZONTAL};\\
\ \mathtt{I01} : \mathit{HORIZONTAL};\ \mathtt{I02} : \mathit{HORIZONTAL};\ \mathtt{I03} : \mathit{HORIZONTAL};\\
\ \mathtt{I04} : \mathit{HORIZONTAL};\ \mathtt{I05} : \mathit{HORIZONTAL};\ \mathtt{M01} : \mathit{VERTICAL};\\
\ \mathtt{M02} : \mathit{VERTICAL};\ \mathtt{M03} : \mathit{VERTICAL};\ \mathtt{M04} : \mathit{VERTICAL};\\
\ \mathtt{M05} : \mathit{VERTICAL};\ \mathtt{N01} : \mathit{VERTICAL};\ \mathtt{N02} : \mathit{VERTICAL};\\
\ \mathtt{N03} : \mathit{VERTICAL};\ \mathtt{N04} : \mathit{VERTICAL};\ \mathtt{N05} : \mathit{VERTICAL};\\
\ \mathtt{N06} : \mathit{VERTICAL};\ \mathtt{N07} : \mathit{VERTICAL};\ \mathtt{N08} : \mathit{VERTICAL};\\
\ \mathtt{N09} : \mathit{VERTICAL};\ \mathtt{N10} : \mathit{VERTICAL};\ \mathtt{N11} : \mathit{VERTICAL};\\
\ \mathtt{N12} : \mathit{VERTICAL};\ \mathtt{N13} : \mathit{VERTICAL};\ \mathtt{N14} : \mathit{VERTICAL};\\
\ \mathtt{N15} : \mathit{VERTICAL};\ \mathtt{N16} : \mathit{VERTICAL};\ \mathtt{N17} : \mathit{VERTICAL};\\
\ \mathtt{N18} : \mathit{VERTICAL};\ \mathtt{N19} : \mathit{VERTICAL};\ \mathtt{N20} : \mathit{VERTICAL};\\
\ \mathtt{N21} : \mathit{VERTICAL};\ \mathtt{N22} : \mathit{VERTICAL};\ \mathtt{N23} : \mathit{VERTICAL};\\
\ \mathtt{N24} : \mathit{VERTICAL};\ \mathtt{N25} : \mathit{VERTICAL};\ \mathtt{N26} : \mathit{VERTICAL};\\
\ \mathtt{N27} : \mathit{VERTICAL};\ \mathtt{N28} : \mathit{VERTICAL};\ \mathtt{N29} : \mathit{VERTICAL};\\
\ \mathtt{N30} : \mathit{VERTICAL};\ \mathtt{N31} : \mathit{VERTICAL};\ \mathtt{N32} : \mathit{VERTICAL};\\
\ \mathtt{N33} : \mathit{VERTICAL};\ \mathtt{N34} : \mathit{VERTICAL};\ \mathtt{N35} : \mathit{VERTICAL};\\
\ \mathtt{N36} : \mathit{VERTICAL};\ \mathtt{N37} : \mathit{VERTICAL};\ \mathtt{N38} : \mathit{VERTICAL};\\
\ \mathtt{N39} : \mathit{VERTICAL};\ \mathtt{P01} : \mathit{VERTICAL};\ \mathtt{P02} : \mathit{VERTICAL};\\
\ \mathtt{P03} : \mathit{VERTICAL};\ \mathtt{P04} : \mathit{VERTICAL};\ \mathtt{P05} : \mathit{VERTICAL};\\
\ \mathtt{P06} : \mathit{VERTICAL};\ \mathtt{P07} : \mathit{VERTICAL};\ \mathtt{T01} : \mathit{CENTRAL};\\
\ \mathtt{U01} : \mathit{CENTRAL};\ \mathtt{W01} : \mathit{VERTICAL};\ \mathtt{W02} : \mathit{VERTICAL};\\
\ \mathtt{W03} : \mathit{VERTICAL};\ \mathtt{W04} : \mathit{VERTICAL};\ \mathtt{W05} : \mathit{VERTICAL};\\
\ \mathtt{W06} : \mathit{VERTICAL};\ \mathtt{W07} : \mathit{VERTICAL};\ \mathtt{X01} : \mathit{VERTICAL};\\
\ \mathtt{X02} : \mathit{VERTICAL};\ \mathtt{X03} : \mathit{VERTICAL};\ \mathtt{X04} : \mathit{VERTICAL};\\
\ \mathtt{X05} : \mathit{VERTICAL};\ \mathtt{X06} : \mathit{VERTICAL};]
\end{array}
$$

$$
\begin{array}{l}
\textit{check-telegram-unicity}([\textit{IdealTelegramDeclarations}'\\
\qquad\qquad\qquad \textit{Identifier}' : \textit{Telegram-type}';\\
\qquad\qquad\qquad \textit{IdealTelegramDeclarations}''\\
\qquad\qquad\qquad \textit{Identifier}' : \textit{Telegram-type}'';\\
\qquad\qquad\qquad \textit{IdealTelegramDeclarations}''']) =
\end{array}
\quad [17]
$$

$$
\textit{redeclaration of Telegram} : \mathrm{Identifier}'\ \textit{in-Telegram-side-file}
$$

[18] $\textit{check-telegram-unicity}(\mathrm{Telenv}') = \qquad$ **otherwise**

$$
\begin{array}{l}
\textit{check-tel-dat-range}(\mathrm{Telegram\text{-}data\text{-}declarations}',\\
\qquad\qquad \mathrm{Telegram\text{-}data\text{-}name}',\ \mathrm{Number}') =
\end{array}
\quad [19]
$$

$$
\begin{array}{l}
\textit{check-tel-dat-int-range}(\mathrm{Telegram\text{-}data\text{-}declarations}',\\
\qquad\qquad \mathrm{Telegram\text{-}data\text{-}name}',\\
\qquad\qquad \textit{simple-expression-to-int}(\mathrm{Number}'))
\end{array}
$$

$$
\begin{array}{l}
\textit{check-tel-dat-range}(\mathrm{Telegram\text{-}data\text{-}declarations}',\\
\qquad\qquad \mathrm{Telegram\text{-}data\text{-}name}',\ \mathrm{Simple\text{-}expression}') =
\end{array}
\quad [20]
$$

**otherwise**

$$[21]\quad \frac{\mathit{value\text{-}in\text{-}range}(\mathit{Int}', \mathit{range\text{-}of\text{-}tel\text{-}data}(\mathit{Telegram\text{-}data\text{-}name}', \mathit{Telegram\text{-}data\text{-}declarations}')) = \mathit{true}}{\mathit{check\text{-}tel\text{-}dat\text{-}int\text{-}range}(\mathit{Telegram\text{-}data\text{-}declarations}', \mathit{Telegram\text{-}data\text{-}name}', \mathit{Int}') \; =}$$

[22] *check-tel-dat-int-range*(*Telegram-data-declarations'*, *Telegram-data-name'*, *Int'*) =
*assignment of Telegram data Telegram-data-name'* **otherwise**
*with int-to-simple-expression*(*Int'*)
*out of bounds range-of-tel-data*(*Telegram-data-name'*, *Telegram-data-declarations'*)

# W Variable-side-file

**The Variable side file** still contains many gaps since not enough information can be drawn from the EURIS documentation for this purpose.

**imports** LSC-mappings$^{(Q)}$ Errors$^{(J)}$
**exports**
  **sorts** VENV
  **context-free syntax**
    "[" Var-declarations "]" → VENV
    *double-check-var-declarations*(VENV, Var-declarations) → ERROR-LIST
    *double-check-var*(VENV, Var-declaration) → ERROR-LIST
    "*IdealVariables*" → VENV

    "*Variable-in*" *var-declaration* Var-declaration
    yet undefined in variable-side-file → ERROR
  **variables**
    *Venv* [′]+→ VENV
**hiddens**
  **context-free syntax**
    *dummy-par-var-type* → Par-var-type
**equations**

[1] *double-check-var-declarations*(*Venv'*, ) =

[2] *double-check-var-declarations*(*Venv'*, *Var-declaration' Var-declarations'*) =
*join-errors*(*double-check-var*(*Venv'*, *Var-declaration'*), *double-check-var-declarations*(*Venv'*, *Var-declarations'*))

[3] *double-check-var*([*Var-declarations'* *Identifier'* : *Par-var-type'*; *Var-declarations''*], *Identifier'* : *Par-var-type''*;) =
*double-check-var*([*Var-declarations'* *Identifier'* : *Par-var-type'*; *Var-declarations''*],

[4] *Identifier' Dir-ini-list'* : *Par-var-type''*;) =

[5] *double-check-var*( *Venv*′, *Var-declaration*′) =

*Variable-in var-declaration* Var-declaration′ *yet undefined in variable-side-file* **otherwise**

This side file is filled with dummies since at this stage it is not perfectly clear which type convention should be chosen for.

[6] *IdealVariables* =

[A03 : *dummy-par-var-type*; A06 : *dummy-par-var-type*;
AMA : *dummy-par-var-type*; AM0 : *dummy-par-var-type*;
AM1 : *dummy-par-var-type*; APM : *dummy-par-var-type*;
ARI : *dummy-par-var-type*; B01 : *dummy-par-var-type*;
B04 : *dummy-par-var-type*; B05 : *dummy-par-var-type*;
B07 : *dummy-par-var-type*; CDT : *dummy-par-var-type*;
CPC : *dummy-par-var-type*; DPP : *dummy-par-var-type*;
ERL : *dummy-par-var-type*; EWT : *dummy-par-var-type*;
FIR : *dummy-par-var-type*; FIT : *dummy-par-var-type*;
FPB : *dummy-par-var-type*; FTO : *dummy-par-var-type*;
LAM : *dummy-par-var-type*; MRN : *dummy-par-var-type*;
MRX : *dummy-par-var-type*; MSW : *dummy-par-var-type*;
NPU : *dummy-par-var-type*; NRI : *dummy-par-var-type*;
NSBA : *dummy-par-var-type*; NSBC : *dummy-par-var-type*;
NSL : *dummy-par-var-type*; NSM : *dummy-par-var-type*;
OSL : *dummy-par-var-type*; OSM : *dummy-par-var-type*;
OPP : *dummy-par-var-type*; PCB : *dummy-par-var-type*;
PCI : *dummy-par-var-type*; PER : *dummy-par-var-type*;
PIB : *dummy-par-var-type*; PIC : *dummy-par-var-type*;
PLBA : *dummy-par-var-type*; PLBC : *dummy-par-var-type*;
PPCB : *dummy-par-var-type*; PPI : *dummy-par-var-type*;
PPP : *dummy-par-var-type*; PPU : *dummy-par-var-type*;
PRBA : *dummy-par-var-type*; PRBC : *dummy-par-var-type*;
PRC : *dummy-par-var-type*; PRL : *dummy-par-var-type*;
PRM : *dummy-par-var-type*; PSB : *dummy-par-var-type*;
PSO : *dummy-par-var-type*; PSS : *dummy-par-var-type*;
PTSO : *dummy-par-var-type*; PTI : *dummy-par-var-type*;
RDI : *dummy-par-var-type*; RLD : *dummy-par-var-type*;
RRX : *dummy-par-var-type*; SPC : *dummy-par-var-type*;
SPCB : *dummy-par-var-type*; SAS : *dummy-par-var-type*;
SCI : *dummy-par-var-type*; SDC : *dummy-par-var-type*;
SIS : *dummy-par-var-type*; SRI : *dummy-par-var-type*;
SRP : *dummy-par-var-type*; STB : *dummy-par-var-type*;
STI : *dummy-par-var-type*; TAI : *dummy-par-var-type*;
TDI : *dummy-par-var-type*; TPC : *dummy-par-var-type*;
TPO : *dummy-par-var-type*; TPS : *dummy-par-var-type*;
TRL : *dummy-par-var-type*; TSO : *dummy-par-var-type*;
TSU : *dummy-par-var-type*; TSU0 : *dummy-par-var-type*;
TSU1 : *dummy-par-var-type*; TSU2 : *dummy-par-var-type*;
TSU3 : *dummy-par-var-type*; TSU4 : *dummy-par-var-type*;
TSU5 : *dummy-par-var-type*; TSU6 : *dummy-par-var-type*;
TSU7 : *dummy-par-var-type*; TSU8 : *dummy-par-var-type*;
TSU9 : *dummy-par-var-type*; W01 : *dummy-par-var-type*;
WDC : *dummy-par-var-type*; WDT : *dummy-par-var-type*;
XPU : *dummy-par-var-type*; XSBA : *dummy-par-var-type*;
XSBC : *dummy-par-var-type*; XSL : *dummy-par-var-type*;
XSM : *dummy-par-var-type*; XTU : *dummy-par-var-type*;]