A Small Language for the Specification of Grid Protocols

Joris Hillebrand

Programming Research Group, University of Amsterdam

Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

Email: joris@fwi.uva.nl

Abstract

We present a tool that transforms specifications in a new format, geared towards Grid Protocols, into μCRL specifications. The specification of Grid Protocols becomes easier in the new format, while the transformation to μCRL keeps existing μCRL tools accessible for these specifications.

Key Words & Phrases: Grid Protocol, Tool support, Language definition, μCRL, Formal specification, Algebraic specification, ASF+SDF, Prettyprinting.

Contents

1 Introduction **2**

2 The Grid Protocol Specification Language **3**
- 2.1 Expanding μCRL with Kleene star, free merge, and free left merge . . . 4
- 2.2 Defining GPSL . . . 5

3 Translating GPSL to mCRL **7**
- 3.1 Transforming the specification style . . . 7
- 3.2 Eliminating prefix and early reads . . . 8
- 3.3 An alternative approach for parallel input . . . 10
- 3.4 Adding some declarations . . . 10

4 Implementation **11**
- 4.1 Checking constraints . . . 12
- 4.2 The straightforward translation . . . 13
- 4.3 The alternative translation for parallel input . . . 15

5 Results and conclusions **16**

A The specification **19**
- A.1 MuCRL . . . 20
- A.2 MiCRL . . . 21
- A.3 GPSL . . . 22
- A.4 Signature-GPSL . . . 23
- A.5 GPSL-aux . . . 25
- A.6 GPSL-errors . . . 32
- A.7 Precheck . . . 34
- A.8 Substitution . . . 44
- A.9 ToBPA . . . 46
- A.10 Translate . . . 53

1 Introduction

In [HK95], an algebraic specification of a well-formedness checker for the specification language μCRL is explained. Using the ASF+SDF specification formalism, conditions for statical semantical correctness (SSC) and some other 'well-formedness' requirements, formulated in [GP95], have been implemented, resulting in a working tool. This tool can serve as a frontend for other tools that require a well-formed μCRL specification.

This paper presents tool support for the specification and prototyping of *grid protocols* in μCRL. Grid Protocols are a topic of interest in the study of parallel computing. A grid protocol is a network of *modules*, elementary processors of data that can cooperate with each other by passing values. A typical example can be found in [BHP], where a parallel algorithm for the numerical computation of solutions of the one-dimension wave equation is presented. The modeling presented there, and which is used in this paper, is mainly based on the work by Tucker and Thompson, [TT94].

When we want to specify grid protocols, we need a way to model parallel input. Due to the intrinsic difficulties that go with variable binding, this is not straightforward in μCRL. To demonstrate this problem, let P be a process that is able to read two naturals along ports 1 and 2. If values n and m are read, respectively, the process evolves into $Q(n, m)$. In case the particular order of reading is immaterial (or *should be* irrelevant), we have a typical case of "parallel inputs". The process P can then be specified as

$$\begin{aligned} P \;=\; & \sum\Big(v : \mathbb{N},\; r_1(v) \cdot \sum\big(w : \mathbb{N},\; r_2(w) \cdot Q(v, w)\big)\Big) \;+ \\ & \sum\Big(w : \mathbb{N},\; r_2(w) \cdot \sum\big(v : \mathbb{N},\; r_1(v) \cdot Q(v, w)\big)\Big) \end{aligned}$$

where the reading is interleaved: either first along port 1 and then along port 2 (first summand), or vice versa (second summand). So the order in which binding takes place is not relevant. The variable binding properties of the $\sum$-operator, however, are straightforward: its scope is given by the brackets in $\sum(v : \mathbb{N}, \ldots)$. Consequently, there is no way of expressing P using the parallel operator $\|$. With the process prefix operation (;), and early read actions $er_i(v)$ with i a channel or port identifier and v a data variable, this problem can be solved. Consider the axioms for process prefixing in Table 1.

Table 1: Process prefixing, $a \in A$.

(PP1)	$\delta; x$	$=$	δ	|	(PP4)	$er_k(v); x$	$=$	$\sum(v : D,\; r_k(v) \cdot x)$
(PP2)	$\tau; x$	$=$	$\tau \cdot x$	|	(PP5)	$(x + y); z$	$=$	$x; z + y; z$
(PP3)	$a; x$	$=$	$a \cdot x$	|	(PP6)	$(x \cdot y); z$	$=$	$x; (y; z)$

The process prefix mostly behaves like sequential composition $(\cdot)$. However, an early-read action in front of a process prefix evolves into a value-receiving term. An introduction to process prefixes in the context of ACP can be found in [BB94]. In [Pon], process prefixes and early-read actions are added to μCRL.

Using process prefixes the process P can be easily described:

$$P \;=\; (er_1(v) \parallel er_2(w)); Q(v, w).$$

Process prefixes are no *functional* extension to μCRL. However, they simplify the specification of protocols with parallel input considerably, as the previous example showed. The problem is that existing theories or tools for μCRL specifications cannot deal with process prefixes, since they are no part of standard μCRL.

In this paper, we give an algebraic specification of a tool that transforms μCRL specifications with process prefixes to standard μCRL specifications. We take the language definition of [Pon], the theory of [BHP] geared towards μCRL, as point of departure. Again using the ASF+SDF formalism, we can integrate this tool with the well-formedness checker from [HK95]. The translated specification will be a pretty-printed μCRL specification. For this purpose, we generated a pretty-printer for μCRL using the system described in [vdBV96]. The window containing this pretty-printed μCRL specification has a button that invokes the well-formedness checker.

In Section 2, an SDF specification of the language definition from [Pon] is given, based on the one for μCRL. Then in Section 3, we describe the transformation process and the necessary constraints. Section 4 deals with the implementation in ASF+SDF, and in Section 5, we present some results and concluding remarks. Finally, Appendix A contains the complete ASF+SDF specification.

Acknowledgements. The author acknowledges Alban Ponse for the many fruitful discussions involving the design of the translation tool. Jan Bergstra came up with the alternative approach for the elimination of parallel input, described in Section 3.3. Mark van den Brand has been of great help in working with the pretty-print tools.

2 The Grid Protocol Specification Language

In this section, we define a language for the specification of Grid Protocols: the Grid Protocol Specification Language, or GPSL. We work conform [Pon], which means that the language will be an adaptation of μCRL [GP95]. The most important feature of the new language will of course be operators/constructs by which we are able to specify parallel input: the prefix operator (;) and early reads. At this point we stress once more that we are *not* going to expand μCRL with these constructs. One reason for this is that the early-read actions only have a meaning in a very specific context, namely in a process prefix (in front of ';'). Another reason is that early reads and the prefix operator do not add functionality to μCRL. Parallel input can be specified without early reads, only in a less attractive way. That is why we consider early reads as a kind of macros and describe a method to expand these macros.

Following [Pon], there is yet another point where our language deviates from μCRL: global variable declarations for processes. In μCRL, we have to write:

proc $P(b1{:}Bool) = Expr1$

$Q(b1{:}Bool,\ b2{:}Bool) = Expr2$

In the new language, we work as follows:

var $b1,\ b2{:}\ Bool$

proc $P(b1) = Expr1$
$Q(b1{,}b2) = Expr2$

So all variables used in a process specification are declared globally for that process specification. This makes the specification, but especially the reading of processes easier.

It is very easy to specify Grid Protocols with the use of *iteration*. We have decided to expand μCRL with the binary Kleene star (*), because iteration is being used more and more frequently in process specification (in [BHP], almost all specifications contain iteration).

In Section 3, we will find ourselves in a situation in which we need parallel composition without communication. Although this is possible by simply defining no communications, it would be easy to write this down explicitly. Therefore, we also add two variations on the merge and left merge: the *free* merge ($|||$) and the *free* left merge ($|||\!\lfloor$). These operators originate from [BK82], where the axiom system PA for interleaving processes without communication was introduced.

$$
\begin{aligned}
x \mathbin{|||} y &= x \mathbin{|||\!\lfloor} y + y \mathbin{|||\!\lfloor} x \\
a \mathbin{|||\!\lfloor} x &= ax \\
ax \mathbin{|||\!\lfloor} y &= a(x \mathbin{|||} y) \\
(x+y) \mathbin{|||\!\lfloor} z &= x \mathbin{|||\!\lfloor} z + y \mathbin{|||\!\lfloor} z
\end{aligned}
$$

(where a is an action).

In the remainder of this section we will first formally expand μCRL with $|||$, $|||\!\lfloor$, and *. We will call this extension mCRL(*mini* CRL, because it contains more operators than *micro* CRL). This will be the basis of GPSL. After that, a formal definition of GPSL is given.

2.1 Expanding μCRL with Kleene star, free merge, and free left merge

In [HK95], a formal definition of the syntax of μCRL in SDF format has been given. This definition is identical to the one given in BNF format in [GP95]. The latter contains a complete definition of the syntax and semantics of μCRL. Unfortunately, the definition of the μCRL syntax is sometimes hard to work with. The priorities of the different process operators have been defined implicitly. This is done by dividing process expressions with a different main operator into different syntactical categories. A side-effect of this approach is that the *bracket* function has not been used. To understand the effect of the bracket function, see the following definition:

"(" Process-expression ")" $\rightarrow$ Process-expression {**bracket**}

The {**bracket**} attribute in this function definition has the effect that a process expression, surrounded by any number of parentheses, is still recognized as a process expression. Moreover, in the ASF+SDF meta-environment, parentheses will be added or omitted automatically. This feature is especially useful when we are going to generate mCRL code later on. For example, consider the following process expression:

$Expr_1 \parallel Expr_2$.

Suppose that $Expr_1$ is transformed to $a + b$, and $Expr_2$ to $a \cdot b$ (for actions a and b). Transforming $Expr_1 \parallel Expr_2$ without bracket function would lead to

$a + b \parallel a \cdot b$.

Since $\parallel$ binds stronger than $+$, this means $a + (b \parallel a \cdot b)$, which is clearly not what we want. A solution would be to translate $Expr_1$ into $(a + b)$ and $Expr_2$ into $(a \cdot b)$, resulting in:

$(a + b) \parallel (a \cdot b)$.

Although correct, this expression has a redundant pair of parentheses surrounding $a \cdot b$. What we see is that working along the definition from [GP95] or [HK95], we risk syntax errors in some contexts if we do not write parentheses around every piece of process expression that we generate. But expressions like $P = ((a) \cdot ((b) \cdot (c)))$ are hardly readable. Using the bracket function, we do not need to think about these problems. In that case, parentheses are added automatically when necessary, by which the result will be

$(a + b) \parallel a \cdot b$.

In Appendix A, an alternative SDF definition of the μCRL syntax definition can be found. This definition is equivalent to the ones in [GP95] and [HK95], that is, the same strings (μCRL specifications) are accepted. In this definition, we have *one* syntactical category for process expressions, including a bracket function. Priorities have been defined explicitly. Given this definition, we are going to add syntax rules for the operators $|||$, $|||\!\!\_$, and *. This is extremely easy when we follow a modular approach. The module that defines (standard) μCRL is called MuCRL. Next, we create a module MiCRL that imports MuCRL. Now MiCRL contains everything from MuCRL. Our next step is to add the syntax rules for the three new operators:

Process-expression "$|||$" Process-expression $\rightarrow$ Process-expression {**assoc**}
Process-expression "$|||\!\!\_$" Process-expression $\rightarrow$ Process-expression {**non-assoc**}
Process-expression "$*$" Process-expression $\rightarrow$ Process-expression {**non-assoc**}

Finally, we redefine the priorities. This is necessary, because we have added new operators for process expressions.

Module MiCRL now defines (standard) μCRL plus the three new operators. The result can be found in Appendix A. For completeness sake, we also extend the well-formedness checker from [HK95] with the new operators. The procedure is identical: create a new check module that imports the old one and add the rules for the new operators.

2.2 Defining GPSL

Basically, there are two approaches to define GPSL. The first one is a complete stand-alone definition. This implies that a lot of syntax rules will be copied from the definition of μCRL. For instance, the rule

delta $\rightarrow$ Process-expression

will appear in both definitions, because in both languages delta is a process expression. However, this approach causes serious problems when we are going to translate specifications in our language into mCRL. For this translation we need a module Translate that imports both MiCRL and the module for GPSL. Then Translate will have a function like

trans(Specification) $\rightarrow$ Specification

that does the actual translation. At some point, the process expressions will be translated. Because delta is the same in both languages, we will have that

trans(delta) = delta.

This equation will not be accepted by the ASF+SDF meta-environment, because it is not clear which delta originates from which language. Because there is no way to express the origin of a piece of syntax in the equations, this ambiguity cannot be solved. We could of course give different names to the syntactical categories in both language definitions, but it hardly makes sense to give identical classes different names.

The second approach is to import MiCRL and use what we need from it. We can re-use everything except the process specification part, which will then have to be redefined. The problem here is that the original definitions are still visible. We can have a new sort Process-expression-gpsl and new rules like

delta $\rightarrow$ Process-expression-gpsl

but the former rule

delta $\rightarrow$ Process-expression

from MiCRL is still valid. Using this approach, it will not be clear which delta is meant when used in a specification. What we need is a construction by which we can hide parts of an imported module. Such a construction is, however, not present in SDF.

Because the two approaches that define GPSL exactly do not work, we choose a weaker, but workable solution. We define a language that is "too big" and forbid the use of constructions that we rather would not have had afterwards.

The first step is to define a new module GPSL that imports MiCRL. Next, we add the prefix operator and the early-read construction:

Process-expression ";" Process-expression → Process-expression {**non-assoc**}
er "(" Data-term "," {Name ","}+ ")" → Process-expression

The first argument of the early-read construction can be used to specify the channel from which the data is read. Because the user is free to specify any kind of channel, this argument can be any data term. The other argument are the input variables, which will have to be declared in the variable-declaration section preceding the process specification. In conformity with [Pon], we allow early-read actions with more than one input variable. Application of axiom PP4 will in that case result in multiple commuting $\sum$-applications.

Next, we define the new way to declare processes:

Name "(" {Name ","}+ ")" "=" Process-expression → Process-declaration

The process headers no longer contain full variable declarations, but only the variable names. These variables will have to be declared in the variable-declaration section preceding the process declarations. So, we also have to express that process specifications are now preceded by a variable-declaration section:

Variable-declaration-section Process-specification → Process-specification-gpsl
Process-specification-gpsl → Single-specification

The new kind of process specification is called Process-specification-gpsl. It consists of a variable-declaration section (which may be empty) followed by an "old style" process specification. The second rule expresses that the new sort Process-specification-gpsl is a single specification, of which specifications are built. To keep a uniform way of process specification, we also replace the variable declaration in the sum construction by just the variable name:

sum "(" Name "," Process-expression ")" → Process-expression

Again, the declaration of the variable is moved to the variable-declaration section preceding the process specification.

Finally, we redefine the priorities, because we have added new operators on process expressions.

The module GPSL can be found in Appendix A. The result is that we have added two constructions (process prefixing and early reads), and introduced a new way for the specification of processes and the use of the sum construction. The problem is that the "old" rules are still valid:

Name "(" {Single-variable-declaration ","}+ ")" "="Process-expression → Process-declaration
Process-specification → Single-specification
sum "(" Single-variable-declaration "," Process-expression ")" → Process-expression

So nothing can keep us from still using process specifications the old style. We have shown, however, that better definitions all cause ambiguity problems at some point. The best thing we can do is to write an extra tool that checks for the use of old-style constructs and subsequently gives an error message.

In the next sections, we describe the functionality and the implementation of the tools.

3 Translating GPSL to mCRL

This section describes how a specification in the GPSL format can be translated into a mCRL specification. During the translation process we will find a number of conditions that have to be met. Without these conditions, no univocal translation, or no translation at all, will be possible. In the next section we show the implementation in the ASF+SDF meta-environment.

3.1 Transforming the specification style

A mCRL specification is built of one or more *single specifications*. A single specification can be the specification of sorts, functions, rewrite rules (with variables), actions, communications, or processes.

A GPSL specification is also built of these single specifications. The only thing that is different is how processes are specified. As a result, sort, function, rewrite, action, and communication specifications are already in mCRL format and do not have to be changed.

As discussed in Section 2.2, the GPSL syntax formally allows process specifications in mCRL format as well. This, however, was a measure in absence of any better solutions and usage of it should be considered undesirable. This brings us to our first **constraint**:

- no mCRL-style process specifications.

What is left is the translation of process specifications in the GPSL format into ones in mCRL format. This means that the specification

var *b1*, *b2*: *Bool*

proc *P*(*b1*) = *Expr1*
Q(*b1*,*b2*) = *Expr2*

must be changed into

proc *P*(*b1*:*Bool*) = *Expr1*

Q(*b1*:*Bool*, *b2*:*Bool*) = *Expr2*

This is not difficult: just look in the variable-declaration section preceding the process specification for the sorts of $b1$ and $b2$. There are two **constraints**:

- the variables in process headers must be declared;
- the variables in a variable-declaration section can only be declared once.

Next, we have to translate the bodies of the processes: the process expressions. The only process expressions that have to be dealt with are those that use the prefix operator or early reads. These are the operators used to give a short notation for parallel input, but they are not present in mCRL and should therefore be eliminated. Apart from that, a sum expression must be changed from

$\sum(b, \mathit{Expr})$,

with b: *Bool* declared in the variable-declaration section, into

$\sum(b{:}\mathit{Bool}, \mathit{Expr})$.

This transformation is practically identical to the one used above for the process headers. Again, we have that our syntax formally allows use of the sum in mCRL format as well. This leads to the following **constraints**:

- no mCRL-style use of the sum;
- the variable in the sum must be declared (uniquely).

All other process expressions are already in mCRL format and can be left unchanged.

3.2 Eliminating prefix and early reads

We have now arrived at the point where it is all about: elimination of early reads and the prefix operator. It is not surprising that this is the major part of the translation.

Because early reads only have a special meaning when used in the left-hand side of the prefix operator, we do not consider other appearances of these actions. **Constraint**:

- early reads should appear in the left-hand side of the prefix operator only.

Table 1 in Section 1 shows how the prefix operator can be eliminated. An early-read action in front of it disappears together with the prefix operator. However, in Table 1, the prefix operator is only defined on so-called pCRL expressions, that is, μCRL expressions without parallelism: processes with plus, times, sum, if-then-else, prefix, and early reads. There is, for instance, no axiom for $(x \parallel y); z$. For this reason, we have to rewrite the process at the left-hand side of the prefix operator to a pCRL expression first. This is where the major constraints come in. When we have to rewrite a process expression, we want the rewrite procedure to terminate. Therefore, we prohibit recursion and the use of iteration in the left-hand side of the prefix operator.

Although a process call does not necessarily lead to recursion, it can lead to very nasty substitution problems. For instance, consider the following process:

$$P(v) = (er_k(v) \parallel a(v)); b(v)$$

with a and b data-parametric actions and v a variable over the naturals. Suppose that the process P is called with value 0. It is definitely wrong to change all occurrences of v in the body of P into 0. First we have to rewrite P, after which it looks as follows:

$$\begin{aligned} P(v) \;=\; (er_k(v) \parallel a(v)); b(v) &= (er_k(v) \cdot a(v) + a(v) \cdot er_k(v)); b(v) \\ &= er_k(v) \cdot a(v); b(v) + (a(v) \cdot er_k(v)); b(v) \\ &= er_k(v); (a(v) \cdot b(v)) + a(v); (er_k(v); b(v)) \\ &= \textstyle\sum(v, r_k(v) \cdot a(v) \cdot b(v)) + a(\underline{v}) \cdot \sum(v, r_k(v) \cdot b(v)). \end{aligned}$$

Now we can see where we can replace v by 0 (the underlined instance). One can imagine that this really becomes a problem when P in its turn is called from within the left-hand side of a prefix operator. It is for this that we prohibit all process calls in the left-hand side of the prefix operator. Because this place contains in most Grid Protocols only a number of early reads in parallel, this is not a very heavy restriction in practice.

Another problem is communication. Suppose we have the following process expression with actions a and b:

$$(a(x) \parallel b(y));\ \mathit{Expr}.$$

Rewriting the part in front of the prefix operator leads to:

$$(a(x)\cdot b(y) \;+\; b(y)\cdot a(x) \;+\; a(x) \mid b(y));\ \mathit{Expr}.$$

The problem is in the last summand. If a communication has been defined on a and b, this communication can only take place if x and y are equal according to the rewrite rules in the specification. This means that we would have to interpret these rewrite rules and that is typically something that we do not want to do. In fact, this is not even decidable in general. We want a tool that takes a specification in one format and then transforms it to another format *without interpreting it*. The rewrite rules in the specification might not even be semantically correct! Therefore, we forbid the use of communication and only allow the use of the free merge and the free left-merge as parallel operators in the left-hand side of the prefix operator. Regarding the fact again that Grid Protocols mostly have a number of early reads in parallel at that place, this restriction does not have very big consequences as well. Especially not because there is no communication between early reads.

(M1)	$x \mathbin{\|\|\|} y$	$\rightarrow$	$x \mathbin{\|\|\|\!\lfloor} y + y \mathbin{\|\|\|\!\lfloor} x$
(M2a)	$\delta \mathbin{\|\|\|\!\lfloor} x$	$\rightarrow$	δ
(M2b)	$\tau \mathbin{\|\|\|\!\lfloor} x$	$\rightarrow$	$\tau \cdot x$
(M2c)	$\alpha \mathbin{\|\|\|\!\lfloor} x$	$\rightarrow$	$\alpha \cdot x$
(M2d)	$er_k(v) \mathbin{\|\|\|\!\lfloor} x$	$\rightarrow$	$er_k(v) \cdot x$
(M3a)	$\delta \cdot x \mathbin{\|\|\|\!\lfloor} y$	$\rightarrow$	δ
(M3b)	$\tau \cdot x \mathbin{\|\|\|\!\lfloor} y$	$\rightarrow$	$\tau \cdot (x \mathbin{\|\|\|} y)$
(M3c)	$\alpha \cdot x \mathbin{\|\|\|\!\lfloor} y$	$\rightarrow$	$\alpha \cdot (x \mathbin{\|\|\|} y)$
(M3d)	$er_k(v) \cdot x \mathbin{\|\|\|\!\lfloor} y$	$\rightarrow$	$er_k(v) \cdot (x \mathbin{\|\|\|} y)$
(M3e)	$(x+y).z \mathbin{\|\|\|\!\lfloor} u$	$\rightarrow$	$x.z \mathbin{\|\|\|\!\lfloor} u + y.z \mathbin{\|\|\|\!\lfloor} u$
(M3f)	$(x \cdot y).z \mathbin{\|\|\|\!\lfloor} u$	$\rightarrow$	$x.(y.z) \mathbin{\|\|\|\!\lfloor} u$
(M3g)**	$\sum(d,x).y \mathbin{\|\|\|\!\lfloor} z$	$\rightarrow$	$\sum(d, x.y \mathbin{\|\|\|\!\lfloor} z)$
(M3h)	$(x \lhd B \rhd y).z \mathbin{\|\|\|\!\lfloor} u$	$\rightarrow$	$(x.z \mathbin{\|\|\|\!\lfloor} u) \lhd B \rhd (y.z \mathbin{\|\|\|\!\lfloor} u)$
(M4)	$(x+y) \mathbin{\|\|\|\!\lfloor} z$	$\rightarrow$	$x \mathbin{\|\|\|\!\lfloor} z + y \mathbin{\|\|\|\!\lfloor} z$
(M5)*	$\sum(d,x) \mathbin{\|\|\|\!\lfloor} y$	$\rightarrow$	$\sum(d, x \mathbin{\|\|\|\!\lfloor} y)$
(M6)	$x \lhd B \rhd y \mathbin{\|\|\|\!\lfloor} z$	$\rightarrow$	$(x \mathbin{\|\|\|\!\lfloor} z) \lhd B \rhd (y \mathbin{\|\|\|\!\lfloor} z)$
(PP1)	$\delta; x$	$\rightarrow$	δ
(PP2)	$\tau; x$	$\rightarrow$	$\tau \cdot x$
(PP3)	$\alpha; x$	$\rightarrow$	$\alpha \cdot x$
(PP4)	$er_k(v); x$	$\rightarrow$	$\sum(v, r_k(v) \cdot x)$
(PP5)	$(x+y); z$	$\rightarrow$	$x; z + y; z$
(PP6)	$(x \cdot y); z$	$\rightarrow$	$x; (y; z)$
(PP7)*	$\sum(d,x); y$	$\rightarrow$	$\sum(d, x; y)$
(PP8)	$(x \lhd B \rhd y); z$	$\rightarrow$	$x; z \lhd B \rhd y; z$

*if d is not free in y
**if d is not free in y or z

(DH1)	$\partial_H(\delta)$	$\rightarrow$	δ
(DH2)	$\partial_H(\tau)$	$\rightarrow$	τ
(DH3)	$\partial_H(er_k(v))$	$\rightarrow$	$er_k(v)$
(DH4)	$\partial_H(x+y)$	$\rightarrow$	$\partial_H(x) + \partial_H(y)$
(DH5)	$\partial_H(x \cdot y)$	$\rightarrow$	$\partial_H(x) \cdot \partial_H(y)$
(DH6)	$\partial_H(\sum(d,x))$	$\rightarrow$	$\sum(d, \partial_H(x))$
(DH7)	$\partial_H(x \lhd B \rhd y)$	$\rightarrow$	$\partial_H(x) \lhd B \rhd \partial_H(y)$
(TI1)	$\tau_I(\delta)$	$\rightarrow$	δ
(TI2)	$\tau_I(\tau)$	$\rightarrow$	τ
(TI3)	$\tau_I(er_k(v))$	$\rightarrow$	$er_k(v)$
(TI4)	$\tau_I(x+y)$	$\rightarrow$	$\tau_I(x) + \tau_I(y)$
(TI5)	$\tau_I(x \cdot y)$	$\rightarrow$	$\tau_I(x) \cdot \tau_I(y)$
(TI6)	$\tau_I(\sum(d,x))$	$\rightarrow$	$\sum(d, \tau_I(x))$
(TI7)	$\tau_I(x \lhd B \rhd y)$	$\rightarrow$	$\tau_I(x) \lhd B \rhd \tau_I(y)$
(RF1)	$\rho_f(\delta)$	$\rightarrow$	δ
(RF2)	$\rho_f(\tau)$	$\rightarrow$	τ
(RF3)	$\rho_f(er_k(v))$	$\rightarrow$	$er_k(v)$
(RF4)	$\rho_f(x+y)$	$\rightarrow$	$\rho_f(x) + \rho_f(y)$
(RF5)	$\rho_f(x \cdot y)$	$\rightarrow$	$\rho_f(x) \cdot \rho_f(y)$
(RF6)	$\rho_f(\sum(d,x))$	$\rightarrow$	$\sum(d, \rho_f(x))$
(RF7)	$\rho_f(x \lhd B \rhd y)$	$\rightarrow$	$\rho_f(x) \lhd B \rhd \rho_f(y)$
(X1)	$x \parallel y$	$\rightarrow$	δ
(X2)	$x \mathbin{\|\!\lfloor} y$	$\rightarrow$	δ
(X3)	$x \mid y$	$\rightarrow$	δ
(X4)	$x^* y$	$\rightarrow$	δ

Table 2: Term rewriting system, used for the elimination of the prefix operator.

Summerizing, we have the following **constraints** for process expressions at the left-hand side of the prefix operator:

- no process calls;
- no use of $\parallel$, $\mathbin{\|\!\lfloor}$, $\mid$, or *.

Note that $\mathbin{|||}$ and $\mathbin{|||\!\lfloor}$ are allowed (that is, $\parallel$ and $\mathbin{\|\!\lfloor}$ without communication).

The complete prefix expression can be rewritten using the term rewriting system in Table 2. This term rewriting system is basically an implementation of the axioms for the free (left) merge, encapsulation, hiding, renaming, and process prefixing. Its normal forms are pCRL expressions. In Table 2, x, y and z stand for arbitrary GPSL processes. The symbol α is a so-called *atomic element*. Atomic elements can be defined as follows:

1. atomic actions $a, b, c, \ldots$ are atomic elements;
2. $\partial_H(\alpha)$, $\tau_I(\alpha)$ and $\rho_f(\alpha)$ are atomic elements if α is.

We use this notion of atomic elements because we do not want to interpret the use of encapsulation, hiding or renaming, just as we do not want to interpret rewrite rules. After interpretation, atomic elements will always end up in the set $A_{\tau\delta}$. Because the rewrite rules from Table 2 remain valid when we substitute elements of this set for α, this does not lead to inconsistencies.

In rewrite rules M3g, M5 and PP7, some reservation has been made in case that the variable d should occur free in y or z. We have implemented the term rewriting system such that a fresh name is given to that variable when this happens.

Rewrite rules X1 to X4 have been added for completeness. These operators are not allowed in terms at the left-hand side of the prefix operator.

The rewrite strategy is as follows. We explicitly use outermost reduction. If none of the rules apply, we apply outermost reduction *once* on the left-most argument of an expression. In case of an expression $Expr_1; Expr_2$, we first rewrite $Expr_1$ to a normal form, say $Expr_3$. Then we apply one of the rules PP1 to PP8 to $Expr_3; Expr_2$. In some cases, this procedure will have to be applied again on the arguments of the resulting expression later. However, in the end we will have eliminated all prefix operators and the result will be a pCRL expression.

3.3 An alternative approach for parallel input

What we have seen so far, is a straightforward transformation following the axioms for the free (left) merge, encapsulation, hiding, renaming, and process prefixing. Although this transformation is correct and terminating, it is yet not completely satisfactory. As mentioned before, in Grid Protocols parallel input is mostly modeled as follows:

$$\big(er_1(v_1) \mid\mid\mid er_2(v_2) \mid\mid\mid \cdots \mid\mid\mid er_n(v_n)\big);\ s_0(F(v_1, v_2, \ldots, v_n)).$$

Via n channels, n values are read in arbitrary order. The problem is that when we expand the left-hand side of the prefix operator following the method explained above, we obtain an expression of length $n!$. The result will already be hard to read for $n = 4$. From $n = 7$, the result will no longer be computable by a tool on a standard workstation. That is why we choose for another approach when we have a number of early reads in parallel in front of the prefix operator. The idea is to use a memory cell for each input channel. These memory cells are filled in arbitrary order, thus preserving the parallel nature of the input, but emptied in fixed order. We visualized this idea in Figure 1. Using this idea, the expression above can be transformed into

$$\begin{array}{ll}\tau_{\{C\}} \circ \partial_{\{R,S\}}\ [& \sum_{v_1}(r_1(v_1) \cdot S_{1'}(v_1)) \parallel \sum_{v_2}(r_2(v_2) \cdot S_{2'}(v_2)) \parallel \cdots \parallel \sum_{v_n}(r_n(v_n) \cdot S_{n'}(v_n)) \\ & \parallel \\ & \sum_{v_1}(R_{1'}(v_1) \cdot \sum_{v_2}(R_{2'}(v_2) \cdot \ldots \cdot \sum_{v_n}(R_{n'}(v_n) \cdot s_0(F(v_1, v_2, \ldots, v_n))) \cdots)) \\ ] & . \end{array}$$

Using this approach, we need to declare special actions R, S and C, and special channels $1'$ to n'. Also, we need to declare the communication $R \mid S = C$. This brings along a few **constraints**:

- no actions named R, S or C should be declared;
- no channels (modeled as constant functions) named $1'$ to n' should be declared.

By demanding that the extra ports and actions are fresh, we prevent undesired interactions.

3.4 Adding some declarations

Now that we have presented a workable translation procedure for parallel input, the transformation of individual processes has finished. What is left to do is the addition of some declarations. Because all early-read actions are changed into ordinary read actions, we need to add declarations for these read actions if they are not already present. This can cause problems. First of all, it should be possible to determine the signature of early reads. The arguments of early reads should therefore be correctly-typed data terms. Furthermore, if there are functions or processes named r with the same input signature as an early read action, an action r with that input signature would cause a name conflict. Because we do not want the resulting μCRL specification to contain semantical errors *caused by the translation*, this leads to the following **constraints**:

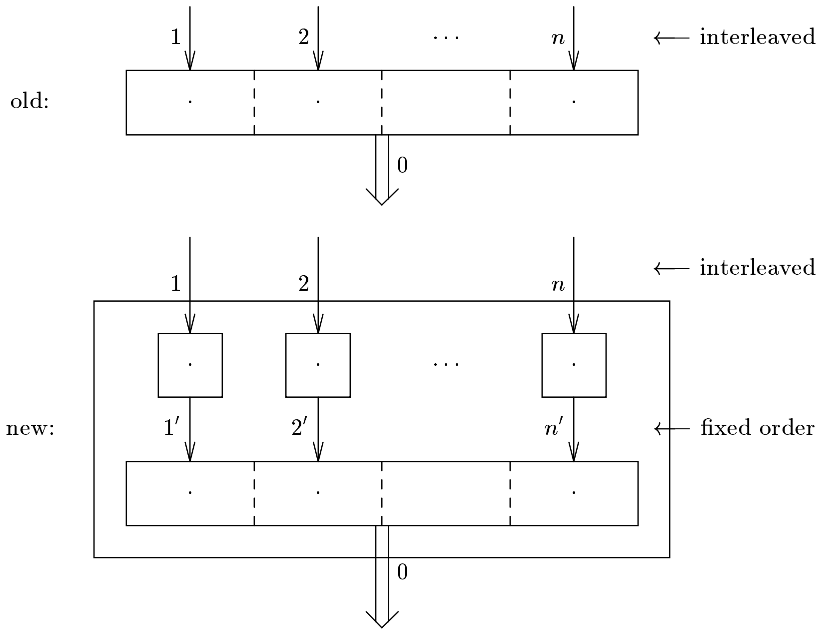

Figure 1: Parallel input with and without memory cells.

- the first argument of an early read should be a correctly-typed data term;
- the other arguments of an early read should be (uniquely) declared variables;
- there should be no function or process named r with the same input signature as an early read.

In [Pon], the standard read-send communication $r \mid s = c$ is assumed. We have to formalize it by adding this declaration if it is not already present. This can, however, cause semantical errors to arise, because communication declarations are only semantically correct under a number of conditions (see [GP95] and [HK95]). In the case of $r \mid s = c$, for example, no other communications can be defined on r and s. Furthermore, actions named r or s should be declared with the same input signatures. Also actions named c should be declared with these input signatures, although other input signatures are also allowed for c. The problem here is that the input signatures of early reads should also be taken into consideration. After all, these early reads will become read actions with the same input signature. So the **constraints** are:

- $r \mid s = c$ should be the only communication defined on actions r and s;
- for each read and early-read action, a send and a communication action with the same input signature should be declared;
- for each send action, a read and a communication action should be declared.

4 Implementation

In this section, we highlight some of the less straightforward constructions that we used to implement the translation and the function that checks for constraints.

4.1 Checking constraints

We have specified a function precheck that takes a GPSL specification and checks whether it meets all constraints as formulated in Section 3. The function returns a list of error messages when constraints are violated. We have adopted the specification of error messages from [HK95].

For the checking of some constraints, we need information about other parts of the specification. This is the case, for instance, when we demand that the first argument of an early-read action (the channel identifier) is a correctly-typed data term. For this, we need to know the signature of the functions used in the data term. In order to have this kind of information at hand, we will make use of a *signature*, as was done in [HK95]. The signature of a specification is a five-tuple, containing all sort, function, action, communication, and process declarations of that specification. By passing this signature to the check functions, we can check small pieces of the specification, while having access to relevant information about other parts of the specification.

Because the process headers do not contain all type information in GPSL, we have to extract the types of the variables in the headers first. This is done by replacing each variable name in a process header by the declaration of that variable.

$$\frac{\mathit{Svds} \;=\; \mathsf{make\text{-}svds}(\mathit{Var},\, \mathit{Cns})}{\mathsf{make\text{-}headers}(\mathit{Var},\, \mathit{N}(\mathit{Cns}) = \mathit{Pre}) \;=\; [\mathit{N}(\mathit{Svds}) = \mathit{Pre}]} \qquad \text{[headers-3]}$$

$$\mathsf{make\text{-}svds}([\mathit{OVds}_1\ \mathit{N} : \mathit{N}'\ \mathit{OVds}_2],\, \mathit{N}) = \mathit{N} : \mathit{N}' \qquad \text{[make-svds-1]}$$

Here, *Var* is the collection of variable declarations preceding the process specification.

The correctness of a data term in a process body is defined with respect to the signature of the specification and a set of variables. This set of local variables is formed by the variables named in the process header, and is a subset of the set of global variables, formed by the variable declarations preceding the process specification section. When we want to check the body of a process, we should therefore know the local variables for that process, and their types. These local variables have to be extracted from the set of global variables.

$$\mathsf{precheck\text{-}spec}(\mathit{Sig},\, \mathsf{var}\ \mathit{Vds}\ \mathsf{proc}\ \mathit{Pds}) = \mathsf{precheck\text{-}pd}(\mathit{Sig},\, [\mathit{Vds}],\, \mathit{Pds}) \qquad \text{[precheck-spec-7]}$$

$$\begin{aligned}&\mathsf{precheck\text{-}pd}(\mathit{Sig},\, \mathit{Globvar},\, \mathit{N}(\mathit{Cns}) = \mathit{Pre}) \qquad \text{[precheck-pd-3]}\\ &= \mathsf{precheck\text{-}expr}(\mathit{Sig},\, \mathit{Globvar},\, \mathsf{get\text{-}locals}(\mathit{Globvar},\, \mathit{Cns}),\, \mathit{Pre},\, \mathsf{outscope})\end{aligned}$$

The first equation shows how the set of global variables *Vds* is passed as an argument to the function precheck-pd, which checks individual process declarations. Notice that the signature of the specification *Sig* is already present.

The second equation shows that the function precheck-expr is called to check the body *Pre* of the process declaration ($\mathit{N}(\mathit{Cns}) = \mathit{Pre}$). The variable *Globvar* is the set of global variables. What we see is that the function get-locals is called to extract the local variables from the set of global variables.

$$\mathsf{get\text{-}locals}([\mathit{OVds}_1\ \mathit{N} : \mathit{N}'\ \mathit{OVds}_2],\, \mathit{N}) = [\mathit{N} : \mathit{N}'] \qquad \text{[get-locals-1]}$$

Yet *Globvar* is also sent to precheck-expr. This is necessary, because the sum operator, for instance, uses a global variable. This variable then becomes local to the process expression within the sum and has to be transferred to the set of local variables.

$$\mathsf{precheck\text{-}expr}(\mathit{Sig},\, \mathit{Globvar},\, \mathit{Locvar},\, \mathsf{sum}(\mathit{N},\, \mathit{Pre}),\, \mathit{Scope}) \qquad \text{[precheck-expr-20]}$$

$$\begin{aligned}= \mathsf{join}(&\mathsf{precheck\text{-}sum}(\mathit{Globvar}, \mathit{N}),\\ &\mathsf{precheck\text{-}expr}(\mathit{Sig}, \mathit{Globvar},\\ &\qquad\mathsf{add\text{-}to\text{-}locals}(\mathit{Globvar}, \mathit{Locvar}, \mathit{N}),\\ &\qquad\mathit{Pre}, \mathit{Scope}))\end{aligned}$$

First, precheck-sum checks whether the sum variable *N* has been declared in *Globvar*. Then the process expression *Pre* in the sum is checked, where *N* is added to the local variables (using add-to-locals).

The *Scope* variable in the equations is a flag that shows whether we are working within the scope of a prefix operator. Different constraints apply inside or outside this scope. For example, we are free to use the sum operator within or outside the scope of the prefix operator. But an early-read action can only occur within the scope of a prefix operator, whereas iteration is forbidden there.

$$\begin{aligned}&\mathsf{precheck\text{-}expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 * \mathit{Pre}_2, \mathsf{outscope})\\ &= \mathsf{join}(\mathsf{precheck\text{-}expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1, \mathsf{outscope}),\\ &\qquad\mathsf{precheck\text{-}expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_2, \mathsf{outscope}))\end{aligned} \qquad \text{[precheck-expr-13]}$$

$$\begin{aligned}&\mathsf{precheck\text{-}expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 * \mathit{Pre}_2, \mathsf{inscope})\\ &= \text{Iteration used in left-hand-side of ‘;’}\end{aligned} \qquad \text{[precheck-expr-14]}$$

4.2 The straightforward translation

The equations for the translation of process expressions are mostly very straightforward.

$$\begin{aligned}&\mathsf{trans\text{-}expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 + \mathit{Pre}_2)\\ &= \mathsf{trans\text{-}expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1) + \mathsf{trans\text{-}expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_2)\end{aligned} \qquad \text{[trans-expr-1]}$$

Things become complicated when we encounter a prefix operator.

$$\frac{\begin{array}{rcl}\mathsf{is\text{-}er\text{-}merge}(\mathit{Pre}_1) & \neq & \mathsf{true},\\ \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1, []) & = & <\mathit{Pre}, \mathit{Newvar}>,\\ \mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}; \mathit{Pre}_2, \mathit{Newvar}) & = & <\mathit{Pre}', \mathit{Newvar}'>\end{array}}{\begin{array}{l}\mathsf{trans\text{-}expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1; \mathit{Pre}_2) =\\ \mathsf{trans\text{-}expr}(\mathit{Globvar} + \mathit{Newvar}, \mathit{Locvar}, \mathit{Pre}')\end{array}} \qquad \text{[trans-expr-8]}$$

The first condition makes sure that we are not dealing with the parallel composition of early reads in front of the prefix operator. In that case, the method with the memory cells is used (see Section 3.3). The other two conditions implement the rewrite strategy explained in Section 3.2. We start off with the expression $\mathit{Pre}_1;\mathit{Pre}_2$. First Pre_1 is normalized, resulting in *Pre*. We come back at the *Newvar* variable later. Next, the function bpa performs one rewrite step on $\mathit{Pre};\mathit{Pre}_2$. The prefix operator is no longer the main operator and the result *Pre'* is given back to trans-expr.

The function bpa performs one rewrite step from the term rewriting system in Table 2. The function normalize is nothing more than a sequence of bpa applications, as long as the result is not yet a pCRL expression. For example, see the equation that implements the rule M3c.

$$\frac{\mathsf{atomic}(\mathit{Pre}_1) = \mathsf{true}}{\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \mathbin{|\!|\!\lfloor} \mathit{Pre}_2, \mathit{Newvar}) = <\mathit{Pre}_1 \,.\, \mathit{Pre}_2, \mathit{Newvar}>} \qquad \text{[bpa-39]}$$

If Pre_1 is an atomic element, then $\mathit{Pre}_1 \mathbin{|\!|\!\lfloor} \mathit{Pre}_2$ is rewritten into $\mathit{Pre}_1 \cdot \mathit{Pre}_2$.

The variable *Newvar* in the equations above has to do with the reservation that has been made towards rules M3g, M5 and PP7 in Table 2. The sum variable should not occur free in the process that will be brought into the scope of the sum. If this is the case, a new name for the sum variable is constructed. Then all occurrences of the sum variable in the process that was originally in the scope of the sum, are substituted by the new variable. Because this variable must be recognized in the remaining translation procedure, it is returned together with the rewritten process expression. The variable *Newvar* contains the newly created variables. For example, the equation for rule M5 reads:

$$\frac{\begin{array}{rcl}\textsf{is-variable}(Locvar, N) &=& \textsf{true},\\ \textsf{make-svds}(Globvar + Newvar, N) &=& N : N',\\ \textsf{new-varname}(Globvar + Locvar + Newvar, N) &=& N'',\\ \textsf{normalize}(Globvar, Locvar, Pre_1, Newvar) &=& < Pre', Newvar' >\end{array}}{\begin{array}{l}\textsf{bpa}(Globvar, Locvar, \textsf{sum}(N, Pre_1) \;|\!|\!\lfloor\; Pre_2, Newvar) =\\ < \textsf{sum}(N'', \textsf{subst}(N'', N, Pre') \;|\!|\!\lfloor\; Pre_2), Newvar' + [N'' : N'] >\end{array}} \quad \text{[bpa-52]}$$

The first condition checks whether the sum variable N occurs free in Pre_2. This is the case if N is in the set of local variables. The second condition looks for the sort N' of N. The new variable name is created by the function new-varname and is assigned to N''. Before all occurrences of N in Pre_1 are substituted by N'', Pre_1 is first normalized in the last condition. This prevents occurrences of N that are bound by another operator to be replaced. Finally, the sum expression with new variable N'' is returned and N'':N' is added to *Newvar*.

In equation [trans-expr-8] above, it is shown how the sets *Newvar* and *Newvar'*, returned by normalize and bpa, are added to *Globvar*.

The new variable name is constructed by the function new-varname.

$$\frac{\textsf{is-variable}(Var, \textsf{name}(Chs\ \texttt{"'"})) = \textsf{true}}{\textsf{new-varname}(Var, \textsf{name}(Chs)) = \textsf{new-varname}(Var, \textsf{name}(Chs\ \texttt{"'"}))} \quad \text{[new-varname-1]}$$

Primes are appended to the original name, until the resulting name no longer appears in the set of already existing variables *Var*.

The strategy to perform outermost reduction once on the first argument when no rewrite rules apply is modeled using default equations.

$$\frac{\textsf{bpa}(Globvar, Locvar, Pre_1, Newvar) = < Pre', Newvar' >}{\begin{array}{l}\textsf{bpa}(Globvar, Locvar, Pre_1 \;|\!|\!\lfloor\; Pre_2, Newvar) =\\ < Pre' \;|\!|\!\lfloor\; Pre_2, Newvar' >\end{array}} \quad \textbf{otherwise} \quad \text{[bpa-54]}$$

The **otherwise** keyword marks this equation as a default equation. It will only be executed if none of the other equations can be executed. Because the free left-merge is the main operator, this will be the case if none of the rules M2a–M6 from Table 2 apply. In the condition of equation [bpa-54], bpa performs one rewrite step on Pre_1, resulting in *Pre'*. Next, $Pre' \;|\!|\!\lfloor\; Pre_2$ is returned for further rewriting.

Finally, we present the equation modeling the most important rewrite rule from Table 2: rule PP4 that eliminates the early read and the prefix operator:

$$\begin{array}{l}\textsf{bpa}(Globvar, Locvar, \textsf{er}(Trm, Cns); Pre, Newvar)\\ = < \textsf{put-sums}(Cns, Trm, Cns, Pre, \texttt{r}), Newvar >\end{array} \quad \text{[bpa-58]}$$

The actual transformation is done in the function put-sums.

$$\frac{Trms = \textsf{transform}(Cns)}{\textsf{put-sums}(N, Trm, Cns, Pre, N') = \textsf{sum}(N, N'(Trm, Trms) \,.\, Pre)} \quad \text{[put-sums-1]}$$

$$\text{put-sums}(N,\ Cns',\ Trm,\ Cns,\ Pre,\ N') = \text{sum}(N,\ \text{put-sums}(Cns',\ Trm,\ Cns,\ Pre,\ N')) \quad \text{[put-sums-2]}$$

The second equation for put-sums shows how a nested sum is constructed from the arguments of the early-read action (channel identifier *Trm* and list of input variables *N*, Cns' with first element *N*). Remember that we can have early-read actions with more than one input variable. The first equation shows the case in which only one input variable (*N*) is left, and is a direct translation of axiom PP4. The variable *Cns* contains the original list of input variables. See the function call in equation bpa-58] for the meaning of the other variables. The name `r` in which the early-read action is changed is passed as an argument (N' in [put-sums-1]) because put-sums will be used later on with another action name.

4.3 The alternative translation for parallel input

For a number of early reads in parallel in front of a prefix operator, we decided to follow an alternative approach

$$\frac{\begin{array}{rcl} \text{is-er-merge}(Pre_1) & = & \text{true}, \\ Pre' & = & \text{hide}(\{\texttt{er-aux-c}\} \\ & & \quad ,\ \text{encap}(\{\texttt{er-aux-r},\ \texttt{er-aux-s}\} \\ & & \qquad ,\ \text{trans-er-merge}(Pre_1,\ \texttt{er-aux-port}) \\ & & \qquad \parallel \text{append-reads}(Pre_1,\ Pre_2,\ \texttt{er-aux-port}))) \end{array}}{\text{trans-expr}(Globvar,\ Locvar,\ Pre_1;\ Pre_2) \ = \ \text{trans-expr}(Globvar,\ Locvar,\ Pre')} \quad \text{[trans-expr-9]}$$

This equation models the transformation described in Section 3.3. Auxiliary actions R, S and C are called `er-aux-r`, `er-aux-s` and `er-aux-c` here. The function trans-er-merge transforms each $er_k(v)$ into $\sum_v (r_k(v) \cdot S_{k'}(v))$, or, in the formal notation: `sum(v, r(k,v)· er-aux-s(er-aux-port,v))`.

$$\frac{Trms \ = \ \text{transform}(Cns)}{\begin{array}{l}\text{trans-er-merge}(\text{er}(Trm,\ Cns),\ N) \ = \\ \text{put-sums}(Cns,\ Trm,\ Cns,\ \texttt{er-aux-s}(N,\ Trms),\ \texttt{r})\end{array}} \quad \text{[trans-er-merge-1]}$$

In this equation, the function put-sums is called again with `er-aux-s`(*N*, *Trms*) the process that must follow the read action `r`. The name of the new auxiliary port `er-aux-port` is passed as a parameter (*N*) because in case of more than one early read, fresh port names are created by appending primes to the original name.

$$\frac{\begin{array}{rcl} N & = & \text{name}(Chs), \\ N' & = & \text{name}(Chs\ \texttt{"'"}) \end{array}}{\begin{array}{l}\text{trans-er-merge}(Pre \ |\!|\!| \ \text{er}(Trm,\ Cns),\ N) \ = \\ \text{trans-er-merge}(\text{er}(Trm,\ Cns),\ N) \parallel \text{trans-er-merge}(Pre,\ N')\end{array}} \quad \text{[trans-er-merge-2]}$$

The fixed-order reading of the memory cells is modeled by the function append-reads.

$$\frac{\begin{array}{rcl} N & = & \text{name}(Chs), \\ N' & = & \text{name}(Chs\ \texttt{"'"}), \\ \text{append-reads}(Pre',\ Pre,\ N') & = & Pre'', \\ Trm' & = & N \end{array}}{\begin{array}{l}\text{append-reads}(Pre' \ |\!|\!| \ \text{er}(Trm,\ Cns),\ Pre,\ N) \ = \\ \text{put-sums}(Cns,\ Trm',\ Cns,\ Pre'',\ \texttt{er-aux-r})\end{array}} \quad \text{[append-reads-2]}$$

The sums are nested (see Section 3.3), which is implemented by a recursive call in the conditions. Again, new names for the auxiliary ports are created by appending primes. Finally, put-sums is used, with the auxiliary read action `er-aux-r` (R in the notation of Section 3.3) in stead of an ordinary read action. In the final (most nested) sum, the process expression that was at the right-hand side of the prefix operator (*Pre*) will be put in place.

The auxiliary ports and read/send actions have to be declared in the resulting specification. This is done by the function add-er-aux.

$$\frac{\begin{array}{rcl} SE_1 &=& \text{sort } \texttt{Er-aux-port}, \\ SE_2 &=& \text{er-aux-ports}(Nat), \\ SE_3 &=& \text{er-aux-acts}(Types), \\ SE_4 &=& \text{comm } \texttt{er-aux-r} \mid \texttt{er-aux-s} = \texttt{er-aux-c}, \\ SEs &=& E \end{array}}{\text{add-er-aux}(Nat,\ Types,\ E) \ = \ SEs\ SE_1\ SE_2\ SE_3\ SE_4} \quad \text{[add-er-aux-1]}$$

In this equation, E is the resulting specification so far. SE_1 is a single specification in which the sort of the auxiliary ports is declared. These ports are declared by the function er-aux-ports in SE_2. The number of auxiliary ports needed is an argument of this function. This is the largest number of early reads in parallel occurring in the original specification. In SE_3, the function er-aux-acts declares the auxiliary actions. The argument *Types* contains the input signatures of all early-read actions occurring in the original specification. For each of these input signatures, an auxiliary read, send and communication action have to be declared. In SE_4, the communication between the auxiliary read and send actions is modeled.

To conclude, we present equations for the functions er-aux-ports and er-aux-acts.

$$\frac{\begin{array}{rcl} Nat_1 &\neq& Nat_2, \\ N &=& \text{name}(Chs), \\ N' &=& \text{name}(Chs\ \texttt{"'"}), \\ Nat' &=& Nat_2 + 1 \end{array}}{\begin{array}{l} \text{er-aux-ports}(Nat_1,\ Fun,\ N,\ Nat_2) \ = \\ \text{er-aux-ports}(Nat_1,\ Fun + [N : \ \to \texttt{Er-aux-port}],\ N',\ Nat') \end{array}} \quad \text{[er-aux-ports-3]}$$

As long as Nat_2 is not equal to Nat_1, a constant function of sort `Er-aux-port` is added to *Fun*. A prime is appended to the original name `er-aux-port` each time.

$$\frac{\begin{array}{rcl} Ip &=& N\#\ Xns, \\ \text{is-action}(Act,\ \texttt{er-aux-c} : \texttt{Er-aux-port}\#\ Xns) &\neq& \text{true} \end{array}}{\begin{array}{l} \text{er-aux-acts}([Ip\ OIps],\ Act) \ = \\ \text{er-aux-acts}([OIps],\ Act + [\texttt{er-aux-r},\ \texttt{er-aux-s},\ \texttt{er-aux-c} : \texttt{Er-aux-port}\#\ Xns]) \end{array}} \quad \text{[er-aux-acts-6]}$$

For each input signature Ip (belonging to an early-read action), auxiliary actions `er-aux-r`, `er-aux-s` and `er-aux-c` are declared. The input signature is a list of names, separated by '#' (N # Xns). The first name (N) is the sort of the original input channel of the early-read action. This name is replaced by `Er-aux-port`, the sort of the auxiliary ports.

5 Results and conclusions

We have succeeded in defining a language for the specification of grid protocols together with a tool that transforms the specifications to standard μCRL specifications. The result is a platform for the

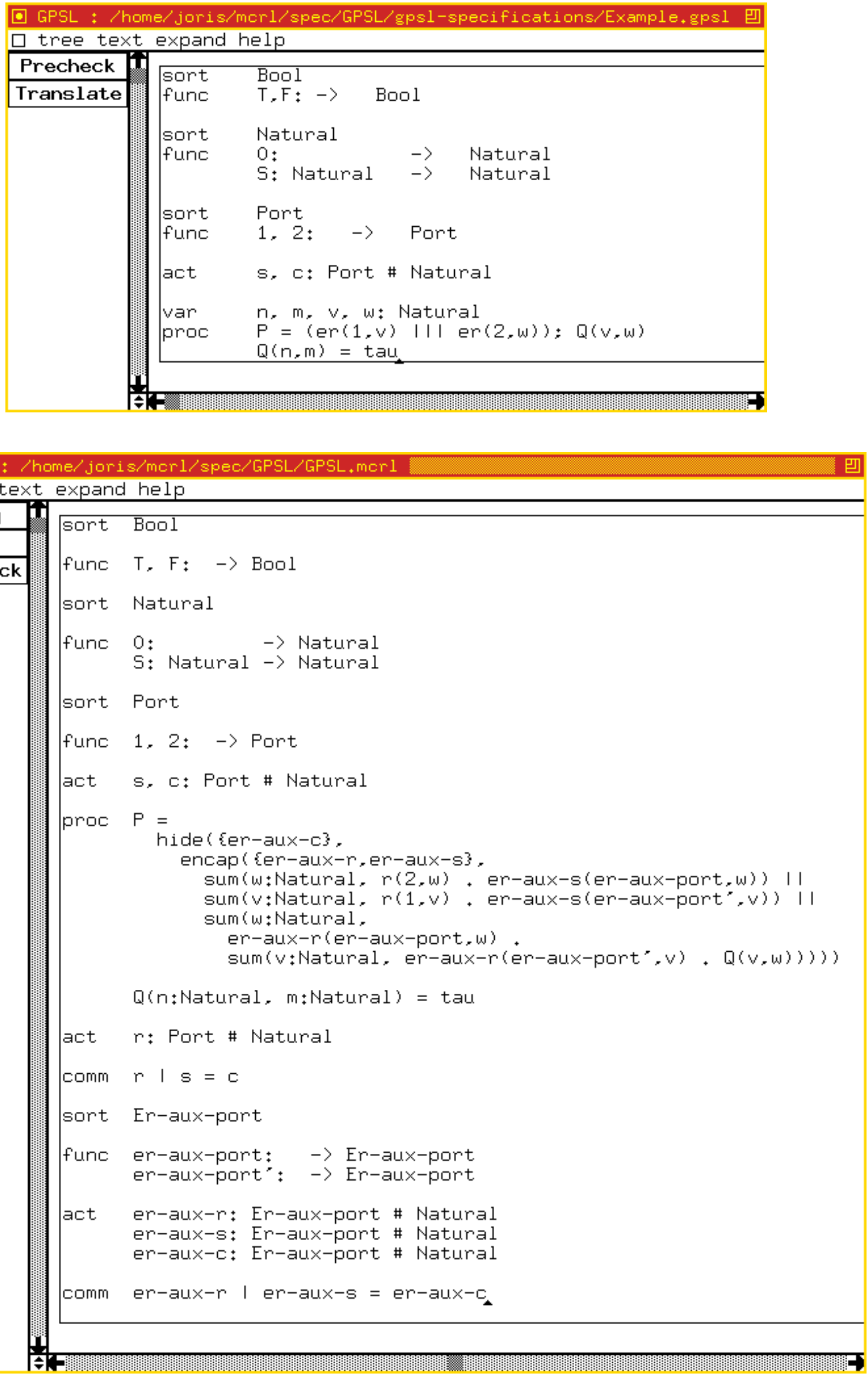

Figure 2: Specification of parallel input, before and after translation.

specification of architectures with parallel input, without losing access to tools or theories that require standard μCRL. See Figure 2 for an example of the working of the tool. The upper window contains a specification of parallel input in the early-read format. In fact, it is the specification of the process P from Section 1 (we define the process Q to consist of the invisible action τ only). The lower window shows the result of the transformation. Notice the neat way in which the resulting μCRL specification has been pretty-printed. This is the work of the tools from [vdBV96]. Also notice the **Typecheck** button using which we can invoke the well-formedness checker from [HK95].

The definition of GPSL in Section 2.2 may not be completely satisfactory. It does not *exactly* define what we had in mind, since it also allows constructions of standard μCRL that are not present in GPSL. But we have shown that exact definitions are not workable for the tool we had in mind. This problem can be solved by expanding SDF such that it becomes possible to hide parts of an imported module. Another solution would be to create a possibility to denote the origin of functions used in the equations of ASF+SDF specifications.

The specification of the transformation function and the function that checks for the preconditions, has made clear that the way processes are specified in GPSL is easy for the specifier, but troublesome for the verifier. The use of global variables allows for a shorter specification of processes with parallel input, but when checking the preconditions for the translation, we have to distinct the local variables of a process from the remaining global variables. Because the sum and the early-read construction still make use of global variables, making them local to their sub-expressions, we continuously have to carry-around two sets of variables: a global one and a local one. In case of a prefix operator, we have to collect the input variables of all early-read actions in the left-hand side of that prefix operator, because these variables are local to the process expression in the right-hand side of the prefix operator. This is a rather time-consuming job, because an entire process expression has to be searched for early reads. In standard μCRL, all variables are declared local to a certain process expression, which makes a single set of variables sufficient.

Future work. The way in which we specify Grid Protocols is rather fixed (see [BHP]). In fact, given a number of modules with a certain functionality each, and connections between these modules, the corresponding process specifications could be automatically generated. As a follow-up of the current project one could imagine a tool that takes a specification consisting of a data specification, a declaration of modules, each parameterized with functions from the data specification, and a declaration of connections between these modules. The output of such a tool can then be a GPSL specification, making the specification of Grid Protocols again easier.

References

[BB94] J.C.M. Baeten and J.A. Bergstra. On sequential composition, action prefixes and process prefix. *Journal of Formal Aspects of Computing Science*, 6(3):83–98, 1994.

[BHP] J.A. Bergstra, J.A. Hillebrand, and A. Ponse. Grid protocols based on synchronous communication. *Science of Computer Programming.* To appear.

[BK82] J.A. Bergstra and J.W. Klop. Fixed point semantics in process algebras. Report IW 206, Mathematisch Centrum, Amsterdam, 1982.

[GP95] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes*, Workshops in Computing, pages 26–62. Springer-Verlag, 1995.

[HK95] J.A. Hillebrand and H.P. Korver. A well-formedness checker for μCRL. Report P9501, Programming Research Group, University of Amsterdam, 1995.

[Pon] A. Ponse. Process prefixing and early reads in μCRL (working title). Technical report, Programming Research Group, University of Amsterdam. In preparation.

[TT94] B.C. Thompson and J.V. Tucker. Equational specification of synchronous concurrent algorithms and architectures (second edition). report CSR 15-94, University of Wales, Swansea, 1994.

[vdBV96] M.G.J. van den Brand and E. Visser. Generation of formatters for context-free languages. *ACM Transactions on Software Engineering and Methodology*, 5(1):1–41, 1996.

A The specification

The remaining part of this paper contains the full ASF+SDF specification of the tool. First the definitions of μCRL, mCRL and GPSL. In the next modules, up to Precheck, the function that checks for the necessary preconditions is specified. These modules import the modules Signature and Errors from [HK95]. The remaining modules take care of the transformation function.

A.1 MuCRL

```
imports Layout
exports
  sorts Name Sort-specification Function-specification Function-declaration
        Rewrite-specification Variable-declaration-section
        Variable-declaration Data-term Rewrite-rules-section Rewrite-rule
        Condition-operator Process-expression Renaming-declaration
        Single-variable-declaration Process-specification Process-declaration
        Action-specification Action-declaration Communication-specification
        Communication-declaration Single-specification Specification
  lexical syntax
    [a-zA-Z0-9]                                  → Name
    [a-zA-Z0-9][a-zA-Z0-9'^_\−]*[a-zA-Z0-9']     → Name
  context-free syntax
    sort Name+                                                     → Sort-specification

    func Function-declaration+                                     → Function-specification
    {Name ","}+ ":" {Name "#"}* "→" Name                           → Function-declaration

    Variable-declaration-section Rewrite-rules-section             → Rewrite-specification
    var Variable-declaration+                                      → Variable-declaration-section
                                                                   → Variable-declaration-section
    {Name ","}+ ":" Name                                           → Variable-declaration
    Name                                                           → Data-term
    Name "(" {Data-term ","}+ ")"                                  → Data-term
    rew Rewrite-rule+                                              → Rewrite-rules-section
    Name "=" Data-term                                             → Rewrite-rule
    Name "(" {Data-term ","}+ ")" "=" Data-term                    → Rewrite-rule

    "◁" Data-term "▷"                                              → Condition-operator

    Process-expression "+" Process-expression                      → Process-expression   {assoc}
    Process-expression "." Process-expression                      → Process-expression   {assoc}
    Process-expression "||" Process-expression                     → Process-expression   {assoc}
    Process-expression "⌊⌊" Process-expression                     → Process-expression   {non-assoc}
    Process-expression "|" Process-expression                      → Process-expression   {assoc}
    Process-expression Condition-operator Process-expression       → Process-expression   {non-assoc}
    "(" Process-expression ")"                                     → Process-expression   {bracket}

    delta                                                          → Process-expression
    tau                                                            → Process-expression
    encap "(" "{" {Name ","}+ "}" "," Process-expression ")"       → Process-expression
    hide "(" "{" {Name ","}+ "}" "," Process-expression ")"        → Process-expression
    rename "(" "{" {Renaming-declaration ","}+ "}" ","
    Process-expression ")"                                         → Process-expression
    sum "(" Single-variable-declaration "," Process-expression ")" → Process-expression
    Name                                                           → Process-expression
    Name "(" {Data-term ","}+ ")"                                  → Process-expression

    Name "→" Name                                                  → Renaming-declaration

    Name ":" Name                                                  → Single-variable-declaration
```

proc Process-declaration+ $\rightarrow$ Process-specification
Name "=" Process-expression $\rightarrow$ Process-declaration
Name "(" {Single-variable-declaration ","}+ ")" "="
Process-expression $\rightarrow$ Process-declaration

act Action-declaration+ $\rightarrow$ Action-specification
Name $\rightarrow$ Action-declaration
{Name ","}+ ":" {Name "#"}+ $\rightarrow$ Action-declaration

comm Communication-declaration+ $\rightarrow$ Communication-specification
Name "|" Name "=" Name $\rightarrow$ Communication-declaration

Sort-specification $\rightarrow$ Single-specification
Function-specification $\rightarrow$ Single-specification
Rewrite-specification $\rightarrow$ Single-specification
Action-specification $\rightarrow$ Single-specification
Communication-specification $\rightarrow$ Single-specification
Process-specification $\rightarrow$ Single-specification

Single-specification+ $\rightarrow$ Specification

priorities

Process-expression "." Process-expression $\rightarrow$ Process-expression
$>$ Process-expression Condition-operator Process-expression $\rightarrow$ Process-expression
$>$ {**non-assoc**: Process-expression "$\|$" Process-expression $\rightarrow$ Process-expression,
Process-expression "$\mathbin{\|\!\_}$" Process-expression $\rightarrow$ Process-expression,
Process-expression "|" Process-expression $\rightarrow$ Process-expression}
$>$ Process-expression "+" Process-expression $\rightarrow$ Process-expression

A.2 MiCRL

imports MuCRL$^{A.1}$
exports
context-free syntax

Process-expression "$|||$" Process-expression $\rightarrow$ Process-expression {**assoc**}
Process-expression "$\mathbin{|||\!\_}$" Process-expression $\rightarrow$ Process-expression {**non-assoc**}
Process-expression "$*$" Process-expression $\rightarrow$ Process-expression {**non-assoc**}

priorities

{**non-assoc**: Process-expression "." Process-expression $\rightarrow$ Process-expression,
Process-expression "$*$" Process-expression $\rightarrow$ Process-expression}
$>$ Process-expression Condition-operator Process-expression $\rightarrow$ Process-expression
$>$ {**non-assoc**: Process-expression "$\|$" Process-expression $\rightarrow$ Process-expression,
Process-expression "$\mathbin{\|\!\_}$" Process-expression $\rightarrow$ Process-expression,
Process-expression "|" Process-expression $\rightarrow$ Process-expression,
Process-expression "$|||$" Process-expression $\rightarrow$ Process-expression,
Process-expression "$\mathbin{|||\!\_}$" Process-expression $\rightarrow$ Process-expression}
$>$ Process-expression "+" Process-expression $\rightarrow$ Process-expression

A.3 GPSL

```
imports MiCRL^A.2
exports
  sorts Process-specification-gpsl
  context-free syntax
    Process-expression ";" Process-expression           → Process-expression          {non-assoc}
    er "(" Data-term "," {Name ","}+ ")"               → Process-expression

    sum "(" Name "," Process-expression ")"            → Process-expression

    Name "(" {Name ","}+ ")" "=" Process-expression   → Process-declaration

    Variable-declaration-section Process-specification → Process-specification-gpsl

    Process-specification-gpsl                          → Single-specification
  priorities
    {non-assoc: Process-expression "." Process-expression → Process-expression,
    Process-expression ";" Process-expression → Process-expression,
    Process-expression "*" Process-expression → Process-expression}
    > Process-expression Condition-operator Process-expression → Process-expression
    > {non-assoc: Process-expression "||" Process-expression → Process-expression,
    Process-expression "⌊⌊" Process-expression → Process-expression,
    Process-expression "|" Process-expression → Process-expression,
    Process-expression "|||" Process-expression → Process-expression,
    Process-expression "|⌊⌊" Process-expression → Process-expression}
    > Process-expression "+" Process-expression → Process-expression
```

A.4 Signature-GPSL

imports GPSL$^{A.3}$ Signature
exports
  sorts Single-variable-declarations
  context-free syntax

{Single-variable-declaration ","}+	→ Single-variable-declarations
sig-gpsl(Specification)	→ Signature
make-headers(Var, Process-declaration+)	→ Proc
make-svds(Var, {Name ","}+)	→ Single-variable-declarations
is-variable(Var, Name)	→ BOOL

  variables
    Globvar [*0-9′*]∗ → Var
    Locvar [*0-9′*]∗ → Var
equations

$$\text{sig-gpsl(sort } \mathit{Ns}) = ([\mathit{Ns}], [], [], [], []) \quad \text{[sig-gpsl-1]}$$
$$\text{sig-gpsl(func } \mathit{Fds}) = ([], [\mathit{Fds}], [], [], []) \quad \text{[sig-gpsl-2]}$$
$$\text{sig-gpsl}(\mathit{Rew\text{-}spec}) = ([], [], [], [], []) \quad \text{[sig-gpsl-3]}$$
$$\text{sig-gpsl(act } \mathit{Ads}) = ([], [], [\mathit{Ads}], [], []) \quad \text{[sig-gpsl-4]}$$
$$\text{sig-gpsl(comm } \mathit{Cds}) = ([], [], [], [\mathit{Cds}], []) \quad \text{[sig-gpsl-5]}$$

$$\frac{\mathit{Proc} = \text{make-headers}([], \mathit{Pds})}{\text{sig-gpsl(proc } \mathit{Pds}) = ([], [], [], [], \mathit{Proc})} \quad \text{[sig-gpsl-6]}$$

$$\frac{\mathit{Proc} = \text{make-headers}([\mathit{Vds}], \mathit{Pds})}{\text{sig-gpsl(var } \mathit{Vds} \text{ proc } \mathit{Pds}) = ([], [], [], [], \mathit{Proc})} \quad \text{[sig-gpsl-7]}$$

$$\frac{\begin{array}{c}\text{sig-gpsl(SE)} = (\mathit{Names}_1, \mathit{Fun}_1, \mathit{Act}_1, \mathit{Comm}_1, \mathit{Proc}_1), \\ \text{sig-gpsl}(\mathit{SEs}) = (\mathit{Names}_2, \mathit{Fun}_2, \mathit{Act}_2, \mathit{Comm}_2, \mathit{Proc}_2)\end{array}}{\begin{array}{rl}\text{sig-gpsl(SE } \mathit{SEs}) = & (\mathit{Names}_1 + \mathit{Names}_2, \\ & \mathit{Fun}_1 + \mathit{Fun}_2, \\ & \mathit{Act}_1 + \mathit{Act}_2, \\ & \mathit{Comm}_1 + \mathit{Comm}_2, \\ & \mathit{Proc}_1 + \mathit{Proc}_2)\end{array}} \quad \text{[sig-gpsl-8]}$$

$$[\mathit{OVds}_1\ N, \mathit{Cns} : N'\ \mathit{OVds}_2] = [\mathit{OVds}_1\ N : N'\ \mathit{Cns} : N'\ \mathit{OVds}_2] \quad \text{[vds-1]}$$

$$\text{make-headers}(\mathit{Var}, N = \mathit{Pre}) = [N = \mathit{Pre}] \quad \text{[headers-1]}$$
$$\text{make-headers}(\mathit{Var}, N(\mathit{Svds}) = \mathit{Pre}) = [N(\mathit{Svds}) = \mathit{Pre}] \quad \text{[headers-2]}$$

$$\frac{\mathit{Svds} = \text{make-svds}(\mathit{Var}, \mathit{Cns})}{\text{make-headers}(\mathit{Var}, N(\mathit{Cns}) = \mathit{Pre}) = [N(\mathit{Svds}) = \mathit{Pre}]} \quad \text{[headers-3]}$$

$$\frac{\mathit{Proc}_1 \;=\; \textsf{make-headers}(\mathit{Var},\,\mathit{Pd}),\quad \mathit{Proc}_2 \;=\; \textsf{make-headers}(\mathit{Var},\,\mathit{Pds})}{\textsf{make-headers}(\mathit{Var},\,\mathit{Pd}\ \mathit{Pds}) \;=\; \mathit{Proc}_1 + \mathit{Proc}_2} \qquad \text{[headers-4]}$$

$$\textsf{make-svds}([\mathit{OVds}_1\ N : N'\ \mathit{OVds}_2],\,N) = N : N' \qquad \text{[make-svds-1]}$$

$$\frac{\textsf{is-variable}(\mathit{Var},\,N) \;\neq\; \textsf{true}}{\textsf{make-svds}(\mathit{Var},\,N) \;=\; N : \texttt{unknown}} \qquad \text{[make-svds-2]}$$

$$\frac{\mathit{Svds}_1 \;=\; \textsf{make-svds}(\mathit{Var},\,N),\quad \mathit{Svds}_2 \;=\; \textsf{make-svds}(\mathit{Var},\,\mathit{Cns})}{\textsf{make-svds}(\mathit{Var},\,N,\,\mathit{Cns}) \;=\; \mathit{Svds}_1,\,\mathit{Svds}_2} \qquad \text{[make-svds-3]}$$

$$\textsf{is-variable}([\mathit{OVds}_1\ N : N'\ \mathit{OVds}_2],\,N) = \textsf{true} \qquad \text{[is-variable-1]}$$

A.5 GPSL-aux

```
imports Signature-GPSL^A.4
exports
  sorts X-name-list Input-pattern Types
  context-free syntax
    {Name "#"}+                                              -> X-name-list

    empty-pattern                                            -> Input-pattern
    {Name "#"}+                                              -> Input-pattern

    "[" Input-pattern* "]"                                   -> Types

    get-locals(Var, {Name ","}+)                             -> Var
    add-to-locals(Var, Var, {Name ","}*)                     -> Var

    er-vars(Var, Var, Process-expression)                    -> Var

    sort(Signature, Var, Data-term)                          -> Name
    sort-list(Signature, Var, {Data-term ","}+)              -> X-name-list

    is-function "(" Fun "," Name ":" X-name-list ")"         -> BOOL
    is-process "(" Proc "," Name ")"                         -> BOOL
    is-process "(" Proc "," Name "(" {Name "#"}+ ")" ")"     -> BOOL

    Types "+" Types                                          -> Types        {right}
    Types "−" Types                                          -> Types        {right}
    Input-pattern in Types                                   -> BOOL

    types(Act, Name)                                         -> Types
    er-types(Signature, Specification)                       -> Types

    transform({Single-variable-declaration ","}+)            -> Var

    Var "|" Name                                             -> Var

    is-action "(" Act "," Name ":" X-name-list ")"           -> BOOL
    is-comm "(" Comm "," Name "|" Name ")"                   -> BOOL
  variables
    Xlist [0-9']*  -> X-name-list

    Ip [0-9']*     -> Input-pattern
    Ips [0-9']*    -> Input-pattern+
    OIps [0-9']*   -> Input-pattern*

    Types [0-9']*  -> Types
hiddens
  sorts Comma-name-list
  context-free syntax
    {Name ","}*                                                -> Comma-name-list

    collect-names(Process-expression)                          -> Comma-name-list

    is-function(Fun, Name)                                     -> BOOL

    var-sorts "(" "[" {Single-variable-declaration ","}+ "]" ")" -> X-name-list
```

er-types-pd(Signature, Var, Process-declaration+) → Types
er-types-expr(Signature, Var, Var, Process-expression) → Types

equations

$$\text{get-locals}([\mathit{OVds}_1\ N : N'\ \mathit{OVds}_2], N) = [N : N'] \qquad \text{[get-locals-1]}$$

$$\frac{\text{is-variable}(\mathit{Var}, N) \neq \text{true}}{\text{get-locals}(\mathit{Var}, N) = [N : \texttt{unknown}]} \qquad \text{[get-locals-2]}$$

$$\text{get-locals}(\mathit{Var}, N, \mathit{Cns}) = \text{get-locals}(\mathit{Var}, N) + \text{get-locals}(\mathit{Var}, \mathit{Cns}) \qquad \text{[get-locals-3]}$$

$$\text{add-to-locals}(\mathit{Globvar}, \mathit{Locvar},) = \mathit{Locvar} \qquad \text{[add-to-locals-1]}$$

$$\frac{N : N' = \text{make-svds}(\mathit{Globvar}, N)}{\begin{array}{l}\text{add-to-locals}(\mathit{Globvar}, \mathit{Locvar}, N, \mathit{OCns}) = \\ \text{add-to-locals}(\mathit{Globvar}, \mathit{Locvar} \mid N + [N : N'], \mathit{OCns})\end{array}} \qquad \text{[add-to-locals-2]}$$

$$\frac{\mathit{OCns} = \text{collect-names}(\mathit{Pre})}{\text{er-vars}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}) = \text{add-to-locals}(\mathit{Globvar}, \mathit{Locvar}, \mathit{OCns})} \qquad \text{[er-vars-1]}$$

$$\frac{\begin{array}{l}\mathit{OCns}_1 = \text{collect-names}(\mathit{Pre}_1), \\ \mathit{OCns}_2 = \text{collect-names}(\mathit{Pre}_2)\end{array}}{\text{collect-names}(\mathit{Pre}_1 + \mathit{Pre}_2) = \mathit{OCns}_1, \mathit{OCns}_2} \qquad \text{[collect-1]}$$

$$\frac{\begin{array}{l}\mathit{OCns}_1 = \text{collect-names}(\mathit{Pre}_1), \\ \mathit{OCns}_2 = \text{collect-names}(\mathit{Pre}_2)\end{array}}{\text{collect-names}(\mathit{Pre}_1 \mathbin{|||} \mathit{Pre}_2) = \mathit{OCns}_1, \mathit{OCns}_2} \qquad \text{[collect-2]}$$

$$\frac{\begin{array}{l}\mathit{OCns}_1 = \text{collect-names}(\mathit{Pre}_1), \\ \mathit{OCns}_2 = \text{collect-names}(\mathit{Pre}_2)\end{array}}{\text{collect-names}(\mathit{Pre}_1 \mathbin{|||\!\_} \mathit{Pre}_2) = \mathit{OCns}_1, \mathit{OCns}_2} \qquad \text{[collect-3]}$$

$$\frac{\begin{array}{l}\mathit{OCns}_1 = \text{collect-names}(\mathit{Pre}_1), \\ \mathit{OCns}_2 = \text{collect-names}(\mathit{Pre}_2)\end{array}}{\text{collect-names}(\mathit{Pre}_1 \triangleleft \mathit{Trm} \triangleright \mathit{Pre}_2) = \mathit{OCns}_1, \mathit{OCns}_2} \qquad \text{[collect-4]}$$

$$\text{collect-names}(\mathit{Pre}_1 ; \mathit{Pre}_2) = \text{collect-names}(\mathit{Pre}_2) \qquad \text{[collect-5]}$$

$$\frac{\begin{array}{l}\mathit{OCns}_1 = \text{collect-names}(\mathit{Pre}_1), \\ \mathit{OCns}_2 = \text{collect-names}(\mathit{Pre}_2)\end{array}}{\text{collect-names}(\mathit{Pre}_1 \,.\, \mathit{Pre}_2) = \mathit{OCns}_1, \mathit{OCns}_2} \qquad \text{[collect-6]}$$

$$\begin{array}{lll}
\text{collect-names(delta)} & = & \quad [\text{collect-7}] \\
\text{collect-names(tau)} & = & \quad [\text{collect-8}] \\
\text{collect-names(encap(}\{\mathit{Cns}\}, \mathit{Pre}\text{))} & = \text{collect-names(}\mathit{Pre}\text{)} & \quad [\text{collect-9}] \\
\text{collect-names(hide(}\{\mathit{Cns}\}, \mathit{Pre}\text{))} & = \text{collect-names(}\mathit{Pre}\text{)} & \quad [\text{collect-10}] \\
\text{collect-names(rename(}\{\mathit{Rds}\}, \mathit{Pre}\text{))} & = \text{collect-names(}\mathit{Pre}\text{)} & \quad [\text{collect-11}] \\
\text{collect-names(sum(}N, \mathit{Pre}\text{))} & = \text{collect-names(}\mathit{Pre}\text{)} & \quad [\text{collect-12}] \\
\text{collect-names(}N\text{)} & = & \quad [\text{collect-13}] \\
\text{collect-names(}N(\mathit{Trms})\text{)} & = & \quad [\text{collect-14}] \\
\text{collect-names(er(}\mathit{Trm}, \mathit{Cns}\text{))} & = \mathit{Cns} & \quad [\text{collect-15}]
\end{array}$$

$$\frac{\begin{array}{rcl} \mathit{OCns}_1 & = & \text{collect-names}(\mathit{Pre}_1), \\ \mathit{OCns}_2 & = & \text{collect-names}(\mathit{Pre}_2) \end{array}}{\text{collect-names}(\mathit{Pre}_1 \mathbin{\lfloor\!\lfloor} \mathit{Pre}_2) \ = \ \mathit{OCns}_1, \mathit{OCns}_2} \quad [\text{collect-16}]$$

$$\frac{\begin{array}{rcl} \mathit{OCns}_1 & = & \text{collect-names}(\mathit{Pre}_1), \\ \mathit{OCns}_2 & = & \text{collect-names}(\mathit{Pre}_2) \end{array}}{\text{collect-names}(\mathit{Pre}_1 \parallel \mathit{Pre}_2) \ = \ \mathit{OCns}_1, \mathit{OCns}_2} \quad [\text{collect-17}]$$

$$\frac{\begin{array}{rcl} \mathit{OCns}_1 & = & \text{collect-names}(\mathit{Pre}_1), \\ \mathit{OCns}_2 & = & \text{collect-names}(\mathit{Pre}_2) \end{array}}{\text{collect-names}(\mathit{Pre}_1 \mid \mathit{Pre}_2) \ = \ \mathit{OCns}_1, \mathit{OCns}_2} \quad [\text{collect-18}]$$

$$\frac{\begin{array}{rcl} \mathit{OCns}_1 & = & \text{collect-names}(\mathit{Pre}_1), \\ \mathit{OCns}_2 & = & \text{collect-names}(\mathit{Pre}_2) \end{array}}{\text{collect-names}(\mathit{Pre}_1 * \mathit{Pre}_2) \ = \ \mathit{OCns}_1, \mathit{OCns}_2} \quad [\text{collect-19}]$$

$$\text{collect-names(sum(}\mathit{Svd}, \mathit{Pre}\text{))} = \text{collect-names(}\mathit{Pre}\text{)} \quad [\text{collect-20}]$$

$$\text{sort}(\mathit{Sig}, [\mathit{OVds}_1 \ N : N' \ \mathit{OVds}_2], N) = N' \quad [\text{sort-1}]$$

$$\frac{\text{sigfun}(\mathit{Sig}) \ = \ [\mathit{OFds}_1 \ N : \ \rightarrow N' \ \mathit{OFds}_2]}{\text{sort}(\mathit{Sig}, \mathit{Var}, N) \ = \ N'} \quad [\text{sort-2}]$$

$$\frac{\begin{array}{rcl} \mathit{Xns} & = & \text{sort-list}(\mathit{Sig}, \mathit{Var}, \mathit{Trms}), \\ \text{sigfun}(\mathit{Sig}) & = & [\mathit{OFds}_1 \ N : \mathit{Xns} \rightarrow N' \ \mathit{OFds}_2] \end{array}}{\text{sort}(\mathit{Sig}, \mathit{Var}, N(\mathit{Trms})) \ = \ N'} \quad [\text{sort-3}]$$

The next two equations describe hypothetical cases.
They have been added to make the function ‘sort’ complete.

$$\frac{\begin{array}{rcl} \text{is-variable}(\mathit{Var}, N) & \neq & \text{true}, \\ \text{is-function}(\text{sigfun}(\mathit{Sig}), N) & \neq & \text{true} \end{array}}{\text{sort}(\mathit{Sig}, \mathit{Var}, N) \ = \ \texttt{unknown}} \quad [\text{sort-4}]$$

$$\frac{\begin{array}{r} \mathit{Xns} \ = \ \text{sort-list}(\mathit{Sig},\ \mathit{Var},\ \mathit{Trms}), \\ \text{is-function}(\text{sigfun}(\mathit{Sig}),\ N : \mathit{Xns}) \ \neq \ \text{true} \end{array}}{\text{sort}(\mathit{Sig},\ \mathit{Var},\ N(\mathit{Trms})) \ = \ \texttt{unknown}} \qquad \text{[sort-5]}$$

$$\text{sort-list}(\mathit{Sig},\ \mathit{Var},\ \mathit{Trm}) = \text{sort}(\mathit{Sig},\ \mathit{Var},\ \mathit{Trm}) \qquad \text{[sort-list-1]}$$

$$\frac{\mathit{Xns} \ = \ \text{sort-list}(\mathit{Sig},\ \mathit{Var},\ \mathit{Trms})}{\text{sort-list}(\mathit{Sig},\ \mathit{Var},\ \mathit{Trm},\ \mathit{Trms}) \ = \ \text{sort}(\mathit{Sig},\ \mathit{Var},\ \mathit{Trm})\#\ \mathit{Xns}} \qquad \text{[sort-list-2]}$$

$$\text{is-function}([\mathit{OFds}_1\ \ N : \ \rightarrow N'\ \ \mathit{OFds}_2],\ N) = \text{true} \qquad \text{[is-function-noargs-1]}$$

$$\text{is-function}([\mathit{OFds}_1\ \ N : \mathit{Xns} \rightarrow N'\ \ \mathit{OFds}_2],\ N : \mathit{Xns}) = \text{true} \qquad \text{[is-function-args-1]}$$

$$\text{is-process}([\mathit{OPds}_1\ \ N = \mathit{Pre}\ \mathit{OPds}_2],\ N) = \text{true} \qquad \text{[is-process-1]}$$

$$\frac{\mathit{Xns} \ = \ \text{var-sorts}([\mathit{Svds}])}{\text{is-process}([\mathit{OPds}_1\ \ N(\mathit{Svds}) = \mathit{Pre}\ \mathit{OPds}_2],\ N(\mathit{Xns})) \ = \ \text{true}} \qquad \text{[is-process-2]}$$

$$\text{var-sorts}([N : N']) = N' \qquad \text{[var-sorts-1]}$$

$$\frac{\mathit{Xns} \ = \ \text{var-sorts}([\mathit{Svds}])}{\text{var-sorts}([N : N',\ \mathit{Svds}]) \ = \ N'\#\ \mathit{Xns}} \qquad \text{[var-sorts-2]}$$

$$[\mathit{OIps}_1] + [\mathit{OIps}_2] = [\mathit{OIps}_1\ \ \mathit{OIps}_2] \qquad \text{[types-union-1]}$$

$$[] - \mathit{Types} = [] \qquad \text{[typesdiff-1]}$$

$$\frac{\mathit{Ip}\ \text{in}\ \mathit{Types} \ = \ \text{true}}{[\mathit{Ip}\ \mathit{OIps}] - \mathit{Types} \ = \ [\mathit{OIps}] - \mathit{Types}} \qquad \text{[typesdiff-2]}$$

$$\frac{\begin{array}{r} \mathit{Ip}\ \text{in}\ \mathit{Types} \ \neq \ \text{true}, \\ \mathit{Types}' \ = \ [\mathit{OIps}] - \mathit{Types} \end{array}}{[\mathit{Ip}\ \mathit{OIps}] - \mathit{Types} \ = \ [\mathit{Ip}] + \mathit{Types}'} \qquad \text{[typesdiff-3]}$$

$$\mathit{Ip}\ \text{in}\ [\mathit{OIps}_1\ \ \mathit{Ip}\ \ \mathit{OIps}_2] = \text{true} \qquad \text{[in-types-1]}$$

$$\begin{array}{lll} \text{types}([],\ N) & = [] & \text{[types-1]} \\ \text{types}([N\ \mathit{OAds}],\ N) & = [\text{empty-pattern}] + \text{types}([\mathit{OAds}],\ N) & \text{[types-2]} \\ \text{types}([N : \mathit{Xns}\ \mathit{OAds}],\ N) & = [\mathit{Xns}] + \text{types}([\mathit{OAds}],\ N) & \text{[types-3]} \end{array}$$

$$\frac{N' \neq N}{\text{types}([N'\ \mathit{OAds}],\ N)\ =\ \text{types}([\mathit{OAds}],\ N)} \qquad \text{[types-4]}$$

$$\frac{N' \neq N}{\text{types}([N' : \mathit{Xns}\ \mathit{OAds}],\ N)\ =\ \text{types}([\mathit{OAds}],\ N)} \qquad \text{[types-5]}$$

$$\begin{array}{lll}
\text{er-types}(\mathit{Sig},\ \text{sort}\ \mathit{Ns}) & = [] & \text{[er-types-1]}\\
\text{er-types}(\mathit{Sig},\ \text{func}\ \mathit{Fds}) & = [] & \text{[er-types-2]}\\
\text{er-types}(\mathit{Sig},\ \mathit{Rew\text{-}spec}) & = [] & \text{[er-types-3]}\\
\text{er-types}(\mathit{Sig},\ \text{act}\ \mathit{Ads}) & = [] & \text{[er-types-4]}\\
\text{er-types}(\mathit{Sig},\ \text{comm}\ \mathit{Cds}) & = [] & \text{[er-types-5]}\\
\text{er-types}(\mathit{Sig},\ \text{proc}\ \mathit{Pds}) & = \text{er-types-pd}(\mathit{Sig},\ [],\ \mathit{Pds}) & \text{[er-types-6]}\\
\text{er-types}(\mathit{Sig},\ \text{var}\ \mathit{Vds}\ \text{proc}\ \mathit{Pds}) & = \text{er-types-pd}(\mathit{Sig},\ [\mathit{Vds}],\ \mathit{Pds}) & \text{[er-types-7]}
\end{array}$$

$$\frac{\begin{array}{l}\mathit{Types}_1\ =\ \text{er-types}(\mathit{Sig},\ \text{SE}),\\ \mathit{Types}_2\ =\ \text{er-types}(\mathit{Sig},\ \mathit{SEs})\end{array}}{\text{er-types}(\mathit{Sig},\ \text{SE}\ \mathit{SEs})\ =\ \mathit{Types}_1\ +\ \mathit{Types}_2} \qquad \text{[er-types-8]}$$

$$\text{er-types-pd}(\mathit{Sig},\ \mathit{Globvar},\ N = \mathit{Pre}) = \text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ [],\ \mathit{Pre}) \qquad \text{[types-pd-1]}$$

$$\begin{array}{ll}\text{er-types-pd}(\mathit{Sig},\ \mathit{Globvar},\ N(\mathit{Svds}) = \mathit{Pre}) & \text{[types-pd-2]}\\ =\ \text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ \text{transform}(\mathit{Svds}),\ \mathit{Pre}) & \end{array}$$

$$\begin{array}{ll}\text{er-types-pd}(\mathit{Sig},\ \mathit{Globvar},\ N(\mathit{Cns}) = \mathit{Pre}) & \text{[types-pd-3]}\\ =\ \text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ \text{get-locals}(\mathit{Globvar},\ \mathit{Cns}),\ \mathit{Pre}) & \end{array}$$

$$\frac{\begin{array}{l}\mathit{Types}_1\ =\ \text{er-types-pd}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Pd}),\\ \mathit{Types}_2\ =\ \text{er-types-pd}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Pds})\end{array}}{\text{er-types-pd}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Pd}\ \mathit{Pds})\ =\ \mathit{Types}_1\ +\ \mathit{Types}_2} \qquad \text{[types-pd-4]}$$

$$\begin{array}{ll}\text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Locvar},\ \mathit{Pre}_1\ +\ \mathit{Pre}_2) & \text{[types-expr-1]}\\ =\ \text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Locvar},\ \mathit{Pre}_1) & \\ \quad +\ \text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Locvar},\ \mathit{Pre}_2) & \end{array}$$

$$\begin{array}{ll}\text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Locvar},\ \mathit{Pre}_1\ \mathbin{\|\!\lfloor}\ \mathit{Pre}_2) & \text{[types-expr-2]}\\ =\ \text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Locvar},\ \mathit{Pre}_1) & \\ \quad +\ \text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Locvar},\ \mathit{Pre}_2) & \end{array}$$

$$\begin{array}{ll}\text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Locvar},\ \mathit{Pre}_1 \parallel \mathit{Pre}_2) & \text{[types-expr-3]}\\ =\ \text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Locvar},\ \mathit{Pre}_1) & \\ \quad +\ \text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Locvar},\ \mathit{Pre}_2) & \end{array}$$

$$\text{er-types-expr}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Locvar},\ \mathit{Pre}_1\ |\ \mathit{Pre}_2) \qquad \text{[types-expr-4]}$$

$= \text{er-types-expr}(Sig, Globvar, Locvar, Pre_1)$
$\quad + \text{er-types-expr}(Sig, Globvar, Locvar, Pre_2)$

$\text{er-types-expr}(Sig, Globvar, Locvar, Pre_1 \; ||| \; Pre_2)$ [types-expr-5]
$= \text{er-types-expr}(Sig, Globvar, Locvar, Pre_1)$
$\quad + \text{er-types-expr}(Sig, Globvar, Locvar, Pre_2)$

$\text{er-types-expr}(Sig, Globvar, Locvar, Pre_1 \; \underline{|||} \; Pre_2)$ [types-expr-6]
$= \text{er-types-expr}(Sig, Globvar, Locvar, Pre_1)$
$\quad + \text{er-types-expr}(Sig, Globvar, Locvar, Pre_2)$

$\text{er-types-expr}(Sig, Globvar, Locvar, Pre_1 \triangleleft Trm \triangleright Pre_2)$ [types-expr-7]
$= \text{er-types-expr}(Sig, Globvar, Locvar, Pre_1)$
$\quad + \text{er-types-expr}(Sig, Globvar, Locvar, Pre_2)$

$\text{er-types-expr}(Sig, Globvar, Locvar, Pre_1; Pre_2)$ [types-expr-8]
$= \text{er-types-expr}(Sig, Globvar, Locvar, Pre_1)$
$\quad + \text{er-types-expr}(Sig, Globvar, \text{er-vars}(Globvar, Locvar, Pre_1), Pre_2)$

$\text{er-types-expr}(Sig, Globvar, Locvar, Pre_1 \, . \, Pre_2)$ [types-expr-9]
$= \text{er-types-expr}(Sig, Globvar, Locvar, Pre_1)$
$\quad + \text{er-types-expr}(Sig, Globvar, Locvar, Pre_2)$

$\text{er-types-expr}(Sig, Globvar, Locvar, Pre_1 * Pre_2)$ [types-expr-10]
$= \text{er-types-expr}(Sig, Globvar, Locvar, Pre_1)$
$\quad + \text{er-types-expr}(Sig, Globvar, Locvar, Pre_2)$

$\text{er-types-expr}(Sig, Globvar, Locvar, \text{delta}) = []$ [types-expr-11]
$\text{er-types-expr}(Sig, Globvar, Locvar, \text{tau}) \quad = []$ [types-expr-12]

$\text{er-types-expr}(Sig, Globvar, Locvar, \text{encap}(\{Cns\}, Pre))$ [types-expr-13]
$= \text{er-types-expr}(Sig, Globvar, Locvar, Pre)$

$\text{er-types-expr}(Sig, Globvar, Locvar, \text{hide}(\{Cns\}, Pre))$ [types-expr-14]
$= \text{er-types-expr}(Sig, Globvar, Locvar, Pre)$

$\text{er-types-expr}(Sig, Globvar, Locvar, \text{rename}(\{Rds\}, Pre))$ [types-expr-15]
$= \text{er-types-expr}(Sig, Globvar, Locvar, Pre)$

$\text{er-types-expr}(Sig, Globvar, Locvar, \text{sum}(N, Pre))$ [types-expr-16]
$= \text{er-types-expr}(Sig, Globvar, \text{add-to-locals}(Globvar, Locvar, N), Pre)$

$\text{er-types-expr}(Sig, Globvar, Locvar, \text{sum}(N : N', Pre))$ [types-expr-17]
$= \text{er-types-expr}(Sig, Globvar, Locvar \mid N + [N : N'], Pre)$

$$\text{er-types-expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{N}) = [] \quad \text{[types-expr-18]}$$
$$\text{er-types-expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{N}(\mathit{Trms})) = [] \quad \text{[types-expr-19]}$$

$$\frac{\begin{array}{rcl} \mathit{N} & = & \text{sort}(\mathit{Sig}, \mathit{Locvar}, \mathit{Trm}), \\ \mathit{Svds} & = & \text{make-svds}(\mathit{Globvar}, \mathit{Cns}), \\ \mathit{Xns} & = & \text{var-sorts}([\mathit{Svds}]) \end{array}}{\text{er-types-expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \text{er}(\mathit{Trm}, \mathit{Cns})) \ = \ [\mathit{N}\# \ \mathit{Xns}]} \quad \text{[types-expr-20]}$$

$$\text{transform}(\mathit{N} : \mathit{N}') = [\mathit{N} : \mathit{N}'] \quad \text{[transf-svds-1]}$$
$$\text{transform}(\mathit{Svd}, \mathit{Svds}) = \text{transform}(\mathit{Svd}) + \text{transform}(\mathit{Svds}) \quad \text{[transf-svds-2]}$$

$$[\mathit{OVds}_1 \ \mathit{N} : \mathit{N}' \ \mathit{OVds}_2] \mid \mathit{N} = [\mathit{OVds}_1 \ \mathit{OVds}_2] \mid \mathit{N} \quad \text{[var-remove-1]}$$

$$\frac{\text{is-variable}(\mathit{Var}, \mathit{N}) \ \neq \ \text{true}}{\mathit{Var} \mid \mathit{N} \ = \ \mathit{Var}} \quad \text{[var-remove-2]}$$

$$\text{is-action}([\mathit{OAds}_1 \ \mathit{N} : \mathit{Xns} \ \mathit{OAds}_2], \mathit{N} : \mathit{Xns}) = \text{true} \quad \text{[is-action-1]}$$

$$\text{is-comm}([\mathit{OCds}_1 \ \mathit{N}_1 \mid \mathit{N}_2 = \mathit{N}_3 \ \mathit{OCds}_2], \mathit{N}_1 \mid \mathit{N}_2) = \text{true} \quad \text{[is-comm-1]}$$
$$\text{is-comm}([\mathit{OCds}_1 \ \mathit{N}_2 \mid \mathit{N}_1 = \mathit{N}_3 \ \mathit{OCds}_2], \mathit{N}_1 \mid \mathit{N}_2) = \text{true} \quad \text{[is-comm-2]}$$

A.6 GPSL-errors

imports Errors
exports
context-free syntax

“Syntax error” “in process header” Name “(” {Single-variable-declaration “,”}+ “).”
“Use names” “instead of” “variable declarations” → Error

“Unknown variable” Name “in process header” → Error

“Unknown variable” Name “as argument” “of sum” → Error

“ACP operator” “$\lfloor\!\lfloor$” “used in” “left-hand-side of” “‘;’.” “Use PA operator” “$|\!|\!\lfloor$” → Error

“ACP operator” “$\|$” “used in” “left-hand-side of” “‘;’.” “Use PA operator” “$|\!|\!|$” → Error

“Communication used” “in left-hand-side of” “;” → Error

“Iteration used” “in left-hand-side of” “;” → Error

“Syntax error” “in sum construction” “sum(” Single-variable-declaration “, xxx).”
“Use name” “instead of” “variable declaration” → Error

“Process” Name “called in” “left-hand-side of” “;” → Error

“Process” Name “(” {Name “#”}+ “)” “called in” “left-hand-side of” “;” → Error

“Early read” “used outside” “scope of” “;” → Error

“Unknown variable” Name “as argument” “of ‘er’” → Error

“Could not” “determine sort” “of data-term” Data-term “in ‘er’” → Error

“Function” Name “:” {Name “#”}+ “has the” “same input-pattern” “as an” “er” → Error

“Process” Name “(” {Name “#”}+ “)” “has the” “same input-pattern” “as an” “er” → Error

“Result” Name “in read-send communication” “r | s =” Name “should be” “c” → Error

“Action s” “should be” “declared to” “match ‘er’” → Error

“Action s:” {Name “#”}+ “should be” “declared to” “match ‘er’” → Error

“Action s” “should be” “declared to” “match ‘r’” → Error

“Action s:” {Name “#”}+ “should be” “declared to” “match ‘r’” → Error

“Action r” “should be” “declared to” “match ‘s’” → Error

“Action r:” {Name “#”}+ “should be” “declared to” “match ‘s’” → Error

“Action c” “should be” “declared to” “match ‘r’” “and ‘s’” → Error

“Action c:” {Name “#”}+ “should be” “declared to” “match ‘r’” “and ‘s’” → Error

“Special sort name” “‘Er-aux-port’ used” → Error

“Special constant” “starting with” “‘er-aux-port’ used” → Error

"Special action name" "'er-aux-r' used" → Error

"Special action name" "'er-aux-s' used" → Error

"Special action name" "'er-aux-c' used" → Error

A.7 Precheck

imports GPSL-errors$^{A.6}$ GPSL-aux$^{A.5}$
exports
  context-free syntax
    precheck(Specification) → Errors
hiddens
  sorts Scope
  context-free syntax

```
inscope                                                   → Scope
outscope                                                  → Scope

precheck(Signature, Specification)                        → Errors

inspect-procspecs(Specification)                          → Errors
inspect-headers(Var, Process-declaration+)                → Errors
inspect-args(Var, {Name ","}+)                            → Errors
uniq-vars(Var)                                            → Errors

precheck-spec(Signature, Specification)                   → Errors
precheck-pd(Signature, Var, Process-declaration+)         → Errors
precheck-expr(Signature, Var, Var, Process-expression, Scope) → Errors
precheck-sum(Var, Name)                                   → Error
precheck-er(Signature, Var, Var, Data-term, {Name ","}+)  → Errors
precheck-er(Var, {Name ","}+)                             → Errors

no-name-clash(Signature, Types)                           → Errors
no-name-clash(Fun, Types)                                 → Errors
no-name-clash(Proc, Types)                                → Errors

read-send-check(Types, Types, Types, Types)               → Errors
check-er(Types, Types, Types, Types, Types, Types)        → Errors
check-r(Types, Types, Types, Types, Types, Types)         → Errors
check-s(Types, Types, Types, Types, Types, Types)         → Errors
check-c(Types, Types, Types, Types, Types, Types)         → Errors

comm-check(Comm)                                          → Errors

er-aux-check(Signature)                                   → Errors
check-er-aux-sort(Names)                                  → Error
check-er-aux-ports(Fun)                                   → Error
check-er-aux-r(Act)                                       → Error
check-er-aux-s(Act)                                       → Error
check-er-aux-c(Act)                                       → Error

is-quote-list(Name)                                       → BOOL
```

  variables
    Chs [*0-9′*]∗ → CHAR+

    Scope [*0-9′*]∗ → Scope
equations

inspect-procspecs(sort *Ns*) = No errors were found [inspect-procspecs-1]

$\text{inspect-procspecs}(\text{func } \mathit{Fds}) = \text{No errors were found}$ [inspect-procspecs-2]
$\text{inspect-procspecs}(\mathit{Rew\text{-}spec}) = \text{No errors were found}$ [inspect-procspecs-3]
$\text{inspect-procspecs}(\text{act } \mathit{Ads}) = \text{No errors were found}$ [inspect-procspecs-4]
$\text{inspect-procspecs}(\text{comm } \mathit{Cds}) = \text{No errors were found}$ [inspect-procspecs-5]
$\text{inspect-procspecs}(\text{proc } \mathit{Pds}) = \text{inspect-headers}([], \mathit{Pds})$ [inspect-procspecs-6]

$$\text{inspect-procspecs}(\text{var } \mathit{Vds} \text{ proc } \mathit{Pds}) = \text{join}(\text{uniq-vars}([\mathit{Vds}]), \text{inspect-headers}([\mathit{Vds}], \mathit{Pds}))$$ [inspect-procspecs-7]

$$\text{inspect-procspecs}(\text{SE } \mathit{SEs}) = \text{join}(\text{inspect-procspecs}(\text{SE}), \text{inspect-procspecs}(\mathit{SEs}))$$ [inspect-procspecs-8]

$\text{inspect-headers}(\mathit{Var}, N = \mathit{Pre}) = \text{No errors were found}$ [inspect-headers-1]

$\text{inspect-headers}(\mathit{Var}, N(\mathit{Svds}) = \mathit{Pre})$ [inspect-headers-2]
$= \text{Syntax error in process header } N(\mathit{Svds}\ ). \text{ Use names instead of variable declarations}$

$\text{inspect-headers}(\mathit{Var}, N(\mathit{Cns}) = \mathit{Pre}) = \text{inspect-args}(\mathit{Var}, \mathit{Cns})$ [inspect-headers-3]

$$\text{inspect-headers}(\mathit{Var}, \mathit{Pd}\ \mathit{Pds}) = \text{join}(\text{inspect-headers}(\mathit{Var}, \mathit{Pd}), \text{inspect-headers}(\mathit{Var}, \mathit{Pds}))$$ [inspect-headers-4]

$$\frac{\text{is-variable}(\mathit{Var}, N) = \text{true}}{\text{inspect-args}(\mathit{Var}, N) = \text{No errors were found}}$$ [inspect-args-1]

$$\frac{\text{is-variable}(\mathit{Var}, N) \neq \text{true}}{\text{inspect-args}(\mathit{Var}, N) = \text{Unknown variable } N \text{ in process header}}$$ [inspect-args-2]

$$\text{inspect-args}(\mathit{Var}, N, \mathit{Cns}) = \text{join}(\text{inspect-args}(\mathit{Var}, N), \text{inspect-args}(\mathit{Var}, \mathit{Cns}))$$ [inspect-args-3]

$\text{uniq-vars}([]) = \text{No errors were found}$ [uniq-vars-1]

$$\frac{\text{is-variable}([\mathit{OVds}], N) \neq \text{true}}{\text{uniq-vars}([N : N'\ \mathit{OVds}]) = \text{uniq-vars}([\mathit{OVds}])}$$ [uniq-vars-2]

$$\frac{\text{is-variable}([\mathit{OVds}], N) = \text{true}}{\text{uniq-vars}([N : N'\ \mathit{OVds}]) = \text{join}(\text{Variable } N \text{ has been defined more than once}, \text{uniq-vars}([\mathit{OVds}]))}$$ [uniq-vars-3]

$\text{precheck-spec}(\mathit{Sig}, \text{sort } \mathit{Ns}) = \text{No errors were found}$ [precheck-spec-1]
$\text{precheck-spec}(\mathit{Sig}, \text{func } \mathit{Fds}) = \text{No errors were found}$ [precheck-spec-2]

precheck-spec(*Sig*, *Rew-spec*) = No errors were found [precheck-spec-3]
precheck-spec(*Sig*, act *Ads*) = No errors were found [precheck-spec-4]
precheck-spec(*Sig*, comm *Cds*) = No errors were found [precheck-spec-5]
precheck-spec(*Sig*, proc *Pds*) = precheck-pd(*Sig*, [], *Pds*) [precheck-spec-6]
precheck-spec(*Sig*, var *Vds* proc *Pds*) = precheck-pd(*Sig*, [*Vds*], *Pds*) [precheck-spec-7]

precheck-spec(*Sig*, SE *SEs*) = join(precheck-spec(*Sig*, SE),
precheck-spec(*Sig*, *SEs*)) [precheck-spec-8]

precheck-pd(*Sig*, *Globvar*, *N* = *Pre*) = precheck-expr(*Sig*, *Globvar*, [], *Pre*, outscope) [precheck-pd-1]

precheck-pd(*Sig*, *Globvar*, *N*(*Svds*) = *Pre*) [precheck-pd-2]
= Syntax error in process header *N*(*Svds*). Use names instead of variable declarations

precheck-pd(*Sig*, *Globvar*, *N*(*Cns*) = *Pre*) [precheck-pd-3]
= precheck-expr(*Sig*, *Globvar*, get-locals(*Globvar*, *Cns*), *Pre*, outscope)

precheck-pd(*Sig*, *Globvar*, *Pd Pds*) = join(precheck-pd(*Sig*, *Globvar*, *Pd*),
precheck-pd(*Sig*, *Globvar*, *Pds*)) [precheck-pd-4]

precheck-expr(*Sig*, *Globvar*, *Locvar*, $Pre_1 + Pre_2$, *Scope*) [precheck-expr-1]
= join(precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1, *Scope*),
precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_2, *Scope*))

precheck-expr(*Sig*, *Globvar*, *Locvar*, $Pre_1 ⌊⌊ Pre_2$, outscope) [precheck-expr-2]
= join(precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1, outscope),
precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_2, outscope))

precheck-expr(*Sig*, *Globvar*, *Locvar*, $Pre_1 ⌊⌊ Pre_2$, inscope) [precheck-expr-3]
= ACP operator '⌊⌊' used in left-hand-side of ';'. Use PA operator '|⌊⌊'

precheck-expr(*Sig*, *Globvar*, *Locvar*, $Pre_1 \| Pre_2$, outscope) [precheck-expr-4]
= join(precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1, outscope),
precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_2, outscope))

precheck-expr(*Sig*, *Globvar*, *Locvar*, $Pre_1 \| Pre_2$, inscope) [precheck-expr-5]
= ACP operator '||' used in left-hand-side of ';'. Use PA operator '|||'

precheck-expr(*Sig*, *Globvar*, *Locvar*, $Pre_1 \mid Pre_2$, outscope) [precheck-expr-6]
= join(precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1, outscope),
precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_2, outscope))

precheck-expr(*Sig*, *Globvar*, *Locvar*, $Pre_1 \mid Pre_2$, inscope) [precheck-expr-7]
= Communication used in left-hand-side of ';'

precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1 ||| Pre_2, *Scope*) [precheck-expr-8]
= join(precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1, *Scope*),
precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_2, *Scope*))

precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1 $\underline{|||}$ Pre_2, *Scope*) [precheck-expr-9]
= join(precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1, *Scope*),
precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_2, *Scope*))

precheck-expr(*Sig*, *Globvar*, *Locvar*, $Pre_1 \triangleleft Trm \triangleright Pre_2$, *Scope*) [precheck-expr-10]
= join(precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1, *Scope*),
precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_2, *Scope*))

precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1; Pre_2, *Scope*) [precheck-expr-11]
= join(precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1, inscope),
precheck-expr(*Sig*, *Globvar*,
er-vars(*Globvar*, *Locvar*, Pre_1),
Pre_2, *Scope*))

precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1 . Pre_2, *Scope*) [precheck-expr-12]
= join(precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1, *Scope*),
precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_2, *Scope*))

precheck-expr(*Sig*, *Globvar*, *Locvar*, $Pre_1 * Pre_2$, outscope) [precheck-expr-13]
= join(precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_1, outscope),
precheck-expr(*Sig*, *Globvar*, *Locvar*, Pre_2, outscope))

precheck-expr(*Sig*, *Globvar*, *Locvar*, $Pre_1 * Pre_2$, inscope) [precheck-expr-14]
= Iteration used in left-hand-side of ';'

precheck-expr(*Sig*, *Globvar*, *Locvar*, delta, *Scope*) = No errors were found [precheck-expr-15]

precheck-expr(*Sig*, *Globvar*, *Locvar*, tau, *Scope*) = No errors were found [precheck-expr-16]

precheck-expr(*Sig*, *Globvar*, *Locvar*, encap({*Cns*}, *Pre*), *Scope*) [precheck-expr-17]
= precheck-expr(*Sig*, *Globvar*, *Locvar*, *Pre*, *Scope*)

precheck-expr(*Sig*, *Globvar*, *Locvar*, hide({*Cns*}, *Pre*), *Scope*) [precheck-expr-18]
= precheck-expr(*Sig*, *Globvar*, *Locvar*, *Pre*, *Scope*)

precheck-expr(*Sig*, *Globvar*, *Locvar*, rename({*Rds*}, *Pre*), *Scope*) [precheck-expr-19]
= precheck-expr(*Sig*, *Globvar*, *Locvar*, *Pre*, *Scope*)

precheck-expr(*Sig*, *Globvar*, *Locvar*, sum(*N*, *Pre*), *Scope*) [precheck-expr-20]

$$\begin{array}{l} = \text{join(precheck-sum}(\mathit{Globvar}, \mathit{N}), \\ \quad\quad \text{precheck-expr}(\mathit{Sig}, \mathit{Globvar}, \\ \quad\quad\quad\quad\quad\quad \text{add-to-locals}(\mathit{Globvar}, \mathit{Locvar}, \mathit{N}), \\ \quad\quad\quad\quad\quad\quad \mathit{Pre}, \mathit{Scope})) \end{array}$$

$$\frac{\text{is-variable}(\mathit{Var}, \mathit{N}) \ = \ \text{true}}{\text{precheck-sum}(\mathit{Var}, \mathit{N}) \ = \ \text{No errors were found}} \quad \text{[precheck-sum-1]}$$

$$\frac{\text{is-variable}(\mathit{Var}, \mathit{N}) \ \neq \ \text{true}}{\text{precheck-sum}(\mathit{Var}, \mathit{N}) \ = \ \text{Unknown variable } \mathit{N} \text{ as argument of sum}} \quad \text{[precheck-sum-2]}$$

$$\begin{array}{l} \text{precheck-expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \text{sum}(\mathit{Svd}, \mathit{Pre}), \mathit{Scope}) \quad \text{[precheck-expr-21]} \\ = \text{Syntax error in sum construction sum(} \mathit{Svd}\text{, xxx). Use name instead of variable declaration} \end{array}$$

$$\text{precheck-expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{N}, \text{outscope}) = \text{No errors were found} \quad \text{[precheck-expr-22]}$$

$$\text{precheck-expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{N}(\mathit{Trms}), \text{outscope}) = \text{No errors were found} \quad \text{[precheck-expr-23]}$$

$$\frac{\text{is-process}(\text{sigproc}(\mathit{Sig}), \mathit{N}) \ \neq \ \text{true}}{\text{precheck-expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{N}, \text{inscope}) \ = \ \text{No errors were found}} \quad \text{[precheck-expr-24]}$$

$$\frac{\text{is-process}(\text{sigproc}(\mathit{Sig}), \mathit{N}) \ = \ \text{true}}{\begin{array}{l}\text{precheck-expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{N}, \text{inscope}) \ = \\ \text{Process } \mathit{N} \text{ called in left-hand-side of `;'}\end{array}} \quad \text{[precheck-expr-25]}$$

$$\frac{\begin{array}{r}\mathit{Xns} \ = \ \text{sort-list}(\mathit{Sig}, \mathit{Locvar}, \mathit{Trms}), \\ \text{is-process}(\text{sigproc}(\mathit{Sig}), \mathit{N}(\mathit{Xns})) \ \neq \ \text{true}\end{array}}{\text{precheck-expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{N}(\mathit{Trms}), \text{inscope}) \ = \ \text{No errors were found}} \quad \text{[precheck-expr-26]}$$

$$\frac{\begin{array}{r}\mathit{Xns} \ = \ \text{sort-list}(\mathit{Sig}, \mathit{Locvar}, \mathit{Trms}), \\ \text{is-process}(\text{sigproc}(\mathit{Sig}), \mathit{N}(\mathit{Xns})) \ = \ \text{true}\end{array}}{\begin{array}{l}\text{precheck-expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{N}(\mathit{Trms}), \text{inscope}) \ = \\ \text{Process } \mathit{N}(\mathit{Xns}) \text{ called in left-hand-side of `;'}\end{array}} \quad \text{[precheck-expr-27]}$$

$$\begin{array}{l} \text{precheck-expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \text{er}(\mathit{Trm}, \mathit{Cns}), \text{outscope}) \quad \text{[precheck-expr-28]} \\ = \text{Early read used outside scope of `;'} \end{array}$$

$$\begin{array}{l} \text{precheck-expr}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \text{er}(\mathit{Trm}, \mathit{Cns}), \text{inscope}) \quad \text{[precheck-expr-29]} \\ = \text{precheck-er}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{Trm}, \mathit{Cns}) \end{array}$$

$$\frac{\text{sort}(\mathit{Sig}, \mathit{Locvar}, \mathit{Trm}) \ \neq \ \texttt{unknown}}{\text{precheck-er}(\mathit{Sig}, \mathit{Globvar}, \mathit{Locvar}, \mathit{Trm}, \mathit{Cns}) \ = \ \text{precheck-er}(\mathit{Globvar}, \mathit{Cns})} \quad \text{[precheck-er-1]}$$

$$\frac{\mathsf{sort}(\mathit{Sig},\ \mathit{Locvar},\ \mathit{Trm})\ =\ \texttt{unknown}}{\begin{array}{l}\mathsf{precheck\text{-}er}(\mathit{Sig},\ \mathit{Globvar},\ \mathit{Locvar},\ \mathit{Trm},\ \mathit{Cns})\ = \\ \mathsf{join}(\text{Could not determine sort of data-term } \mathit{Trm} \text{ in `er'}, \\ \quad \mathsf{precheck\text{-}er}(\mathit{Globvar},\ \mathit{Cns}))\end{array}} \quad \text{[precheck-er-2]}$$

$$\frac{\mathsf{is\text{-}variable}(\mathit{Var},\ \mathit{N})\ =\ \mathsf{true}}{\mathsf{precheck\text{-}er}(\mathit{Var},\ \mathit{N})\ =\ \text{No errors were found}} \quad \text{[precheck-er-3]}$$

$$\frac{\mathsf{is\text{-}variable}(\mathit{Var},\ \mathit{N})\ \neq\ \mathsf{true}}{\mathsf{precheck\text{-}er}(\mathit{Var},\ \mathit{N})\ =\ \text{Unknown variable } \mathit{N} \text{ as argument of `er'}} \quad \text{[precheck-er-4]}$$

$$\begin{array}{l}\mathsf{precheck\text{-}er}(\mathit{Var},\ \mathit{N},\ \mathit{Cns})\ =\ \mathsf{join}(\mathsf{precheck\text{-}er}(\mathit{Var},\ \mathit{N}), \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathsf{precheck\text{-}er}(\mathit{Var},\ \mathit{Cns}))\end{array} \quad \text{[precheck-er-5]}$$

$$\begin{array}{l}\mathsf{no\text{-}name\text{-}clash}(\mathit{Sig},\ \mathit{Types})\ =\ \mathsf{join}(\mathsf{no\text{-}name\text{-}clash}(\mathsf{sigfun}(\mathit{Sig}),\ \mathit{Types}), \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathsf{no\text{-}name\text{-}clash}(\mathsf{sigproc}(\mathit{Sig}),\ \mathit{Types}))\end{array} \quad \text{[clash-1]}$$

$$\mathsf{no\text{-}name\text{-}clash}(\mathit{Fun},\ []) = \text{No errors were found} \quad \text{[clash-2]}$$

$$\frac{\mathit{Ip}\ =\ \mathsf{empty\text{-}pattern}}{\mathsf{no\text{-}name\text{-}clash}(\mathit{Fun},\ [\mathit{Ip}\ \mathit{OIps}])\ =\ \mathsf{no\text{-}name\text{-}clash}(\mathit{Fun},\ [\mathit{OIps}])} \quad \text{[clash-3]}$$

$$\frac{\begin{array}{r}\mathit{Ip}\ =\ \mathit{Xns}, \\ \mathsf{is\text{-}function}(\mathit{Fun},\ \texttt{r} : \mathit{Xns})\ \neq\ \mathsf{true}\end{array}}{\mathsf{no\text{-}name\text{-}clash}(\mathit{Fun},\ [\mathit{Ip}\ \mathit{OIps}])\ =\ \mathsf{no\text{-}name\text{-}clash}(\mathit{Fun},\ [\mathit{OIps}])} \quad \text{[clash-4]}$$

$$\frac{\begin{array}{r}\mathit{Ip}\ =\ \mathit{Xns}, \\ \mathsf{is\text{-}function}(\mathit{Fun},\ \texttt{r} : \mathit{Xns})\ =\ \mathsf{true}\end{array}}{\begin{array}{l}\mathsf{no\text{-}name\text{-}clash}(\mathit{Fun},\ [\mathit{Ip}\ \mathit{OIps}])\ = \\ \mathsf{join}(\text{Function } \texttt{r} : \mathit{Xns} \text{ has the same input-pattern as an `er'}, \\ \quad \mathsf{no\text{-}name\text{-}clash}(\mathit{Fun},\ [\mathit{OIps}]))\end{array}} \quad \text{[clash-5]}$$

$$\mathsf{no\text{-}name\text{-}clash}(\mathit{Proc},\ []) = \text{No errors were found} \quad \text{[clash-6]}$$

$$\frac{\mathit{Ip}\ =\ \mathsf{empty\text{-}pattern}}{\mathsf{no\text{-}name\text{-}clash}(\mathit{Proc},\ [\mathit{Ip}\ \mathit{OIps}])\ =\ \mathsf{no\text{-}name\text{-}clash}(\mathit{Proc},\ [\mathit{OIps}])} \quad \text{[clash-7]}$$

$$\frac{\begin{array}{r}\mathit{Ip}\ =\ \mathit{Xns}, \\ \mathsf{is\text{-}process}(\mathit{Proc},\ \texttt{r}(\mathit{Xns}))\ \neq\ \mathsf{true}\end{array}}{\mathsf{no\text{-}name\text{-}clash}(\mathit{Proc},\ [\mathit{Ip}\ \mathit{OIps}])\ =\ \mathsf{no\text{-}name\text{-}clash}(\mathit{Proc},\ [\mathit{OIps}])} \quad \text{[clash-8]}$$

$$\frac{\mathit{Ip} \;=\; \mathit{Xns}, \quad \text{is-process}(\mathit{Proc}, \mathtt{r}(\mathit{Xns})) \;=\; \text{true}}{\begin{array}{l}\text{no-name-clash}(\mathit{Proc}, [\mathit{Ip}\ \mathit{OIps}]) \;= \\ \text{join}(\text{Process } \mathtt{r}(\mathit{Xns}) \text{ has the same input-pattern as an `er' }, \\ \quad \text{no-name-clash}(\mathit{Proc}, [\mathit{OIps}]))\end{array}} \qquad \text{[clash-9]}$$

$$\begin{array}{l}\text{read-send-check}(\mathit{Types}_1, \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4) \\ = \text{check-er}(\mathit{Types}_1, \mathit{Types}_2 - \mathit{Types}_1, \\ \qquad \mathit{Types}_3, \mathit{Types}_1 + \mathit{Types}_2, \mathit{Types}_1 + \mathit{Types}_2 + \mathit{Types}_3, \mathit{Types}_4)\end{array} \qquad \text{[rs-check-1]}$$

$$\begin{array}{l}\text{check-er}([], \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6) \\ = \text{check-r}([], \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6)\end{array} \qquad \text{[check-er-1]}$$

$$\frac{\mathit{Ip} \text{ in } \mathit{Types}_3 \;=\; \text{true}}{\begin{array}{l}\text{check-er}([\mathit{Ip}\ \mathit{OIps}], \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6) \;= \\ \text{check-er}([\mathit{OIps}], \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6)\end{array}} \qquad \text{[check-er-2]}$$

$$\frac{\mathit{Ip} \text{ in } \mathit{Types}_3 \;\neq\; \text{true}, \quad \mathit{Ip} \;=\; \text{empty-pattern}}{\begin{array}{l}\text{check-er}([\mathit{Ip}\ \mathit{OIps}], \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6) \;= \\ \text{join}(\text{Action s should be declared to match `er'}, \\ \quad \text{check-er}([\mathit{OIps}], \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6))\end{array}} \qquad \text{[check-er-3]}$$

$$\frac{\mathit{Ip} \text{ in } \mathit{Types}_3 \;\neq\; \text{true}, \quad \mathit{Ip} \;=\; \mathit{Xns}}{\begin{array}{l}\text{check-er}([\mathit{Ip}\ \mathit{OIps}], \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6) \;= \\ \text{join}(\text{Action s: } \mathit{Xns} \text{ should be declared to match `er'}, \\ \quad \text{check-er}([\mathit{OIps}], \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6))\end{array}} \qquad \text{[check-er-4]}$$

$$\begin{array}{l}\text{check-r}(\mathit{Types}_1, [], \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6) \\ = \text{check-s}(\mathit{Types}_1, [], \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6)\end{array} \qquad \text{[check-r-1]}$$

$$\frac{\mathit{Ip} \text{ in } \mathit{Types}_3 \;=\; \text{true}}{\begin{array}{l}\text{check-r}(\mathit{Types}_1, [\mathit{Ip}\ \mathit{OIps}], \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6) \;= \\ \text{check-r}(\mathit{Types}_1, [\mathit{OIps}], \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6)\end{array}} \qquad \text{[check-r-2]}$$

$$\frac{\mathit{Ip} \text{ in } \mathit{Types}_3 \;\neq\; \text{true}, \quad \mathit{Ip} \;=\; \text{empty-pattern}}{\begin{array}{l}\text{check-r}(\mathit{Types}_1, [\mathit{Ip}\ \mathit{OIps}], \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6) \;= \\ \text{join}(\text{Action s should be declared to match `r'}, \\ \quad \text{check-r}(\mathit{Types}_1, [\mathit{OIps}], \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6))\end{array}} \qquad \text{[check-r-3]}$$

$$\frac{\mathit{Ip} \text{ in } \mathit{Types}_3 \;\neq\; \text{true}, \quad \mathit{Ip} \;=\; \mathit{Xns}}{\begin{array}{l}\text{check-r}(\mathit{Types}_1, [\mathit{Ip}\ \mathit{OIps}], \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6) \;= \\ \text{join}(\text{Action s: } \mathit{Xns} \text{ should be declared to match `r'}, \\ \quad \text{check-r}(\mathit{Types}_1, [\mathit{OIps}], \mathit{Types}_3, \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6))\end{array}} \qquad \text{[check-r-4]}$$

$$\text{check-s}(\mathit{Types}_1, \mathit{Types}_2, [\,], \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6) = \text{check-c}(\mathit{Types}_1, \mathit{Types}_2, [\,], \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6)$$ [check-s-1]

$$\frac{\mathit{Ip} \text{ in } \mathit{Types}_4 = \text{true}}{\text{check-s}(\mathit{Types}_1, \mathit{Types}_2, [\mathit{Ip}\ \mathit{OIps}], \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6) = \text{check-s}(\mathit{Types}_1, \mathit{Types}_2, [\mathit{OIps}], \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6)}$$ [check-s-2]

$$\frac{\mathit{Ip} \text{ in } \mathit{Types}_4 \neq \text{true},\quad \mathit{Ip} = \text{empty-pattern}}{\text{check-s}(\mathit{Types}_1, \mathit{Types}_2, [\mathit{Ip}\ \mathit{OIps}], \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6) = \text{join}(\text{Action r should be declared to match `s'}, \text{check-s}(\mathit{Types}_1, \mathit{Types}_2, [\mathit{OIps}], \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6))}$$ [check-s-3]

$$\frac{\mathit{Ip} \text{ in } \mathit{Types}_4 \neq \text{true},\quad \mathit{Ip} = \mathit{Xns}}{\text{check-s}(\mathit{Types}_1, \mathit{Types}_2, [\mathit{Ip}\ \mathit{OIps}], \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6) = \text{join}(\text{Action r: } \mathit{Xns} \text{ should be declared to match `s'}, \text{check-s}(\mathit{Types}_1, \mathit{Types}_2, [\mathit{OIps}], \mathit{Types}_4, \mathit{Types}_5, \mathit{Types}_6))}$$ [check-s-4]

$$\text{check-c}(\mathit{Types}_1, \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, [\,], \mathit{Types}_6) = \text{No errors were found}$$ [check-c-1]

$$\frac{\mathit{Ip} \text{ in } \mathit{Types}_6 = \text{true}}{\text{check-c}(\mathit{Types}_1, \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, [\mathit{Ip}\ \mathit{OIps}], \mathit{Types}_6) = \text{check-c}(\mathit{Types}_1, \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, [\mathit{OIps}], \mathit{Types}_6)}$$ [check-c-2]

$$\frac{\mathit{Ip} \text{ in } \mathit{Types}_6 \neq \text{true},\quad \mathit{Ip} = \text{empty-pattern}}{\text{check-c}(\mathit{Types}_1, \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, [\mathit{Ip}\ \mathit{OIps}], \mathit{Types}_6) = \text{join}(\text{Action c should be declared to match `r' and `s'}, \text{check-c}(\mathit{Types}_1, \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, [\mathit{OIps}], \mathit{Types}_6))}$$ [check-c-3]

$$\frac{\mathit{Ip} \text{ in } \mathit{Types}_6 \neq \text{true},\quad \mathit{Ip} = \mathit{Xns}}{\text{check-c}(\mathit{Types}_1, \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, [\mathit{Ip}\ \mathit{OIps}], \mathit{Types}_6) = \text{join}(\text{Action c: } \mathit{Xns} \text{ should be declared to match `r' and `s'}, \text{check-c}(\mathit{Types}_1, \mathit{Types}_2, \mathit{Types}_3, \mathit{Types}_4, [\mathit{OIps}], \mathit{Types}_6))}$$ [check-c-4]

$$\text{comm-check}([\,]) = \text{No errors were found}$$ [comm-check-1]

$$\frac{N \neq \text{c}}{\text{comm-check}([\mathtt{r} \mid \text{s} = N\ \mathit{OCds}]) = \text{join}(\text{Result } N \text{ in read-send communication r} \mid \text{s} = N \text{ should be `c'}, \text{comm-check}([\mathit{OCds}]))}$$ [comm-check-2]

$\mathsf{comm\text{-}check}([\mathit{Cd}\ \mathit{OCds}]) = \mathsf{comm\text{-}check}([\mathit{OCds}])$ **otherwise** [comm-check-3]

$$\begin{aligned}&\mathsf{er\text{-}aux\text{-}check}(\mathit{Sig})\\&= \mathsf{join}(\mathsf{check\text{-}er\text{-}aux\text{-}sort}(\mathsf{sigsort}(\mathit{Sig})),\\&\qquad \mathsf{join}(\mathsf{check\text{-}er\text{-}aux\text{-}ports}(\mathsf{sigfun}(\mathit{Sig})),\\&\qquad\quad \mathsf{join}(\mathsf{check\text{-}er\text{-}aux\text{-}r}(\mathsf{sigact}(\mathit{Sig})),\\&\qquad\qquad \mathsf{join}(\mathsf{check\text{-}er\text{-}aux\text{-}s}(\mathsf{sigact}(\mathit{Sig})),\\&\qquad\qquad\quad \mathsf{check\text{-}er\text{-}aux\text{-}c}(\mathsf{sigact}(\mathit{Sig}))))))\end{aligned}$$ [er-aux-check-1]

$$\frac{\texttt{Er-aux-port}\ \mathsf{in}\ \mathit{Names} \neq \mathsf{true}}{\mathsf{check\text{-}er\text{-}aux\text{-}sort}(\mathit{Names}) = \text{No errors were found}}$$ [check-er-aux-sort-1]

$$\frac{\texttt{Er-aux-port}\ \mathsf{in}\ \mathit{Names} = \mathsf{true}}{\mathsf{check\text{-}er\text{-}aux\text{-}sort}(\mathit{Names}) = \text{Special sort name `Er-aux-port' used}}$$ [check-er-aux-sort-2]

$$\begin{aligned}&\mathsf{check\text{-}er\text{-}aux\text{-}ports}([\mathit{OFds}_1\ \texttt{er-aux-port} : \ \to N\ \mathit{OFds}_2])\\&= \text{Special constant starting with `er-aux-port' used}\end{aligned}$$ [check-er-aux-ports-1]

$$\frac{\mathsf{is\text{-}quote\text{-}list}(\mathsf{name}(\mathit{Chs})) = \mathsf{true}}{\begin{array}{l}\mathsf{check\text{-}er\text{-}aux\text{-}ports}([\mathit{OFds}_1\\ \qquad \mathsf{name}(\texttt{"e" "r" "-" "a" "u" "x" "-" "p" "o" "r" "t"}\ \mathit{Chs}) : \ \to N\\ \qquad \mathit{OFds}_2]) =\\ \text{Special constant starting with `er-aux-port' used}\end{array}}$$ [check-er-aux-ports-2]

$\mathsf{check\text{-}er\text{-}aux\text{-}ports}(\mathit{Fun}) = \text{No errors were found}$ **otherwise** [check-er-aux-ports-3]

$$\frac{\mathsf{types}(\mathit{Act}, \texttt{er-aux-r}) = []}{\mathsf{check\text{-}er\text{-}aux\text{-}r}(\mathit{Act}) = \text{No errors were found}}$$ [check-er-aux-r-1]

$$\frac{\mathsf{types}(\mathit{Act}, \texttt{er-aux-r}) \neq []}{\mathsf{check\text{-}er\text{-}aux\text{-}r}(\mathit{Act}) = \text{Special action name `er-aux-r' used}}$$ [check-er-aux-r-2]

$$\frac{\mathsf{types}(\mathit{Act}, \texttt{er-aux-s}) = []}{\mathsf{check\text{-}er\text{-}aux\text{-}s}(\mathit{Act}) = \text{No errors were found}}$$ [check-er-aux-s-1]

$$\frac{\mathsf{types}(\mathit{Act}, \texttt{er-aux-s}) \neq []}{\mathsf{check\text{-}er\text{-}aux\text{-}s}(\mathit{Act}) = \text{Special action name `er-aux-s' used}}$$ [check-er-aux-s-2]

$$\frac{\mathsf{types}(\mathit{Act}, \texttt{er-aux-c}) = []}{\mathsf{check\text{-}er\text{-}aux\text{-}c}(\mathit{Act}) = \text{No errors were found}}$$ [check-er-aux-c-1]

$$\frac{\mathsf{types}(\mathit{Act}, \texttt{er-aux-c}) \neq []}{\mathsf{check\text{-}er\text{-}aux\text{-}c}(\mathit{Act}) = \text{Special action name `er-aux-c' used}}$$ [check-er-aux-c-2]

$$\text{is-quote-list}(\text{name}(\texttt{"'"})) = \text{true} \qquad \text{[is-quote-list-1]}$$

$$\text{is-quote-list}(\text{name}(\texttt{"'"}\ \mathit{Chs})) = \text{is-quote-list}(\text{name}(\mathit{Chs})) \qquad \text{[is-quote-list-2]}$$

$$\frac{\text{inspect-procspecs}(E) \ = \ \text{No errors were found}}{\text{precheck}(E) \ = \ \text{precheck}(\text{sig-gpsl}(E),\ E)} \qquad \text{[precheck-1]}$$

$$\frac{\begin{array}{rcl}\text{inspect-procspecs}(E) & = & \mathit{Errs}, \\ \mathit{Errs} & \neq & \text{No errors were found}\end{array}}{\text{precheck}(E) \ = \ \mathit{Errs}} \qquad \text{[precheck-2]}$$

$$\frac{\begin{array}{rcl}\text{precheck-spec}(\mathit{Sig},\ E) & = & \text{No errors were found}, \\ \mathit{Act} & = & \text{sigact}(\mathit{Sig}), \\ \mathit{Types} & = & \text{er-types}(\mathit{Sig},\ E)\end{array}}{\begin{array}{l}\text{precheck}(\mathit{Sig},\ E) \ = \\ \text{join}(\text{no-name-clash}(\mathit{Sig},\ \mathit{Types}), \\ \quad \text{join}(\text{read-send-check}(\mathit{Types},\ \text{types}(\mathit{Act},\ \texttt{r}),\ \text{types}(\mathit{Act},\ \texttt{s}),\ \text{types}(\mathit{Act},\ \texttt{c})), \\ \qquad \text{join}(\text{comm-check}(\text{sigcomm}(\mathit{Sig})), \\ \qquad\quad \text{er-aux-check}(\mathit{Sig}))))\end{array}} \qquad \text{[precheck-3]}$$

$$\frac{\begin{array}{rcl}\text{precheck-spec}(\mathit{Sig},\ E) & = & \mathit{Errs}, \\ \mathit{Errs} & \neq & \text{No errors were found}\end{array}}{\text{precheck}(\mathit{Sig},\ E) \ = \ \mathit{Errs}} \qquad \text{[precheck-4]}$$

A.8 Substitution

imports GPSL$^{A.3}$
exports
  sorts Data-terms
  context-free syntax
    {Data-term ","}+ → Data-terms

    subst(Data-term, Name, Process-expression) → Process-expression
    subst-trms(Data-term, Name, {Data-term ","}+) → Data-terms
equations

$$\mathrm{subst}(Trm, N, Pre_1 + Pre_2) = \mathrm{subst}(Trm, N, Pre_1) + \mathrm{subst}(Trm, N, Pre_2) \qquad \text{[subst-1]}$$

$$\mathrm{subst}(Trm, N, Pre_1 \mathbin{\|\!\lfloor} Pre_2) = \mathrm{subst}(Trm, N, Pre_1) \mathbin{\|\!\lfloor} \mathrm{subst}(Trm, N, Pre_2) \qquad \text{[subst-2]}$$

$$\mathrm{subst}(Trm, N, Pre_1 \parallel Pre_2) = \mathrm{subst}(Trm, N, Pre_1) \parallel \mathrm{subst}(Trm, N, Pre_2) \qquad \text{[subst-3]}$$

$$\mathrm{subst}(Trm, N, Pre_1 \mid Pre_2) = \mathrm{subst}(Trm, N, Pre_1) \mid \mathrm{subst}(Trm, N, Pre_2) \qquad \text{[subst-4]}$$

$$\mathrm{subst}(Trm, N, Pre_1 \mathbin{|||} Pre_2) = \mathrm{subst}(Trm, N, Pre_1) \mathbin{|||} \mathrm{subst}(Trm, N, Pre_2) \qquad \text{[subst-5]}$$

$$\mathrm{subst}(Trm, N, Pre_1 \mathbin{|\|\!\lfloor} Pre_2) = \mathrm{subst}(Trm, N, Pre_1) \mathbin{|\|\!\lfloor} \mathrm{subst}(Trm, N, Pre_2) \qquad \text{[subst-6]}$$

$$\mathrm{subst}(Trm, N, Pre_1 \triangleleft Trm \triangleright Pre_2) = \mathrm{subst}(Trm, N, Pre_1) \triangleleft Trm \triangleright \mathrm{subst}(Trm, N, Pre_2) \qquad \text{[subst-7]}$$

$$\mathrm{subst}(Trm, N, Pre_1; Pre_2) = \mathrm{subst}(Trm, N, Pre_1); \mathrm{subst}(Trm, N, Pre_2) \qquad \text{[subst-8]}$$

$$\mathrm{subst}(Trm, N, Pre_1 \,.\, Pre_2) = \mathrm{subst}(Trm, N, Pre_1) \,.\, \mathrm{subst}(Trm, N, Pre_2) \qquad \text{[subst-9]}$$

$$\mathrm{subst}(Trm, N, Pre_1 * Pre_2) = \mathrm{subst}(Trm, N, Pre_1) * \mathrm{subst}(Trm, N, Pre_2) \qquad \text{[subst-10]}$$

$$\mathrm{subst}(Trm, N, \mathrm{delta}) = \mathrm{delta} \qquad \text{[subst-11]}$$

$$\mathrm{subst}(Trm, N, \mathrm{tau}) = \mathrm{tau} \qquad \text{[subst-12]}$$

$$\mathrm{subst}(Trm, N, \mathrm{encap}(\{Cns\}, Pre)) = \mathrm{encap}(\{Cns\}, \mathrm{subst}(Trm, N, Pre)) \qquad \text{[subst-13]}$$

$$\mathrm{subst}(Trm, N, \mathrm{hide}(\{Cns\}, Pre)) = \mathrm{hide}(\{Cns\}, \mathrm{subst}(Trm, N, Pre)) \qquad \text{[subst-14]}$$

$$\mathrm{subst}(Trm, N, \mathrm{rename}(\{Rds\}, Pre)) = \mathrm{rename}(\{Rds\}, \mathrm{subst}(Trm, N, Pre)) \qquad \text{[subst-15]}$$

$$\mathsf{subst}(\mathit{Trm}, N, \mathsf{sum}(N, \mathit{Pre})) = \mathsf{sum}(N, \mathit{Pre}) \qquad \text{[subst-16]}$$

$$\frac{N \neq N'}{\mathsf{subst}(\mathit{Trm}, N, \mathsf{sum}(N', \mathit{Pre})) = \mathsf{sum}(N', \mathsf{subst}(\mathit{Trm}, N, \mathit{Pre}))} \qquad \text{[subst-17]}$$

$$\mathsf{subst}(\mathit{Trm}, N, N') = N' \qquad \text{[subst-18]}$$

$$\frac{\mathit{Trms}' = \mathsf{subst\text{-}trms}(\mathit{Trm}, N, \mathit{Trms})}{\mathsf{subst}(\mathit{Trm}, N, N'(\mathit{Trms})) = N'(\mathit{Trms}')} \qquad \text{[subst-19]}$$

$$\frac{\mathit{Trm}'' = \mathsf{subst\text{-}trms}(\mathit{Trm}, N, \mathit{Trm}')}{\mathsf{subst}(\mathit{Trm}, N, \mathsf{er}(\mathit{Trm}', \mathit{Cns})) = \mathsf{er}(\mathit{Trm}'', \mathit{Cns})} \qquad \text{[subst-20]}$$

$$\mathsf{subst\text{-}trms}(\mathit{Trm}, N, N) = \mathit{Trm} \qquad \text{[subst-trms-1]}$$

$$\frac{N \neq N'}{\mathsf{subst\text{-}trms}(\mathit{Trm}, N, N') = N'} \qquad \text{[subst-trms-2]}$$

$$\frac{\mathit{Trms}' = \mathsf{subst\text{-}trms}(\mathit{Trm}, N, \mathit{Trms})}{\mathsf{subst\text{-}trms}(\mathit{Trm}, N, N'(\mathit{Trms})) = N'(\mathit{Trms}')} \qquad \text{[subst-trms-3]}$$

$$\frac{\begin{array}{c}\mathit{Trm}'' = \mathsf{subst\text{-}trms}(\mathit{Trm}, N, \mathit{Trm}'), \\ \mathit{Trms}' = \mathsf{subst\text{-}trms}(\mathit{Trm}, N, \mathit{Trms})\end{array}}{\mathsf{subst\text{-}trms}(\mathit{Trm}, N, \mathit{Trm}', \mathit{Trms}) = \mathit{Trm}'', \mathit{Trms}'} \qquad \text{[subst-trms-4]}$$

A.9 ToBPA

imports Substitution$^{A.8}$ GPSL-aux$^{A.5}$
exports
  sorts BPA-pair
  context-free syntax
    "<" Process-expression "," Var ">" → BPA-pair

    bpa(Var, Var, Process-expression, Var) → BPA-pair
    normalize(Var, Var, Process-expression, Var) → BPA-pair

    put-sums({Name ","}+, Data-term, {Name ","}+, Process-expression, Name) → Process-expression

    transform({Name ","}+) → Data-terms
  variables
    Newvar [0-9′]∗ → Var
hiddens
  context-free syntax
    new-varname(Var, Name) → Name

    atomic(Process-expression) → BOOL

    is-bpa(Process-expression) → BOOL
  variables
    Chs [0-9′]∗ → CHAR+
equations

$$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{delta}, \mathit{Newvar}) = <\mathsf{delta}, \mathit{Newvar}> \quad \text{[bpa-1]}$$
$$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{tau}, \mathit{Newvar}) = <\mathsf{tau}, \mathit{Newvar}> \quad \text{[bpa-2]}$$

$$\frac{\mathsf{atomic}(\mathit{Pre}) = \mathsf{true}}{\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}, \mathit{Newvar}) = <\mathit{Pre}, \mathit{Newvar}>} \quad \text{[bpa-3]}$$

$$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{er}(\mathit{Trm}, \mathit{Cns}), \mathit{Newvar}) = <\mathsf{er}(\mathit{Trm}, \mathit{Cns}), \mathit{Newvar}> \quad \text{[bpa-4]}$$
$$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 + \mathit{Pre}_2, \mathit{Newvar}) = <\mathit{Pre}_1 + \mathit{Pre}_2, \mathit{Newvar}> \quad \text{[bpa-5]}$$
$$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \,.\, \mathit{Pre}_2, \mathit{Newvar}) = <\mathit{Pre}_1 \,.\, \mathit{Pre}_2, \mathit{Newvar}> \quad \text{[bpa-6]}$$
$$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{sum}(\mathit{N}, \mathit{Pre}), \mathit{Newvar}) = <\mathsf{sum}(\mathit{N}, \mathit{Pre}), \mathit{Newvar}> \quad \text{[bpa-7]}$$
$$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \triangleleft \mathit{Trm} \triangleright \mathit{Pre}_2, \mathit{Newvar}) = <\mathit{Pre}_1 \triangleleft \mathit{Trm} \triangleright \mathit{Pre}_2, \mathit{Newvar}> \quad \text{[bpa-8]}$$

$$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{encap}(\{\mathit{Cns}\}, \mathsf{delta}), \mathit{Newvar}) = <\mathsf{delta}, \mathit{Newvar}> \quad \text{[bpa-9]}$$
$$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{encap}(\{\mathit{Cns}\}, \mathsf{tau}), \mathit{Newvar}) = <\mathsf{tau}, \mathit{Newvar}> \quad \text{[bpa-10]}$$

$$\frac{\mathsf{atomic}(\mathit{Pre}) = \mathsf{true}}{\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{encap}(\{\mathit{Cns}\}, \mathit{Pre}), \mathit{Newvar}) = <\mathsf{encap}(\{\mathit{Cns}\}, \mathit{Pre}), \mathit{Newvar}>} \quad \text{[bpa-11]}$$

$$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{encap}(\{\mathit{Cns}\}, \mathsf{er}(\mathit{Trm}, \mathit{Cns})), \mathit{Newvar}) = <\mathsf{er}(\mathit{Trm}, \mathit{Cns}), \mathit{Newvar}> \quad \text{[bpa-12]}$$

$$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{encap}(\{\mathit{Cns}\}, \mathit{Pre}_1 + \mathit{Pre}_2), \mathit{Newvar}) \quad \text{[bpa-13]}$$

$= \ <\mathsf{encap}(\{Cns\},\ Pre_1) + \mathsf{encap}(\{Cns\},\ Pre_2),\ Newvar>$

$\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{encap}(\{Cns\},\ Pre_1\ .\ Pre_2),\ Newvar)$ [bpa-14]
$= \ <\mathsf{encap}(\{Cns\},\ Pre_1)\ .\ \mathsf{encap}(\{Cns\},\ Pre_2),\ Newvar>$

$\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{encap}(\{Cns\},\ \mathsf{sum}(N,\ Pre)),\ Newvar)$ [bpa-15]
$= \ <\mathsf{sum}(N,\ \mathsf{encap}(\{Cns\},\ Pre)),\ Newvar>$

$\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{encap}(\{Cns\},\ Pre_1 \triangleleft Trm \triangleright Pre_2),\ Newvar)$ [bpa-16]
$= \ <\mathsf{encap}(\{Cns\},\ Pre_1) \triangleleft Trm \triangleright \mathsf{encap}(\{Cns\},\ Pre_2),\ Newvar>$

$$\frac{\mathsf{bpa}(Globvar,\ Locvar,\ Pre,\ Newvar)\ =\ <Pre',\ Newvar'>}{\begin{array}{l}\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{encap}(\{Cns\},\ Pre),\ Newvar)\ = \\ <\mathsf{encap}(\{Cns\},\ Pre'),\ Newvar'>\end{array}} \quad \textbf{otherwise} \qquad \text{[bpa-17]}$$

$\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{hide}(\{Cns\},\ \mathsf{delta}),\ Newvar) = <\mathsf{delta},\ Newvar>$ [bpa-18]
$\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{hide}(\{Cns\},\ \mathsf{tau}),\ Newvar) \ = <\mathsf{tau},\ Newvar>$ [bpa-19]

$$\frac{\mathsf{atomic}(Pre)\ =\ \mathsf{true}}{\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{hide}(\{Cns\},\ Pre),\ Newvar)\ =\ <\mathsf{hide}(\{Cns\},\ Pre),\ Newvar>} \qquad \text{[bpa-20]}$$

$\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{hide}(\{Cns\},\ \mathsf{er}(Trm,\ Cns)),\ Newvar) = <\mathsf{er}(Trm,\ Cns),\ Newvar>$ [bpa-21]

$\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{hide}(\{Cns\},\ Pre_1 + Pre_2),\ Newvar)$ [bpa-22]
$= \ <\mathsf{hide}(\{Cns\},\ Pre_1) + \mathsf{hide}(\{Cns\},\ Pre_2),\ Newvar>$

$\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{hide}(\{Cns\},\ Pre_1\ .\ Pre_2),\ Newvar)$ [bpa-23]
$= \ <\mathsf{hide}(\{Cns\},\ Pre_1)\ .\ \mathsf{hide}(\{Cns\},\ Pre_2),\ Newvar>$

$\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{hide}(\{Cns\},\ \mathsf{sum}(N,\ Pre)),\ Newvar)$ [bpa-24]
$= \ <\mathsf{sum}(N,\ \mathsf{hide}(\{Cns\},\ Pre)),\ Newvar>$

$\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{hide}(\{Cns\},\ Pre_1 \triangleleft Trm \triangleright Pre_2),\ Newvar)$ [bpa-25]
$= \ <\mathsf{hide}(\{Cns\},\ Pre_1) \triangleleft Trm \triangleright \mathsf{hide}(\{Cns\},\ Pre_2),\ Newvar>$

$$\frac{\mathsf{bpa}(Globvar,\ Locvar,\ Pre,\ Newvar)\ =\ <Pre',\ Newvar'>}{\begin{array}{l}\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{hide}(\{Cns\},\ Pre),\ Newvar)\ = \\ <\mathsf{hide}(\{Cns\},\ Pre'),\ Newvar'>\end{array}} \quad \textbf{otherwise} \qquad \text{[bpa-26]}$$

$\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{rename}(\{Rds\},\ \mathsf{delta}),\ Newvar) = <\mathsf{delta},\ Newvar>$ [bpa-27]
$\mathsf{bpa}(Globvar,\ Locvar,\ \mathsf{rename}(\{Rds\},\ \mathsf{tau}),\ Newvar) \ = <\mathsf{tau},\ Newvar>$ [bpa-28]

$$\frac{\mathrm{atomic}(\mathit{Pre}) \;=\; \mathrm{true}}{\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}), \mathit{Newvar}) \;=\; < \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}), \mathit{Newvar} >} \qquad \text{[bpa-29]}$$

$$\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{rename}(\{\mathit{Rds}\}, \mathrm{er}(\mathit{Trm}, \mathit{Cns})), \mathit{Newvar}) = < \mathrm{er}(\mathit{Trm}, \mathit{Cns}), \mathit{Newvar} > \qquad \text{[bpa-30]}$$

$$\begin{aligned} &\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}_1 + \mathit{Pre}_2), \mathit{Newvar}) \qquad \text{[bpa-31]}\\ &= \; < \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}_1) + \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}_2), \mathit{Newvar} > \end{aligned}$$

$$\begin{aligned} &\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}_1 \,.\, \mathit{Pre}_2), \mathit{Newvar}) \qquad \text{[bpa-32]}\\ &= \; < \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}_1) \,.\, \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}_2), \mathit{Newvar} > \end{aligned}$$

$$\begin{aligned} &\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{rename}(\{\mathit{Rds}\}, \mathrm{sum}(N, \mathit{Pre})), \mathit{Newvar}) \qquad \text{[bpa-33]}\\ &= \; < \mathrm{sum}(N, \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre})), \mathit{Newvar} > \end{aligned}$$

$$\begin{aligned} &\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}_1 \triangleleft \mathit{Trm} \triangleright \mathit{Pre}_2), \mathit{Newvar}) \qquad \text{[bpa-34]}\\ &= \; < \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}_1) \triangleleft \mathit{Trm} \triangleright \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}_2), \mathit{Newvar} > \end{aligned}$$

$$\frac{\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}, \mathit{Newvar}) \;=\; < \mathit{Pre}', \mathit{Newvar}' >}{\begin{aligned}&\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}), \mathit{Newvar}) \;=\\ &< \mathrm{rename}(\{\mathit{Rds}\}, \mathit{Pre}'), \mathit{Newvar}' >\end{aligned}} \quad \textbf{otherwise} \qquad \text{[bpa-35]}$$

$$\begin{aligned} &\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \mathbin{|||} \mathit{Pre}_2, \mathit{Newvar}) \qquad \text{[bpa-36]}\\ &= \; < \mathit{Pre}_1 \mathbin{\parallel\!\lfloor} \mathit{Pre}_2 + \mathit{Pre}_2 \mathbin{\parallel\!\lfloor} \mathit{Pre}_1, \mathit{Newvar} > \end{aligned}$$

$$\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{delta} \mathbin{\parallel\!\lfloor} \mathit{Pre}, \mathit{Newvar}) = < \mathrm{delta}, \mathit{Newvar} > \qquad \text{[bpa-37]}$$

$$\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{tau} \mathbin{\parallel\!\lfloor} \mathit{Pre}, \mathit{Newvar}) \;\; = < \mathrm{tau} \,.\, \mathit{Pre}, \mathit{Newvar} > \qquad \text{[bpa-38]}$$

$$\frac{\mathrm{atomic}(\mathit{Pre}_1) \;=\; \mathrm{true}}{\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \mathbin{\parallel\!\lfloor} \mathit{Pre}_2, \mathit{Newvar}) \;=\; < \mathit{Pre}_1 \,.\, \mathit{Pre}_2, \mathit{Newvar} >} \qquad \text{[bpa-39]}$$

$$\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{er}(\mathit{Trm}, \mathit{Cns}) \mathbin{\parallel\!\lfloor} \mathit{Pre}, \mathit{Newvar}) = < \mathrm{er}(\mathit{Trm}, \mathit{Cns}) \,.\, \mathit{Pre}, \mathit{Newvar} > \qquad \text{[bpa-40]}$$

$$\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{delta} \,.\, \mathit{Pre}_1 \mathbin{\parallel\!\lfloor} \mathit{Pre}_2, \mathit{Newvar}) = < \mathrm{delta}, \mathit{Newvar} > \qquad \text{[bpa-41]}$$

$$\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{tau} \,.\, \mathit{Pre}_1 \mathbin{\parallel\!\lfloor} \mathit{Pre}_2, \mathit{Newvar}) = < \mathrm{tau} \,.\, (\mathit{Pre}_1 \mathbin{|||} \mathit{Pre}_2), \mathit{Newvar} > \qquad \text{[bpa-42]}$$

$$\frac{\mathrm{atomic}(\mathit{Pre}_1) \;=\; \mathrm{true}}{\begin{aligned}&\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \,.\, \mathit{Pre}_2 \mathbin{\parallel\!\lfloor} \mathit{Pre}_3, \mathit{Newvar}) \;=\\ &< \mathit{Pre}_1 \,.\, (\mathit{Pre}_2 \mathbin{|||} \mathit{Pre}_3), \mathit{Newvar} >\end{aligned}} \qquad \text{[bpa-43]}$$

$$\mathrm{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathrm{er}(\mathit{Trm}, \mathit{Cns}) \,.\, \mathit{Pre}_1 \mathbin{\parallel\!\lfloor} \mathit{Pre}_2, \mathit{Newvar}) \qquad \text{[bpa-44]}$$

$$= \ <\mathsf{er}(\mathit{Trm}, \mathit{Cns}) \ . \ (\mathit{Pre}_1 \ \mathbin{|||} \ \mathit{Pre}_2), \mathit{Newvar}>$$

$$\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, (\mathit{Pre}_1 + \mathit{Pre}_2) \ . \ \mathit{Pre}_3 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_4, \mathit{Newvar}) \\ = \ <\mathit{Pre}_1 \ . \ \mathit{Pre}_3 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_4 + \mathit{Pre}_2 \ . \ \mathit{Pre}_3 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_4, \mathit{Newvar}>\end{array} \qquad \text{[bpa-45]}$$

$$\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \ . \ \mathit{Pre}_2 \ . \ \mathit{Pre}_3 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_4, \mathit{Newvar}) \\ = \ <\mathit{Pre}_1 \ . \ (\mathit{Pre}_2 \ . \ \mathit{Pre}_3) \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_4, \mathit{Newvar}>\end{array} \qquad \text{[bpa-46]}$$

$$\frac{\mathsf{is\text{-}variable}(\mathit{Locvar}, N) \ \neq \ \mathsf{true}}{\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{sum}(N, \mathit{Pre}_1) \ . \ \mathit{Pre}_2 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_3, \mathit{Newvar}) \ = \\ <\mathsf{sum}(N, \mathit{Pre}_1 \ . \ \mathit{Pre}_2 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_3), \mathit{Newvar}>\end{array}} \qquad \text{[bpa-47]}$$

$$\frac{\begin{array}{rcl}\mathsf{is\text{-}variable}(\mathit{Locvar}, N) & = & \mathsf{true}, \\ \mathsf{make\text{-}svds}(\mathit{Globvar} + \mathit{Newvar}, N) & = & N : N', \\ \mathsf{new\text{-}varname}(\mathit{Globvar} + \mathit{Locvar} + \mathit{Newvar}, N) & = & N'', \\ \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1, \mathit{Newvar}) & = & <\mathit{Pre}', \mathit{Newvar}'>\end{array}}{\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{sum}(N, \mathit{Pre}_1) \ . \ \mathit{Pre}_2 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_3, \mathit{Newvar}) \ = \\ <\mathsf{sum}(N'', \mathsf{subst}(N'', N, \mathit{Pre}') \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_2), \mathit{Newvar}' + [N'' : N']>\end{array}} \qquad \text{[bpa-48]}$$

$$\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, (\mathit{Pre}_1 \triangleleft \mathit{Trm} \triangleright \mathit{Pre}_2) \ . \ \mathit{Pre}_3 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_4, \mathit{Newvar}) \\ = \ <(\mathit{Pre}_1 \ . \ \mathit{Pre}_3 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_4) \triangleleft \mathit{Trm} \triangleright (\mathit{Pre}_2 \ . \ \mathit{Pre}_3 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_4), \mathit{Newvar}>\end{array} \qquad \text{[bpa-49]}$$

$$\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, (\mathit{Pre}_1 + \mathit{Pre}_2) \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_3, \mathit{Newvar}) \\ = \ <\mathit{Pre}_1 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_3 + \mathit{Pre}_2 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_3, \mathit{Newvar}>\end{array} \qquad \text{[bpa-50]}$$

$$\frac{\mathsf{is\text{-}variable}(\mathit{Locvar}, N) \ \neq \ \mathsf{true}}{\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{sum}(N, \mathit{Pre}_1) \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_2, \mathit{Newvar}) \ = \\ <\mathsf{sum}(N, \mathit{Pre}_1 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_2), \mathit{Newvar}>\end{array}} \qquad \text{[bpa-51]}$$

$$\frac{\begin{array}{rcl}\mathsf{is\text{-}variable}(\mathit{Locvar}, N) & = & \mathsf{true}, \\ \mathsf{make\text{-}svds}(\mathit{Globvar} + \mathit{Newvar}, N) & = & N : N', \\ \mathsf{new\text{-}varname}(\mathit{Globvar} + \mathit{Locvar} + \mathit{Newvar}, N) & = & N'', \\ \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1, \mathit{Newvar}) & = & <\mathit{Pre}', \mathit{Newvar}'>\end{array}}{\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{sum}(N, \mathit{Pre}_1) \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_2, \mathit{Newvar}) \ = \\ <\mathsf{sum}(N'', \mathsf{subst}(N'', N, \mathit{Pre}') \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_2), \mathit{Newvar}' + [N'' : N']>\end{array}} \qquad \text{[bpa-52]}$$

$$\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \triangleleft \mathit{Trm} \triangleright \mathit{Pre}_2 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_3, \mathit{Newvar}) \\ = \ <(\mathit{Pre}_1 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_3) \triangleleft \mathit{Trm} \triangleright (\mathit{Pre}_2 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_3), \mathit{Newvar}>\end{array} \qquad \text{[bpa-53]}$$

$$\frac{\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1, \mathit{Newvar}) \ = \ <\mathit{Pre}', \mathit{Newvar}'>}{\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_2, \mathit{Newvar}) \ = \\ <\mathit{Pre}' \ \mathbin{|\!|\!\lfloor} \ \mathit{Pre}_2, \mathit{Newvar}'>\end{array}} \quad \textbf{otherwise} \qquad \text{[bpa-54]}$$

$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{delta}; \mathit{Pre}, \mathit{Newvar}) = < \mathsf{delta}, \mathit{Newvar} >$ [bpa-55]
$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{tau}; \mathit{Pre}, \mathit{Newvar}) \quad = < \mathsf{tau}\ .\ \mathit{Pre}, \mathit{Newvar} >$ [bpa-56]

$$\frac{\mathsf{atomic}(\mathit{Pre}_1) \ = \ \mathsf{true}}{\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1; \mathit{Pre}_2, \mathit{Newvar}) \ = \ < \mathit{Pre}_1\ .\ \mathit{Pre}_2, \mathit{Newvar} >} \quad \text{[bpa-57]}$$

$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{er}(\mathit{Trm}, \mathit{Cns}); \mathit{Pre}, \mathit{Newvar})$ [bpa-58]
$= \ < \mathsf{put\text{-}sums}(\mathit{Cns}, \mathit{Trm}, \mathit{Cns}, \mathit{Pre}, \mathtt{r}), \mathit{Newvar} >$

$$\frac{\mathit{Trms} \ = \ \mathsf{transform}(\mathit{Cns})}{\mathsf{put\text{-}sums}(N, \mathit{Trm}, \mathit{Cns}, \mathit{Pre}, N') \ = \ \mathsf{sum}(N, N'(\mathit{Trm}, \mathit{Trms})\ .\ \mathit{Pre})} \quad \text{[put-sums-1]}$$

$\mathsf{put\text{-}sums}(N, \mathit{Cns}', \mathit{Trm}, \mathit{Cns}, \mathit{Pre}, N') = \mathsf{sum}(N, \mathsf{put\text{-}sums}(\mathit{Cns}', \mathit{Trm}, \mathit{Cns}, \mathit{Pre}, N'))$ [put-sums-2]

$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, (\mathit{Pre}_1 + \mathit{Pre}_2); \mathit{Pre}_3, \mathit{Newvar})$ [bpa-59]
$= \ < \mathit{Pre}_1; \mathit{Pre}_3 + \mathit{Pre}_2; \mathit{Pre}_3, \mathit{Newvar} >$

$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, (\mathit{Pre}_1\ .\ \mathit{Pre}_2); \mathit{Pre}_3, \mathit{Newvar})$ [bpa-60]
$= \ \mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1; (\mathit{Pre}_2; \mathit{Pre}_3), \mathit{Newvar})$

$$\frac{\mathsf{is\text{-}variable}(\mathit{Locvar}, N) \ \neq \ \mathsf{true}}{\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{sum}(N, \mathit{Pre}_1); \mathit{Pre}_2, \mathit{Newvar}) \ = \\ < \mathsf{sum}(N, \mathit{Pre}_1; \mathit{Pre}_2), \mathit{Newvar} >\end{array}} \quad \text{[bpa-61]}$$

$$\frac{\begin{array}{rcl}\mathsf{is\text{-}variable}(\mathit{Locvar}, N) & = & \mathsf{true}, \\ \mathsf{make\text{-}svds}(\mathit{Globvar} + \mathit{Newvar}, N) & = & N : N', \\ \mathsf{new\text{-}varname}(\mathit{Globvar} + \mathit{Locvar} + \mathit{Newvar}, N) & = & N'', \\ \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1, \mathit{Newvar}) & = & < \mathit{Pre}', \mathit{Newvar}' >\end{array}}{\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{sum}(N, \mathit{Pre}_1); \mathit{Pre}_2, \mathit{Newvar}) \ = \\ < \mathsf{sum}(N'', \mathsf{subst}(N'', N, \mathit{Pre}'); \mathit{Pre}_2), \mathit{Newvar}' + [N'' : N'] >\end{array}} \quad \text{[bpa-62]}$$

$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, (\mathit{Pre}_1 \triangleleft \mathit{Trm} \triangleright \mathit{Pre}_2); \mathit{Pre}_3, \mathit{Newvar})$ [bpa-63]
$= \ < \mathit{Pre}_1; \mathit{Pre}_3 \triangleleft \mathit{Trm} \triangleright \mathit{Pre}_2; \mathit{Pre}_3, \mathit{Newvar} >$

$$\frac{\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1, \mathit{Newvar}) \ = \ < \mathit{Pre}', \mathit{Newvar}' >}{\begin{array}{l}\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1; \mathit{Pre}_2, \mathit{Newvar}) \ = \\ < \mathit{Pre}'; \mathit{Pre}_2, \mathit{Newvar}' >\end{array}} \quad \textbf{otherwise} \quad \text{[bpa-64]}$$

$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \parallel \mathit{Pre}_2, \mathit{Newvar}) \quad = < \mathsf{delta}, \mathit{Newvar} >$ [bpa-65]
$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \mathbin{\|\!\!\lfloor} \mathit{Pre}_2, \mathit{Newvar}) = < \mathsf{delta}, \mathit{Newvar} >$ [bpa-66]
$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \mid \mathit{Pre}_2, \mathit{Newvar}) \quad = < \mathsf{delta}, \mathit{Newvar} >$ [bpa-67]
$\mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 * \mathit{Pre}_2, \mathit{Newvar}) \quad = < \mathsf{delta}, \mathit{Newvar} >$ [bpa-68]

$$\frac{\mathsf{is\text{-}bpa}(\mathit{Pre}) \;=\; \mathsf{true}}{\mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}, \mathit{Newvar}) \;=\; < \mathit{Pre}, \mathit{Newvar} >} \quad \text{[normalize-1]}$$

$$\mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{er}(\mathit{Trm}, \mathit{Cns}), \mathit{Newvar}) = < \mathsf{er}(\mathit{Trm}, \mathit{Cns}), \mathit{Newvar} > \quad \text{[normalize-2]}$$

$$\frac{\begin{array}{r} \mathsf{is\text{-}bpa}(\mathit{Pre}) \;\neq\; \mathsf{true}, \\ \mathsf{normalize}(\mathit{Globvar}, \mathsf{add\text{-}to\text{-}locals}(\mathit{Globvar}, \mathit{Locvar}, \mathit{N}), \\ \mathit{Pre}, \mathit{Newvar}) \;=\; < \mathit{Pre}', \mathit{Newvar}' > \end{array}}{\mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathsf{sum}(\mathit{N}, \mathit{Pre}), \mathit{Newvar}) \;=\; < \mathsf{sum}(\mathit{N}, \mathit{Pre}'), \mathit{Newvar}' >} \quad \text{[normalize-3]}$$

$$\frac{\begin{array}{r} \mathsf{is\text{-}bpa}(\mathit{Pre}_1 + \mathit{Pre}_2) \;\neq\; \mathsf{true}, \\ \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1, \mathit{Newvar}) \;=\; < \mathit{Pre}_1', \mathit{Newvar}_1 >, \\ \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_2, \mathit{Newvar}_1) \;=\; < \mathit{Pre}_2', \mathit{Newvar}_2 > \end{array}}{\mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 + \mathit{Pre}_2, \mathit{Newvar}) \;=\; < \mathit{Pre}_1' + \mathit{Pre}_2', \mathit{Newvar}_2 >} \quad \text{[normalize-4]}$$

$$\frac{\begin{array}{r} \mathsf{is\text{-}bpa}(\mathit{Pre}_1 \;.\; \mathit{Pre}_2) \;\neq\; \mathsf{true}, \\ \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1, \mathit{Newvar}) \;=\; < \mathit{Pre}_1', \mathit{Newvar}_1 >, \\ \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_2, \mathit{Newvar}_1) \;=\; < \mathit{Pre}_2', \mathit{Newvar}_2 > \end{array}}{\mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \;.\; \mathit{Pre}_2, \mathit{Newvar}) \;=\; < \mathit{Pre}_1' \;.\; \mathit{Pre}_2', \mathit{Newvar}_2 >} \quad \text{[normalize-5]}$$

$$\frac{\begin{array}{r} \mathsf{is\text{-}bpa}(\mathit{Pre}_1 \lhd \mathit{Trm} \rhd \mathit{Pre}_2) \;\neq\; \mathsf{true}, \\ \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1, \mathit{Newvar}) \;=\; < \mathit{Pre}_1', \mathit{Newvar}_1 >, \\ \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_2, \mathit{Newvar}_1) \;=\; < \mathit{Pre}_2', \mathit{Newvar}_2 > \end{array}}{\begin{array}{l} \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \lhd \mathit{Trm} \rhd \mathit{Pre}_2, \mathit{Newvar}) \;=\; \\ < \mathit{Pre}_1' \lhd \mathit{Trm} \rhd \mathit{Pre}_2', \mathit{Newvar}_2 > \end{array}} \quad \text{[normalize-6]}$$

$$\frac{\begin{array}{r} \mathsf{is\text{-}bpa}(\mathit{Pre}) \;\neq\; \mathsf{true}, \\ \mathsf{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}, \mathit{Newvar}) \;=\; < \mathit{Pre}', \mathit{Newvar}' > \end{array}}{\begin{array}{l} \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}, \mathit{Newvar}) \;=\; \\ \mathsf{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}', \mathit{Newvar}') \end{array}} \quad \textbf{otherwise} \quad \text{[normalize-7]}$$

$$\frac{\mathsf{is\text{-}variable}(\mathit{Var}, \mathsf{name}(\mathit{Chs}\; \texttt{"'"})) \;=\; \mathsf{true}}{\mathsf{new\text{-}varname}(\mathit{Var}, \mathsf{name}(\mathit{Chs})) \;=\; \mathsf{new\text{-}varname}(\mathit{Var}, \mathsf{name}(\mathit{Chs}\; \texttt{"'"}))} \quad \text{[new-varname-1]}$$

$$\frac{\mathsf{is\text{-}variable}(\mathit{Var}, \mathsf{name}(\mathit{Chs}\; \texttt{"'"})) \;\neq\; \mathsf{true}}{\mathsf{new\text{-}varname}(\mathit{Var}, \mathsf{name}(\mathit{Chs})) \;=\; \mathsf{name}(\mathit{Chs}\; \texttt{"'"})} \quad \text{[new-varname-2]}$$

$$\begin{array}{lll}
\mathsf{atomic}(\mathit{N}) & = \mathsf{true} & \text{[atomic-1]} \\
\mathsf{atomic}(\mathit{N}(\mathit{Trms})) & = \mathsf{true} & \text{[atomic-2]} \\
\mathsf{atomic}(\mathsf{encap}(\{\mathit{Cns}\}, \mathit{Pre})) & = \mathsf{atomic}(\mathit{Pre}) & \text{[atomic-3]} \\
\mathsf{atomic}(\mathsf{hide}(\{\mathit{Cns}\}, \mathit{Pre})) & = \mathsf{atomic}(\mathit{Pre}) & \text{[atomic-4]} \\
\mathsf{atomic}(\mathsf{rename}(\{\mathit{Rds}\}, \mathit{Pre})) & = \mathsf{atomic}(\mathit{Pre}) & \text{[atomic-5]}
\end{array}$$

$$\frac{\mathsf{atomic}(\mathit{Pre}) \ = \ \mathsf{true}}{\mathsf{is\text{-}bpa}(\mathit{Pre}) \ = \ \mathsf{true}} \qquad \text{[is-bpa-1]}$$

$$\begin{array}{lll}
\mathsf{is\text{-}bpa}(\mathsf{delta}) & = \mathsf{true} & \text{[is-bpa-2]} \\
\mathsf{is\text{-}bpa}(\mathsf{tau}) & = \mathsf{true} & \text{[is-bpa-3]} \\
\mathsf{is\text{-}bpa}(\mathit{Pre}_1 \ + \ \mathit{Pre}_2) & = \mathsf{is\text{-}bpa}(\mathit{Pre}_1) \ \& \ \mathsf{is\text{-}bpa}(\mathit{Pre}_2) & \text{[is-bpa-4]} \\
\mathsf{is\text{-}bpa}(\mathit{Pre}_1 \ . \ \mathit{Pre}_2) & = \mathsf{is\text{-}bpa}(\mathit{Pre}_1) \ \& \ \mathsf{is\text{-}bpa}(\mathit{Pre}_2) & \text{[is-bpa-5]} \\
\mathsf{is\text{-}bpa}(\mathsf{sum}(\mathit{N}, \ \mathit{Pre})) & = \mathsf{is\text{-}bpa}(\mathit{Pre}) & \text{[is-bpa-6]} \\
\mathsf{is\text{-}bpa}(\mathit{Pre}_1 \ \triangleleft \ \mathit{Trm} \ \triangleright \ \mathit{Pre}_2) & = \mathsf{is\text{-}bpa}(\mathit{Pre}_1) \ \& \ \mathsf{is\text{-}bpa}(\mathit{Pre}_2) & \text{[is-bpa-7]}
\end{array}$$

$$\mathsf{transform}(\mathit{N}) = \mathit{N} \qquad \text{[transf-cns-1]}$$

$$\frac{\mathit{Trms} \ = \ \mathsf{transform}(\mathit{Cns})}{\mathsf{transform}(\mathit{N}, \ \mathit{Cns}) \ = \ \mathit{N}, \ \mathit{Trms}} \qquad \text{[transf-cns-2]}$$

A.10 Translate

imports ToBPA$^{A.9}$ Integers
exports
  context-free syntax
    trans(Specification) → Specification
hiddens
  context-free syntax

trans-spec(Specification)	→ Specification
trans-pds(Var, Process-declaration+)	→ Proc
trans-expr(Var, Var, Process-expression)	→ Process-expression
is-er-merge(Process-expression)	→ BOOL
trans-er-merge(Process-expression, Name)	→ Process-expression
append-reads(Process-expression, Process-expression, Name)	→ Process-expression
add-reads(Act, Types, Specification, Act)	→ Specification
add-comm(Comm, Specification)	→ Specification
add-er-aux(NAT, Types, Specification)	→ Specification
er-aux-ports(NAT)	→ Single-specification
er-aux-ports(NAT, Fun, Name, NAT)	→ Single-specification
er-aux-acts(Types)	→ Single-specification
er-aux-acts(Types, Act)	→ Single-specification
longest-er-merge(Proc)	→ NAT
lerm(Process-expression)	→ NAT
er-length(Process-expression)	→ NAT
max(NAT, NAT)	→ NAT

  variables
    Chs [*0-9′*]∗ → CHAR+
equations

trans-spec(sort *Ns*)	= sort *Ns*	[trans-spec-1]
trans-spec(func *Fds*)	= func *Fds*	[trans-spec-2]
trans-spec(*Var-sec* rew *Rws*)	= *Var-sec* rew *Rws*	[trans-spec-3]
trans-spec(act *Ads*)	= act *Ads*	[trans-spec-4]
trans-spec(comm *Cds*)	= comm *Cds*	[trans-spec-5]

$$\frac{[\mathit{Pds}'] \ = \ \text{trans-pds}([\,], \mathit{Pds})}{\text{trans-spec}(\text{proc } \mathit{Pds}) \ = \ \text{proc } \mathit{Pds}'} \qquad \text{[trans-spec-6]}$$

$$\frac{[\mathit{Pds}'] \ = \ \text{trans-pds}([\mathit{Vds}], \mathit{Pds})}{\text{trans-spec}(\text{var } \mathit{Vds} \text{ proc } \mathit{Pds}) \ = \ \text{proc } \mathit{Pds}'} \qquad \text{[trans-spec-7]}$$

$$\frac{\mathsf{SE}' \;=\; \text{trans-spec}(\mathsf{SE}), \quad \mathit{SEs}' \;=\; \text{trans-spec}(\mathit{SEs})}{\text{trans-spec}(\mathsf{SE}\ \mathit{SEs}) \;=\; \mathsf{SE}'\ \mathit{SEs}'} \qquad \text{[trans-spec-8]}$$

$$\text{trans-pds}(\mathit{Globvar}, N = \mathit{Pre}) = [N = \text{trans-expr}(\mathit{Globvar}, [], \mathit{Pre})] \qquad \text{[trans-pds-1]}$$

$$\text{trans-pds}(\mathit{Globvar}, N(\mathit{Svds}) = \mathit{Pre}) \qquad \text{[trans-pds-2]}$$
$$= \; [N(\mathit{Svds}) = \text{trans-expr}(\mathit{Globvar}, \text{transform}(\mathit{Svds}), \mathit{Pre})]$$

$$\frac{\mathit{Svds} \;=\; \text{make-svds}(\mathit{Globvar}, \mathit{Cns})}{\begin{array}{l}\text{trans-pds}(\mathit{Globvar}, N(\mathit{Cns}) = \mathit{Pre}) \;=\\ {[N(\mathit{Svds}) = \text{trans-expr}(\mathit{Globvar}, \text{get-locals}(\mathit{Globvar}, \mathit{Cns}), \mathit{Pre})]}\end{array}} \qquad \text{[trans-pds-3]}$$

$$\frac{\mathit{Proc}_1 \;=\; \text{trans-pds}(\mathit{Globvar}, \mathit{Pd}), \quad \mathit{Proc}_2 \;=\; \text{trans-pds}(\mathit{Globvar}, \mathit{Pds})}{\text{trans-pds}(\mathit{Globvar}, \mathit{Pd}\ \mathit{Pds}) \;=\; \mathit{Proc}_1 + \mathit{Proc}_2} \qquad \text{[trans-pds-4]}$$

$$\text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 + \mathit{Pre}_2) \qquad \text{[trans-expr-1]}$$
$$= \; \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1) + \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_2)$$

$$\text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \;\|\!\_\; \mathit{Pre}_2) \qquad \text{[trans-expr-2]}$$
$$= \; \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1) \;\|\!\_\; \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_2)$$

$$\text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \parallel \mathit{Pre}_2) \qquad \text{[trans-expr-3]}$$
$$= \; \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1) \parallel \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_2)$$

$$\text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \mid \mathit{Pre}_2) \qquad \text{[trans-expr-4]}$$
$$= \; \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1) \mid \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_2)$$

$$\text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \;|\!|\!|\!\_\; \mathit{Pre}_2) \qquad \text{[trans-expr-5]}$$
$$= \; \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1) \;|\!|\!|\!\_\; \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_2)$$

$$\text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \;|\!|\!|\; \mathit{Pre}_2) \qquad \text{[trans-expr-6]}$$
$$= \; \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1) \;|\!|\!|\; \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_2)$$

$$\text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1 \triangleleft \mathit{Trm} \triangleright \mathit{Pre}_2) \qquad \text{[trans-expr-7]}$$
$$= \; \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1) \triangleleft \mathit{Trm} \triangleright \text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_2)$$

$$\frac{\begin{array}{r}\text{is-er-merge}(\mathit{Pre}_1) \;\neq\; \text{true},\\ \text{normalize}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1, []) \;=\; <\mathit{Pre}, \mathit{Newvar}>,\\ \text{bpa}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}; \mathit{Pre}_2, \mathit{Newvar}) \;=\; <\mathit{Pre}', \mathit{Newvar}'>\end{array}}{\begin{array}{l}\text{trans-expr}(\mathit{Globvar}, \mathit{Locvar}, \mathit{Pre}_1; \mathit{Pre}_2) \;=\\ \text{trans-expr}(\mathit{Globvar} + \mathit{Newvar}, \mathit{Locvar}, \mathit{Pre}')\end{array}} \qquad \text{[trans-expr-8]}$$

$$\frac{\begin{array}{l}\text{is-er-merge}(Pre_1) \;=\; \text{true}, \\ \qquad\qquad Pre' \;=\; \text{hide}(\{\texttt{er-aux-c}\} \\ \qquad\qquad\qquad\qquad , \text{encap}(\{\texttt{er-aux-r}, \texttt{er-aux-s}\} \\ \qquad\qquad\qquad\qquad\qquad , \text{trans-er-merge}(Pre_1, \texttt{er-aux-port}) \\ \qquad\qquad\qquad\qquad\qquad\; \| \; \text{append-reads}(Pre_1, Pre_2, \texttt{er-aux-port})))\end{array}}{\text{trans-expr}(Globvar, Locvar, Pre_1; Pre_2) \;=\; \text{trans-expr}(Globvar, Locvar, Pre')} \quad \text{[trans-expr-9]}$$

$\text{trans-expr}(Globvar, Locvar, Pre_1 \;.\; Pre_2)$ [trans-expr-10]
$= \;\text{trans-expr}(Globvar, Locvar, Pre_1) \;.\; \text{trans-expr}(Globvar, Locvar, Pre_2)$

$\text{trans-expr}(Globvar, Locvar, Pre_1 * Pre_2)$ [trans-expr-11]
$= \;\text{trans-expr}(Globvar, Locvar, Pre_1) * \text{trans-expr}(Globvar, Locvar, Pre_2)$

$\text{trans-expr}(Globvar, Locvar, \text{delta}) = \text{delta}$ [trans-expr-12]
$\text{trans-expr}(Globvar, Locvar, \text{tau}) \quad = \text{tau}$ [trans-expr-13]

$\text{trans-expr}(Globvar, Locvar, \text{encap}(\{Cns\}, Pre))$ [trans-expr-14]
$= \;\text{encap}(\{Cns\}, \text{trans-expr}(Globvar, Locvar, Pre))$

$\text{trans-expr}(Globvar, Locvar, \text{hide}(\{Cns\}, Pre))$ [trans-expr-15]
$= \;\text{hide}(\{Cns\}, \text{trans-expr}(Globvar, Locvar, Pre))$

$\text{trans-expr}(Globvar, Locvar, \text{rename}(\{Rds\}, Pre))$ [trans-expr-16]
$= \;\text{rename}(\{Rds\}, \text{trans-expr}(Globvar, Locvar, Pre))$

$$\frac{Svd \;=\; \text{make-svds}(Globvar, N)}{\begin{array}{l}\text{trans-expr}(Globvar, Locvar, \text{sum}(N, Pre)) \;= \\ \text{sum}(Svd, \text{trans-expr}(Globvar, \\ \qquad\qquad\qquad \text{add-to-locals}(Globvar, Locvar, N), \\ \qquad\qquad\qquad Pre))\end{array}} \quad \text{[trans-expr-17]}$$

$\text{trans-expr}(Globvar, Locvar, N) \qquad\qquad\quad = N$ [trans-expr-18]
$\text{trans-expr}(Globvar, Locvar, N(Trms)) \qquad = N(Trms)$ [trans-expr-19]
$\text{trans-expr}(Globvar, Locvar, \text{er}(Trm, Cns)) = \text{name}(\texttt{"e" "r"})$ [trans-expr-20]

$\text{trans-expr}(Globvar, Locvar, \text{sum}(N : N', Pre))$ [trans-expr-21]
$= \;\text{sum}(N : N', \text{trans-expr}(Globvar, Locvar \mid N + [N : N'], Pre))$

$\text{is-er-merge}(\text{er}(Trm, Cns) \;|||\; \text{er}(Trm', Cns')) = \text{true}$ [is-er-merge-1]

$\text{is-er-merge}(Pre \;|||\; \text{er}(Trm, Cns) \;|||\; \text{er}(Trm', Cns'))$ [is-er-merge-2]
$= \;\text{is-er-merge}(Pre \;|||\; \text{er}(Trm, Cns))$

$$\frac{\mathit{Trms} \;=\; \mathsf{transform}(\mathit{Cns})}{\begin{array}{l}\mathsf{trans\text{-}er\text{-}merge}(\mathsf{er}(\mathit{Trm},\, \mathit{Cns}),\, \mathit{N}) \;=\\ \mathsf{put\text{-}sums}(\mathit{Cns},\, \mathit{Trm},\, \mathit{Cns},\, \mathtt{er\text{-}aux\text{-}s}(\mathit{N},\, \mathit{Trms}),\, \mathtt{r})\end{array}} \quad \text{[trans-er-merge-1]}$$

$$\frac{\begin{array}{rcl}\mathit{N} &=& \mathsf{name}(\mathit{Chs}),\\ \mathit{N}' &=& \mathsf{name}(\mathit{Chs}\ \mathtt{"'"})\end{array}}{\begin{array}{l}\mathsf{trans\text{-}er\text{-}merge}(\mathit{Pre}\ |||\ \mathsf{er}(\mathit{Trm},\, \mathit{Cns}),\, \mathit{N}) \;=\\ \mathsf{trans\text{-}er\text{-}merge}(\mathsf{er}(\mathit{Trm},\, \mathit{Cns}),\, \mathit{N}) \parallel \mathsf{trans\text{-}er\text{-}merge}(\mathit{Pre},\, \mathit{N}')\end{array}} \quad \text{[trans-er-merge-2]}$$

$$\mathsf{trans\text{-}er\text{-}merge}(\mathit{Pre},\, \mathit{N}) = \mathsf{delta} \quad \textbf{otherwise} \quad \text{[trans-er-merge-3]}$$

$$\frac{\mathit{Trm}' \;=\; \mathit{N}}{\mathsf{append\text{-}reads}(\mathsf{er}(\mathit{Trm},\, \mathit{Cns}),\, \mathit{Pre},\, \mathit{N}) \;=\; \mathsf{put\text{-}sums}(\mathit{Cns},\, \mathit{Trm}',\, \mathit{Cns},\, \mathit{Pre},\, \mathtt{er\text{-}aux\text{-}r})} \quad \text{[append-reads-1]}$$

$$\frac{\begin{array}{rcl}\mathit{N} &=& \mathsf{name}(\mathit{Chs}),\\ \mathit{N}' &=& \mathsf{name}(\mathit{Chs}\ \mathtt{"'"}),\\ \mathsf{append\text{-}reads}(\mathit{Pre}',\, \mathit{Pre},\, \mathit{N}') &=& \mathit{Pre}'',\\ \mathit{Trm}' &=& \mathit{N}\end{array}}{\begin{array}{l}\mathsf{append\text{-}reads}(\mathit{Pre}'\ |||\ \mathsf{er}(\mathit{Trm},\, \mathit{Cns}),\, \mathit{Pre},\, \mathit{N}) \;=\\ \mathsf{put\text{-}sums}(\mathit{Cns},\, \mathit{Trm}',\, \mathit{Cns},\, \mathit{Pre}'',\, \mathtt{er\text{-}aux\text{-}r})\end{array}} \quad \text{[append-reads-2]}$$

$$\mathsf{append\text{-}reads}(\mathit{Pre},\, \mathit{Pre}',\, \mathit{N}) = \mathsf{delta} \quad \textbf{otherwise} \quad \text{[append-reads-3]}$$

$$\mathsf{add\text{-}reads}(\mathit{Act},\, [],\, \mathit{E},\, []) = \mathit{E} \quad \text{[add-reads-1]}$$

$$\frac{\begin{array}{rcl}\mathsf{SE} &=& \mathsf{act}\ \mathit{Ads},\\ \mathit{SEs} &=& \mathit{E}\end{array}}{\mathsf{add\text{-}reads}(\mathit{Act},\, [],\, \mathit{E},\, [\mathit{Ads}]) \;=\; \mathit{SEs}\ \mathsf{SE}} \quad \text{[add-reads-2]}$$

$$\frac{\mathit{Ip} \;=\; \mathsf{empty\text{-}pattern}}{\mathsf{add\text{-}reads}(\mathit{Act}_1,\, [\mathit{Ip}\ \mathit{OIps}],\, \mathit{E},\, \mathit{Act}_2) \;=\; \mathsf{add\text{-}reads}(\mathit{Act}_1,\, [\mathit{OIps}],\, \mathit{E},\, \mathit{Act}_2)} \quad \text{[add-reads-3]}$$

$$\frac{\begin{array}{rcl}\mathit{Ip} &=& \mathit{Xns},\\ \mathsf{is\text{-}action}(\mathit{Act}_1,\, \mathtt{r} : \mathit{Xns}) &=& \mathsf{true}\end{array}}{\mathsf{add\text{-}reads}(\mathit{Act}_1,\, [\mathit{Ip}\ \mathit{OIps}],\, \mathit{E},\, \mathit{Act}_2) \;=\; \mathsf{add\text{-}reads}(\mathit{Act}_1,\, [\mathit{OIps}],\, \mathit{E},\, \mathit{Act}_2)} \quad \text{[add-reads-4]}$$

$$\frac{\begin{array}{rcl}\mathit{Ip} &=& \mathit{Xns},\\ \mathsf{is\text{-}action}(\mathit{Act}_1,\, \mathtt{r} : \mathit{Xns}) &\neq& \mathsf{true},\\ \mathsf{is\text{-}action}(\mathit{Act}_2,\, \mathtt{r} : \mathit{Xns}) &=& \mathsf{true}\end{array}}{\mathsf{add\text{-}reads}(\mathit{Act}_1,\, [\mathit{Ip}\ \mathit{OIps}],\, \mathit{E},\, \mathit{Act}_2) \;=\; \mathsf{add\text{-}reads}(\mathit{Act}_1,\, [\mathit{OIps}],\, \mathit{E},\, \mathit{Act}_2)} \quad \text{[add-reads-5]}$$

$$\frac{\begin{array}{rcl}\mathit{Ip} &=& \mathit{Xns},\\ \mathsf{is\text{-}action}(\mathit{Act}_1,\, \mathtt{r} : \mathit{Xns}) &\neq& \mathsf{true},\\ \mathsf{is\text{-}action}(\mathit{Act}_2,\, \mathtt{r} : \mathit{Xns}) &\neq& \mathsf{true}\end{array}}{\mathsf{add\text{-}reads}(\mathit{Act}_1,\, [\mathit{Ip}\ \mathit{OIps}],\, \mathit{E},\, \mathit{Act}_2) \;=\; \mathsf{add\text{-}reads}(\mathit{Act}_1,\, [\mathit{OIps}],\, \mathit{E},\, \mathit{Act}_2 + [\mathtt{r} : \mathit{Xns}])} \quad \text{[add-reads-6]}$$

$$\frac{\text{is-comm}(\mathit{Comm}, \mathtt{r} \mid \mathtt{s}) \;=\; \text{true}}{\text{add-comm}(\mathit{Comm}, E) \;=\; E} \quad \text{[add-comm-1]}$$

$$\frac{\begin{array}{rcl}\text{is-comm}(\mathit{Comm}, \mathtt{r} \mid \mathtt{s}) & \neq & \text{true},\\ \text{SE} & = & \text{comm } \mathtt{r} \mid \mathtt{s} = \mathtt{c},\\ \mathit{SEs} & = & E\end{array}}{\text{add-comm}(\mathit{Comm}, E) \;=\; \mathit{SEs}\ \text{SE}} \quad \text{[add-comm-2]}$$

$$\frac{\begin{array}{rcl}\mathit{SE}_1 & = & \text{sort } \mathtt{Er\text{-}aux\text{-}port},\\ \mathit{SE}_2 & = & \text{er-aux-ports}(\mathit{Nat}),\\ \mathit{SE}_3 & = & \text{er-aux-acts}(\mathit{Types}),\\ \mathit{SE}_4 & = & \text{comm } \mathtt{er\text{-}aux\text{-}r} \mid \mathtt{er\text{-}aux\text{-}s} = \mathtt{er\text{-}aux\text{-}c},\\ \mathit{SEs} & = & E\end{array}}{\text{add-er-aux}(\mathit{Nat}, \mathit{Types}, E) \;=\; \mathit{SEs}\ \mathit{SE}_1\ \mathit{SE}_2\ \mathit{SE}_3\ \mathit{SE}_4} \quad \text{[add-er-aux-1]}$$

$$\begin{array}{lll}\text{er-aux-ports}(\mathit{Nat}) & = \text{er-aux-ports}(\mathit{Nat}, [], \mathtt{er\text{-}aux\text{-}port}, 0) & \text{[er-aux-ports-1]}\\ \text{er-aux-ports}(\mathit{Nat}, [\mathit{OFds}], N, \mathit{Nat}) & = \text{func } \mathit{OFds} & \text{[er-aux-ports-2]}\end{array}$$

$$\frac{\begin{array}{rcl}\mathit{Nat}_1 & \neq & \mathit{Nat}_2,\\ N & = & \text{name}(\mathit{Chs}),\\ N' & = & \text{name}(\mathit{Chs}\ \texttt{"'"}),\\ \mathit{Nat}' & = & \mathit{Nat}_2 + 1\end{array}}{\begin{array}{l}\text{er-aux-ports}(\mathit{Nat}_1, \mathit{Fun}, N, \mathit{Nat}_2) \;=\\ \text{er-aux-ports}(\mathit{Nat}_1, \mathit{Fun} + [N : \ \to \mathtt{Er\text{-}aux\text{-}port}], N', \mathit{Nat}')\end{array}} \quad \text{[er-aux-ports-3]}$$

$$\begin{array}{lll}\text{er-aux-acts}(\mathit{Types}) & = \text{er-aux-acts}(\mathit{Types}, []) & \text{[er-aux-acts-1]}\\ \text{er-aux-acts}([], [\mathit{OAds}]) & = \text{act } \mathit{OAds} & \text{[er-aux-acts-2]}\end{array}$$

$$\frac{\mathit{Ip} \;=\; \text{empty-pattern}}{\text{er-aux-acts}([\mathit{Ip}\ \mathit{OIps}], \mathit{Act}) \;=\; \text{er-aux-acts}([\mathit{OIps}], \mathit{Act})} \quad \text{[er-aux-acts-3]}$$

$$\frac{\mathit{Ip} \;=\; N}{\text{er-aux-acts}([\mathit{Ip}\ \mathit{OIps}], \mathit{Act}) \;=\; \text{er-aux-acts}([\mathit{OIps}], \mathit{Act})} \quad \text{[er-aux-acts-4]}$$

$$\frac{\begin{array}{rcl}\mathit{Ip} & = & N\#\ \mathit{Xns},\\ \text{is-action}(\mathit{Act}, \mathtt{er\text{-}aux\text{-}c} : \mathtt{Er\text{-}aux\text{-}port}\#\ \mathit{Xns}) & = & \text{true}\end{array}}{\text{er-aux-acts}([\mathit{Ip}\ \mathit{OIps}], \mathit{Act}) \;=\; \text{er-aux-acts}([\mathit{OIps}], \mathit{Act})} \quad \text{[er-aux-acts-5]}$$

$$\frac{\begin{array}{rcl}\mathit{Ip} & = & N\#\ \mathit{Xns},\\ \text{is-action}(\mathit{Act}, \mathtt{er\text{-}aux\text{-}c} : \mathtt{Er\text{-}aux\text{-}port}\#\ \mathit{Xns}) & \neq & \text{true}\end{array}}{\begin{array}{l}\text{er-aux-acts}([\mathit{Ip}\ \mathit{OIps}], \mathit{Act}) \;=\\ \text{er-aux-acts}([\mathit{OIps}], \mathit{Act} + [\mathtt{er\text{-}aux\text{-}r}, \mathtt{er\text{-}aux\text{-}s}, \mathtt{er\text{-}aux\text{-}c} : \mathtt{Er\text{-}aux\text{-}port}\#\ \mathit{Xns}])\end{array}} \quad \text{[er-aux-acts-6]}$$

$$\text{longest-er-merge}([]) \;=\; 0 \quad \text{[lerm-1]}$$

$$\text{longest-er-merge}([N = \mathit{Pre}\ \mathit{OPds}]) = \max(\text{lerm}(\mathit{Pre}), \text{longest-er-merge}([\mathit{OPds}])) \quad \text{[lerm-2]}$$

$$\text{longest-er-merge}([N(\mathit{Svds}) = \mathit{Pre}\ \mathit{OPds}]) = \max(\text{lerm}(\mathit{Pre}), \text{longest-er-merge}([\mathit{OPds}])) \quad \text{[lerm-3]}$$

$$\begin{array}{lll}
\text{lerm}(\mathit{Pre}_1 + \mathit{Pre}_2) & = \max(\text{lerm}(\mathit{Pre}_1), \text{lerm}(\mathit{Pre}_2)) & \text{[lerm-3]} \\
\text{lerm}(\mathit{Pre}_1 \mathbin{\|\!_} \mathit{Pre}_2) & = \max(\text{lerm}(\mathit{Pre}_1), \text{lerm}(\mathit{Pre}_2)) & \text{[lerm-4]} \\
\text{lerm}(\mathit{Pre}_1 \parallel \mathit{Pre}_2) & = \max(\text{lerm}(\mathit{Pre}_1), \text{lerm}(\mathit{Pre}_2)) & \text{[lerm-5]} \\
\text{lerm}(\mathit{Pre}_1 \mid \mathit{Pre}_2) & = \max(\text{lerm}(\mathit{Pre}_1), \text{lerm}(\mathit{Pre}_2)) & \text{[lerm-6]} \\
\text{lerm}(\mathit{Pre}_1 \mathbin{|||} \mathit{Pre}_2) & = \max(\text{lerm}(\mathit{Pre}_1), \text{lerm}(\mathit{Pre}_2)) & \text{[lerm-7]} \\
\text{lerm}(\mathit{Pre}_1 \mathbin{|||\!_} \mathit{Pre}_2) & = \max(\text{lerm}(\mathit{Pre}_1), \text{lerm}(\mathit{Pre}_2)) & \text{[lerm-8]} \\
\text{lerm}(\mathit{Pre}_1 \triangleleft \mathit{Trm} \triangleright \mathit{Pre}_2) & = \max(\text{lerm}(\mathit{Pre}_1), \text{lerm}(\mathit{Pre}_2)) & \text{[lerm-9]}
\end{array}$$

$$\frac{\text{is-er-merge}(\mathit{Pre}_1) \neq \text{true}}{\text{lerm}(\mathit{Pre}_1 ; \mathit{Pre}_2) = \max(\text{lerm}(\mathit{Pre}_1), \text{lerm}(\mathit{Pre}_2))} \quad \text{[lerm-10]}$$

$$\frac{\text{is-er-merge}(\mathit{Pre}_1) = \text{true}}{\text{lerm}(\mathit{Pre}_1 ; \mathit{Pre}_2) = \max(\text{er-length}(\mathit{Pre}_1), \text{lerm}(\mathit{Pre}_2))} \quad \text{[lerm-11]}$$

$$\begin{array}{lll}
\text{lerm}(\mathit{Pre}_1 \,.\, \mathit{Pre}_2) & = \max(\text{lerm}(\mathit{Pre}_1), \text{lerm}(\mathit{Pre}_2)) & \text{[lerm-12]} \\
\text{lerm}(\mathit{Pre}_1 * \mathit{Pre}_2) & = \max(\text{lerm}(\mathit{Pre}_1), \text{lerm}(\mathit{Pre}_2)) & \text{[lerm-13]} \\
\text{lerm}(\text{delta}) & = 0 & \text{[lerm-14]} \\
\text{lerm}(\text{tau}) & = 0 & \text{[lerm-15]} \\
\text{lerm}(\text{encap}(\{\mathit{Cns}\}, \mathit{Pre})) & = \text{lerm}(\mathit{Pre}) & \text{[lerm-16]} \\
\text{lerm}(\text{hide}(\{\mathit{Cns}\}, \mathit{Pre})) & = \text{lerm}(\mathit{Pre}) & \text{[lerm-17]} \\
\text{lerm}(\text{rename}(\{\mathit{Rds}\}, \mathit{Pre})) & = \text{lerm}(\mathit{Pre}) & \text{[lerm-18]} \\
\text{lerm}(\text{sum}(N, \mathit{Pre})) & = \text{lerm}(\mathit{Pre}) & \text{[lerm-19]} \\
\text{lerm}(\text{sum}(\mathit{Svd}, \mathit{Pre})) & = \text{lerm}(\mathit{Pre}) & \text{[lerm-20]} \\
\text{lerm}(N) & = 0 & \text{[lerm-21]} \\
\text{lerm}(N(\mathit{Trms})) & = 0 & \text{[lerm-22]} \\
\text{lerm}(\text{er}(\mathit{Trm}, \mathit{Cns})) & = 0 & \text{[lerm-23]}
\end{array}$$

$$\text{er-length}(\text{er}(\mathit{Trm}, \mathit{Cns})) = 1 \quad \text{[er-length-1]}$$

$$\frac{\mathit{Nat} = \text{er-length}(\mathit{Pre}) + 1}{\text{er-length}(\mathit{Pre} \mathbin{|||} \text{er}(\mathit{Trm}, \mathit{Cns})) = \mathit{Nat}} \quad \text{[er-length-2]}$$

$$\text{er-length}(\mathit{Pre}) = 0 \quad \textbf{otherwise} \quad \text{[er-length-3]}$$

$$\frac{\mathit{Nat}_1 > \mathit{Nat}_2 = \text{true}}{\max(\mathit{Nat}_1, \mathit{Nat}_2) = \mathit{Nat}_1} \quad \text{[max-1]}$$

$$\frac{\mathit{Nat}_1 > \mathit{Nat}_2 \neq \text{true}}{\max(\mathit{Nat}_1, \mathit{Nat}_2) = \mathit{Nat}_2} \quad \text{[max-2]}$$

$$\frac{\begin{array}{rcl} \mathit{Sig} &=& \textsf{sig-gpsl}(E),\\ \mathit{Act} &=& \textsf{sigact}(\mathit{Sig}),\\ \mathit{Types}_1 &=& \textsf{er-types}(\mathit{Sig}, E),\\ \mathit{Types}_2 &=& \textsf{types}(\mathit{Act}, \texttt{r}),\\ \mathit{Types}_3 &=& \textsf{types}(\mathit{Act}, \texttt{s}),\\ \mathit{Types} &=& \mathit{Types}_1 + \mathit{Types}_2 + \mathit{Types}_3,\\ \mathit{Types} &=& [] \end{array}}{\textsf{trans}(E) = \textsf{add-reads}(\mathit{Act}, \mathit{Types}_1, \textsf{trans-spec}(E), [])} \quad \text{[trans-1]}$$

$$\frac{\begin{array}{rcl} \mathit{Sig} &=& \textsf{sig-gpsl}(E),\\ \mathit{Act} &=& \textsf{sigact}(\mathit{Sig}),\\ \mathit{Types}_1 &=& \textsf{er-types}(\mathit{Sig}, E),\\ \mathit{Types}_2 &=& \textsf{types}(\mathit{Act}, \texttt{r}),\\ \mathit{Types}_3 &=& \textsf{types}(\mathit{Act}, \texttt{s}),\\ \mathit{Types} &=& \mathit{Types}_1 + \mathit{Types}_2 + \mathit{Types}_3,\\ \mathit{Types} &\neq& [],\\ \mathit{Types}_1 &=& [] \end{array}}{\textsf{trans}(E) = \textsf{add-comm}(\textsf{sigcomm}(\mathit{Sig}), \textsf{add-reads}(\mathit{Act}, \mathit{Types}_1, \textsf{trans-spec}(E), []))} \quad \text{[trans-2]}$$

$$\frac{\begin{array}{rcl} \mathit{Sig} &=& \textsf{sig-gpsl}(E),\\ \mathit{Act} &=& \textsf{sigact}(\mathit{Sig}),\\ \mathit{Types}_1 &=& \textsf{er-types}(\mathit{Sig}, E),\\ \mathit{Types}_2 &=& \textsf{types}(\mathit{Act}, \texttt{r}),\\ \mathit{Types}_3 &=& \textsf{types}(\mathit{Act}, \texttt{s}),\\ \mathit{Types} &=& \mathit{Types}_1 + \mathit{Types}_2 + \mathit{Types}_3,\\ \mathit{Types} &\neq& [],\\ \mathit{Types}_1 &\neq& [] \end{array}}{\begin{array}{l} \textsf{trans}(E) = \textsf{add-er-aux}(\textsf{longest-er-merge}(\textsf{sigproc}(\mathit{Sig})),\\ \qquad \mathit{Types}_1,\\ \qquad \textsf{add-comm}(\textsf{sigcomm}(\mathit{Sig}),\\ \qquad\qquad \textsf{add-reads}(\mathit{Act}, \mathit{Types}_1, \textsf{trans-spec}(E), []))) \end{array}} \quad \text{[trans-3]}$$