NAME

exp, EXP - language for describing networks of communicating automata

DESCRIPTION

This manual page presents EXP 2.0, which is a conservative extension of a previous version EXP 1.0. EXP 2.0 allows networks of communicating automata (also called *behaviours* in the sequel) to be modelled using operators of several languages.

In addition to LOTOS parallel composition and hiding [ISO-89], already implemented in EXP 1.0, EXP 2.0 contains operators taken from E-LOTOS [ISO-01] and LNT, CCS [Milner-89], CSP [Brookes-Hoare-Roscoe-84], and mCRL [Groote-Ponse-90], as well as parallel composition using synchronisation vectors, and generalized operators for label hiding, renaming, and cutting (also called restriction in CCS or encapsulation in mCRL). EXP2.0 also contains a priority operator that allows priorities between transitions to be defined.

Since these operators are taken from various languages that are not always compatible (for instance, the hidden event is written "i" in LOTOS, E-LOTOS, and LNT, "t" in CCS and CSP, "tau" in mCRL), **exp.open** allows the graph module to be customized with respect to a particular language (called the *reference language*).

The customization operates on the following parameters:

- The string representing hidden events.
- The string representing the gate used to express behaviour termination (e.g., "exit" in LOTOS, E-LOTOS, and LNT, "delta" in CSP, irrelevant in CCS and mCRL).
- Case sensitivity, i.e., whether labels in the EXP file should be turned to upper-case or not.
- The meaning of symbols existing in different languages with different semantics ("\" in CCS and CSP, "||" in LOTOS and mCRL).

See **exp.open**(LOCAL) for options enabling a definition of these parameters. When these options are not used, default values are associated to each of these parameters, depending on the reference language selected. For backward compatibility reasons, there is no reference language selected by default. See Section LANGUAGE PARAMETERS for details.

Beware that the **aldebaran**(LOCAL) tool only works with EXP 1.0 descriptions and that the **exp2fc2** tool is now deprecated.

The publication [Lang-05] provides an overview of **exp.open** 2.0 and its input language.

SYNTAX OF EXP 2.0

The syntax of EXP 2.0 is presented using a grammar in the Extended Backus-Naur Form (EBNF):

- Non-terminal symbols are written using capital letters (e.g., *B*).
- Keywords are written using lowercase characters (e.g., hide, par, in, etc.).
- Key symbols are written between double quotes to avoid ambiguities with EBNF symbols (in particular, "|", "(", "[", "*", etc.). However, the quotes may be omitted for non-ambiguous symbols (",", "<-", etc.)

- Definitions in natural language are enclosed in angles (e.g., <filename>).

- Optional sequences of symbols are written between square brackets.

- Star (∗) denotes repetition zero or more times of the preceding EBNF expression.

- Plus (+) denotes repetition one or more times of the preceding EBNF expression.

- Operators ∗ and + have precedence over other EBNF operators. Parentheses are used to associate sequences of symbols (e.g., "*G* (, *G*)∗").

The EXP 2.0 grammar is presented in several parts, corresponding to the different languages supported. We write "*B* ::= ... | *expression*" to express that the definition of *B* is extended with *expression*. The entry point of the grammar is the non-terminal symbol *AXIOM*.

The following productions are common to all languages. *IDF* stands for any sequence of letters, digits, and underscores ("_"), which begins with a letter and does not end with an underscore. *LTS* stands for files containing an automaton, *FILE* for other files, *G* for gates, *GL* for gate lists, *L* for labels, and *LL* for label lists.

Comments of the form
   (∗ <any text on one or several lines> ∗)
and
   -- <any text on one line>
are allowed in EXP 2.0 behaviours.

LTS ::= "<filename>"
   | *IDF* (∗ automaton without extension ∗)
   | "<filename>.bcg"
   | *IDF*.bcg (∗ automaton in BCG format ∗)
   | "<filename>.aut"
   | *IDF*.aut (∗ automaton in AUT format ∗)
   | "<filename>.seq"
   | *IDF*.seq (∗ automaton in SEQ format ∗)
   | "<filename>.fc2
   | *IDF*.fc2 (∗ automaton in FC2 format ∗)
   | "<filename>.exp" (∗ include file in EXP format ∗)

FILE ::= "<filename>"
   | *IDF* (∗ file without extension ∗)
   | "<filename>(.hid|.hide)"
   | *IDF*(.hid|.hide) (∗ hide file ∗)
   | "<filename>(.ren|.rename)"
   | *IDF*(.ren|.rename) (∗ rename file ∗)
   | "<filename>.cut"
   | *IDF*.cut (∗ cut file ∗)

G ::= *IDF* (∗ gate ∗)
  | "<string denoting a gate>"

GL ::= [*G* (, *G*)∗] (∗ gate list ∗)

L ::= *G* (∗ label ∗)
  | "<gate with offers>"

| "<regular expression denoting a gate>"
| "<regular expression denoting a gate with offers>"

LL ::= [*L* (, *L*)*] (* label list *)

Note that *L* can be a gate, or a gate followed by experiment offers, or a regular expression denoting a gate possibly followed by experiment offers. (See the definition of regular expressions in the **regexp**(LOCAL) man page.) The gate of a label *L* is the sub-string starting at the beginning of *L* and ending at the first character !, ?, (, space, or tabulation, if any, or at the end of *L* otherwise.

For instance, "G", "G.*", "G !1", "G(1)", "G.* !.*" are labels, among which "G" is a gate, "G.*" is a regular expression denoting a gate or a gate with offers, "G !1" is a gate with offers, and "G.* !.*" is a regular expression denoting a gate with offers (following LOTOS syntax). Note that the syntax of offers is not restricted to LOTOS: "G(1, 2)" also denotes a gate with offers.

Double quotes around a label can be omitted if and only if the label is a gate satisfying the syntax of *IDF*, but they are mandatory to avoid syntactic ambiguities when a gate has the same name as a reserved EXP keyword (e.g. "cut", "all", etc.) or when a label contains special characters.

Note: all escape character sequences defined in the specification of ANSI C character constants, except octal numbers, hexadecimal numbers, and '\n', are allowed in labels and filenames enclosed in double quotes. These escape character sequences are replaced by a single character accordingly. Every '\' character that does not belong to an ANSI C escape character sequence is kept unchanged. For instance, '\"' and '\\' are replaced respectively by '"' and '\', whereas '\p' and '\(' are kept unchanged.

LOTOS OPERATORS

AXIOM ::= [[lotos] (behaviour | behavior)] *B*

B ::= *LTS*
| hide *GL* in *B*
| *B* "||" *B*
| *B* "|||" *B*
| *B* "|[" *GL* "]|" *B*
| (*B*)

Note that the LOTOS syntax corresponds exactly to EXP 1.0, except the "lotos" keyword, which was absent in EXP 1.0. In EXP 2.0, LOTOS operators are subsumed by E-LOTOS/LNT operators, but they are kept for backward compatibility with EXP 1.0.

E-LOTOS/LNT OPERATORS

AXIOM ::= ...
| [[elotos] (behaviour | behavior)] *B*

B ::= ...

| [gate | total | partial] hide
([all but] *LL* | using *FILE*) in *B* end hide

| [gate | total | single | multiple] rename
(*L* -> *L* (, *L* -> *L*)* | using *FILE*) in *B*
end rename

| [gate | label] par [(all | *S*) in]
[*S* ->] *B* ("||" [*S* ->] *B*)+ end par

S ::= [*L* [# *N*] (, *L* [# *N*])*]

N ::= <natural number>

CCS OPERATORS

AXIOM ::= ...
| [[ccs] (behaviour | behavior)] *B*

B ::= ...
| *B* "|" *B*
| *B* \ *L*
| *B* \ "{" *LL* "}"
| *B* "[" *L* / *L* (, *L* / *L*)* "]"

CSP OPERATORS

AXIOM ::= ...
| [[csp] (behaviour | behavior)] *B*

B ::= ...
| *B* "|||" *B*
| *B* "[|" "{" *GL* "}" "|]" *B*
| *B* "[" "{" *GL* "}" "||" "{" *GL* "}" "]" *B*
| *B* \ *L*
| *B* \ "{" *LL* "}"
| *B* "[[" *L* <- *L* (, *L* <- *L*)* "]]"

MCRL OPERATORS

AXIOM ::= ...
| [[mcrl] (behaviour | behavior)]
[comm *C* (, *C*)* end comm] *B*

C ::= *G* "|" *G* "=" *G*

B ::= ...
| *B* "||" *B*

The mCRL operators "hide", "rename", and "encaps" are subsumed by the operators "hide", "rename", and "cut" presented in Sections "E-LOTOS/LNT OPERATORS" above and "OTHER OPERATORS" below.

OTHER OPERATORS

B ::= ...

| [gate | total | partial] cut
([all but] *LL* | using *FILE*) in *B* end cut

| [gate | total | partial] prio
([all but] *LL* (> [all but] *LL*)+)+
in *B* end prio

| [gate | label] par using
SV (, *SV*)* in *B* ("||" *B*)* end par

R ::=

SV ::= (*L* | _) ("*" (*L* | _))* -> *L*
| (*L* | _) ("||" (*L* | _))* -> *L*

SYNTAX CONVENTIONS

EXP 2.0 satisfies the following precedence and associativity rules:

- LOTOS "hide" gives precedence to infix parallel operators (same as in LOTOS). For instance, the behaviour:
  hide G in "f1.bcg" ||| "f2.bcg"
  is parsed as:
  hide G in ("f1.bcg" ||| "f2.bcg")
- Infix parallel operators associate to the right. For instance, the behaviour:
  "f1.bcg" ||| "f2.bcg" |[G]| "f3.bcg"
  is parsed as:
  "f1.bcg" ||| ("f2.bcg" |[G]| "f3.bcg")

- Postfix operators have precedence over all other operators. For instance, the behaviour:
  "f1.bcg" | "f2.bcg" \ G
  is parsed as:
  "f1.bcg" | ("f2.bcg" \ G)

The syntax of EXP 2.0 is the union of all EBNF productions presented above, with the following exceptions:

- Inside "par" operators, behaviours using infix parallel operators are enclosed in parenthesis or parenthesized operators (hide...end hide, rename...end rename, etc.). For instance, the following behaviour is not syntactically correct:

```
par G in
  "f1.bcg" ||| "f2.bcg"
|| "f3.bcg"
end par
```

  whereas the following behaviours are syntactically correct:

```
par G in
  ("f1.bcg" ||| "f2.bcg")
|| "f3.bcg"
end par
```

```
par G in
  hide H in "f1.bcg" ||| "f2.bcg" end hide
|| "f3.bcg"
end par
```

  Note in particular that the following two behaviours have different semantics:

```
par G in
  "f1.bcg"
|| "f2.bcg"
|| "f3.bcg"
end par
```

  and

```
par G in
  ("f1.bcg" || "f2.bcg")
|| "f3.bcg"
end par
```

In the latter case, the "||" occurring in the parentheses corresponds to the LOTOS or mCRL parallel composition operator, whereas all other occurrences of "||" correspond to the E-LOTOS/LNT parallel symbol.

- Similarly, the "hide" operator must be ended by "end hide" (recommended) or enclosed in parentheses when used inside parenthesized operators.

SEMANTICS OF EXP 2.0

This section describes the semantics of all EXP 2.0 constructs.

LABELLED TRANSITION SYSTEMS

LTS may be the name of a file containing an automaton in one of the AUT (extension **.aut**), BCG (extension **.bcg**), FC2 (extension **.fc2**), or SEQ (extension **.seq**) file formats.

All automaton files must have a known extension. However, the extension may be omitted in the behaviour description if the automaton is in the BCG or in the AUT format. If *LTS* has not a known extension, then **exp.open** will first attempt to open *LTS***.bcg**. If *LTS***.bcg** does not exist in the current directory, **exp.open** will then attempt to open *LTS***.aut**.

Use of the BCG format is recommended since **exp.open** will automatically convert other formats to BCG using the **bcg_io**(LOCAL) tool.

EXP FILE INCLUSION

LTS may also be the name of an EXP file (extension **.exp**) within double quotes. In this case, the contents of this EXP file is simply copied inside the current EXP expression. This allows large expressions to be split into a hierarchy of EXP files.

Example: The expression
  "a.exp" || "b.bcg"
where file "a.exp" contains the expression
  hide A in "a.bcg"
is equivalent to
  hide A in "a.bcg" || "b.bcg"
Note that this is parsed as
  hide A in ("a.bcg" || "b.bcg")
hence parentheses may be required to avoid lexical scoping issues.

GENERALIZED HIDING

[gate | total | partial] hide [all but] *LL* in *B* end hide

| [gate | total | partial] hide using *FILE* in *B* end hide

will hide the labels found in *B* using the given hiding rules. These rules can be specified either as a list of labels (first form), or using an external file *FILE* (second form).

In the first case, **exp.open** builds a temporary file, filled with the given labels. In the second case, the hide file must be provided by the user. No particular extension is required, but the filename must be written between double quotes if it does not have the form *IDF*, *IDF***.hide**, or *IDF***.hid**. See the **caesar_hide_1**(LOCAL) man page for a definition of the hide file format.

The "all but" keywords modify the semantics of the hiding rules: all the labels, except the labels specified in the list of labels, are hidden.

The keywords "total", "partial", and "gate" modify the matching mode, that is the way the hiding rules are interpreted. See the **caesar_hide_1**(LOCAL) man page. See also Section SYNTAX OF EXP 2.0 above, which explains how the gate of a label is recognized. If no matching mode is specified, then the default is "gate", which implements the LOTOS hiding operator extended with regular expressions denoting gates.

For every hiding with "gate" matching, **exp.open** checks whether the gates to be hidden have an appropriate syntax and issues a warning if they appear to contain experiment offers (which is a common mistake for novice users). For instance,

hide "G !1"

will trigger a warning message because of the occurrence of "!1".

Examples:

total hide "G"

hides every label equal to "G",

gate hide "G"

hides every label whose gate is G, e.g., "G !1", "G !2",

gate hide ".*G.*"

hides every label whose gate contains the character G and

partial hide "G"

hides every label whose gate or offers contain the character G.

See the **caesar_hide_1**(LOCAL) man page for more information on matching mode semantics. **regexp**(LOCAL) man page for information about regular expression syntax.

The semantics of LOTOS "hide" is compatible with that of E-LOTOS/LNT, i.e.,

hide *GL* in *B*

is equivalent to

gate hide *GL* in *B* end hide

GENERALIZED RENAMING

[gate | total | single | multiple] rename
L -> *L* (, *L* -> *L*)* in *B* end rename

| [gate | total | single | multiple] rename
using *FILE* in *B* end rename

will rename the labels of *B* using the given renaming rules. These rules can be specified either as a list of items of the form *L* -> *L* (first form), or using an external file *FILE* (second form).

In the first case **exp.open** builds a temporary file, filled with the given substitution rules. In the second case **exp.open** uses the given renaming file. No particular extension is required, but the filename must be written between double quotes if it does not have the form *IDF*, *IDF***.rename**, or *IDF***.ren**. See the **caesar_rename_1**(LOCAL) man page for a definition of the hide file format.

See also Section SYNTAX OF EXP 2.0 above, which explains how the gate of a label is recognized. The keywords "total", "single", "multiple", and "gate" modify the way the left-hand sides of the renaming rules are interpreted, see the **caesar_rename_1**(LOCAL) man page. If no matching mode is specified, then the default is "gate".

For every renaming with "gate" matching, **exp.open** checks whether the gates to be renamed have an appropriate syntax and issues a warning if they appear to contain experiment offers (which is a common mistake for novice users). For instance,

rename "G !1" -> "G !2"

will trigger a warning message because of the occurrence of "!1" in the left-hand side. Note however that
rename "G" -> "G !1"
is correct.

Examples:

total rename "G" -> "H"
renames to "H" every label equal to "G",

gate rename "G" -> "H"
renames to "H" every gate equal to "G", e.g., "G !1" is renamed to "H !1",

gate rename ".*G.*" -> "H"
renames to "H" every gate that contains a G character,

single rename "G" -> "H"
replaces the first occurrence of "G" by "H" in every label whose gate or offers contain a G character,

multiple rename "G" -> "H"
replaces every occurrence of "G" by "H" in every label whose gate or offers contain a G character, and

total rename "\([a-zA-Z0-9]*\) \(!.*\)" -> "\1 !1 \2"
inserts "!1" between every gate (sequence of letters and digits) and its offers (prefixed with !).

See the **caesar_rename_1**(LOCAL) man page for more information on matching mode semantics. See also the **regexp**(LOCAL) man page for more information on regular expressions.

GENERALIZED CUT

[gate | total | partial] cut [all but] *LL* in *B* end cut

| [gate | total | partial] cut using *FILE* in *B* end cut

is a generalization of mCRL encapsulation, on the same principles as the generalized "hide" operator. Instead of being converted into silent transitions as does the "hide" operator, transitions whose label matches the given regular expressions are simply cut off.

Cut files have the following syntax:

AXIOM ::= <blanks> cut <blanks> \n *LABELS*
| <blanks> cut all but <blanks> \n *LABELS*

LABELS ::= (<blanks> *L* <blanks> \n)*

where *L* is a label, and <blanks> is any sequence of spaces, tabulations, carriage returns, newlines, vertical tabulation, or form feeds; these characters are those recognized by the POSIX function isspace(); they are always skipped and ignored.

GENERALIZED PRIORITY

[gate | total | partial] prio
([all but] *LL* (> [all but] *LL*)+)+
in *B* end prio

sets priorities between the transitions of *B*. In each state of *B*, a transition may be executed only if all transitions of higher priority are not ready for execution.

Priorities between transitions (or equivalently, between labels) are defined by a set of priority rules *X1* > ... *Xn*, where each *Xi* has the form [all but] *LLi* and *LL1*, ..., *LLn* are lists of regular expressions denoting gates or labels. The "all but" keywords that may precede some *LLi* means all gates or labels but those matching *LLi*.

Such priority rules define a transitive relation ">>" on labels as follows:

- if *X* > *X'*, the visible label *L* of *B* matches *X*, and the visible label *L'* of *B* matches *X'* then *L* >> *L'*
- if *L* >> *L'* and *L'* >> *L"* then *L* >> *L"*

L >> *L'* means that any transition labeled *L* has priority over any transition labeled *L'* or, equivalently, any transition labeled *L'* yields priority to any transition labeled *L*.

The relation ">>" must be a strict partial order: *B* must not contain any label *L* such that *L* >> *L*. If ">>" is not a strict partial order, **exp.open** will issue an error message and exit.

Beware that the rules *X* > *X'* and *X'* > *X"* (which are equivalent to *X* > *X'* > *X"*) imply *X* > *X"* if and only if some label of *B* matches *X'*. Therefore, to avoid tricky errors, **exp.open** checks that every individual regular expression *L* in *LL1*, ..., *LLn* matches some label of *B*. If not, then **exp.open** will issue a warning.

The optional "gate", "total", and "partial" keywords define the matching mode, in the same way as for the "hide" and "cut" operators. The matching mode by default is "gate".

Examples:

- The following expression:

```
gate prio
   "A.*" > B > all but "A.*", B
in
   "f.bcg"
end prio
```

  defines an LTS in which every transition whose gate starts with the letter "A" has priority over every transition whose gate is "B", which themselves have priority over all other transitions.

- The following expression:

```
partial prio
   "A" > all but "A"
in
   "f.bcg"
end prio
```

  defines an LTS in which every transition whose label contains the letter "A" has priority over every transition whose label does not contain the letter "A" (including hidden transitions).

- The following expression:

```
total prio
   "A" > "B" > "C"
   "D" > "E" > "F"
   "A" > "D"
   "B" > "E"
   "C" > "F"
in
   "f.bcg"
end prio
```

defines an LTS in which A has priority over B, C, D, E, and F, B has priority over C, E, and F, C has priority over F, D has priority over E and F, and E has priority over F.

Note: Strong bisimulation is a congruence for all **exp.open** operators, including "prio". However, branching, observational, and safety equivalences are congruences for all **exp.open** hiding, cutting, renaming, and parallel composition operators, but not for "prio". It should also be noted that tau∗.a equivalence is not a congruence for parallel composition.

GENERALIZED PARALLEL

[gate | label] par [(all | *S*) in]
[*S* ->] *B* ("||" [*S* ->] *B*)+ end par

is an extension of the E-LOTOS/LNT generalized parallel composition operator presented in [Garavel-Sighireanu-99]. It denotes the concurrent execution of parallel behaviours following synchronisation rules expressed using:

- the keyword "all" or the list *S* that follows the keyword "par", called *global synchronisation interface*, and
- the lists *S* that precede the symbols "->", called *local synchronisation interfaces*.

Synchronisation interfaces *S* are lists of synchronisation elements of the form *L* or *L#N*, where *L* is a gate or a label and *N* is a natural number called *synchronisation degree*.

The semantics of synchronisation rules is defined in the following paragraphs.

The "gate" or "label" keywords indicate the *synchronisation mode*, which influences the way synchronisation rules apply to transition labels. We say that a transition matches a synchronisation element of the form *L* or *L#N* in the following cases:

- In "gate" synchronisation mode, a transition matches *L* or *L#N* if the gate of the transition label is *L*. Therefore, for every synchronisation element of the form *L* or *L#N* occurring in a synchronisation interface, *L* must be a gate without offers. See Section SYNTAX OF EXP 2.0 above, which explains how the gate of a transition is extracted from its label.
- In "label" synchronisation mode, a transition matches *L* or *L#N* if the transition label is *L*. Therefore, for every synchronisation element of the form *L* or *L#N* occurring in a synchronisation interface, *L* must be a full label (i.e., a gate possibly followed by offers).

Unlike "hide", "cut", and "rename", regular expressions are not allowed in synchronisation elements. If not specified, the synchronisation mode by default is "gate".

The keyword "all" is a shorthand notation for the global synchronisation set consisting of all synchronisation elements *L* (without degree) such that *L* is the gate (in "gate" synchronisation mode) or the label (in "label" synchronisation mode) of a transition in at least one of the parallel behaviours, except hidden and termination labels.

Synchronisation elements have the following meaning:

- A synchronisation element of the form *L* (without degree) occurring in the global synchronisation interface indicates that all parallel behaviours may synchronise all together on transitions that match *L*.
- A synchronisation element of the form *L* (without degree) occurring in a local synchronisation interface indicates that all parallel behaviours that contain *L* in their synchronisation interface may synchronise all together on transitions that match *L*.
- A synchronisation element of the form *L#N* occurring in the global synchronisation interface indicates that *N* behaviours among the parallel behaviours may synchronise together on transitions that match *L*.
- A synchronisation element of the form *L#N* occurring in a local synchronisation interface indicates

that *N* behaviours among the parallel behaviours that contain *L#N* in their synchronisation interface may synchronise on transitions that match *L*.

- A transition in a parallel behaviour may execute asynchronously if both the global synchronisation interface and the local synchronisation interface of that behaviour do not contain any element of the form *L* or *L#N* such that the transition matches *L*.

Note that both the global synchronisation interface and local synchronisation interfaces may contain several synchronisation elements with same label *L* but different synchronisation degrees. In this case, the corresponding synchronisation rules apply nondeterministically.

Following the above meaning of synchronisation elements, it is possible to prevent the execution of particular transitions matching *L* by using synchronisation elements of the form *L#0*, either in the global synchronisation interface (thus preventing execution of transitions matching *L* in all parallel behaviours) or in local synchronisation interfaces (thus preventing execution of transitions matching *L* in those behaviours containing *L#0* in their interface), provided the (global or local) interface does not contain another occurrence of *L* or *L#N* with *N* a strictly positive number.

Transition synchronisation is a generalization of LOTOS rendezvous: synchronisation requires that all transitions have exactly the same label (i.e., gate and possible offers), which is also the label of the resulting transition.

Note that synchronisation interfaces can neither contain the hidden gate nor the termination gate. Behaviours always synchronise on labels whose gate is the termination gate and never synchronise on hidden transitions.

Before generating the graph module, **exp.open** checks that generalized parallel operators are well-formed, i.e.:

- If the global synchronisation interface contains a synchronisation element of the form *L#N* with $N > 0$, then the parallel composition must contain at least *N* parallel behaviours.
- If the global synchronisation interface contains a synchronisation element of the form *L#0*, then no synchronisation element of the form *L* or *L#N* with same *L* and $N > 0$ should occur in the global synchronisation interface or in any local synchronisation interface.
- If a local synchronisation interface contains a synchronisation element of the form *L* (without degree), then at least two local synchronisation interfaces (this one included) should contain the same synchronisation element. However, if the intention is to explicitly indicate that the behaviour should not synchronise on *L*, then the synchronisation element can be written in the form *L#1*. Synchronisation elements of the form *L#1* can also be used in the global interface to indicate that all transitions matching *L* may execute asynchronously in all behaviours that contain *L*.
- If a local synchronisation interface contains a synchronisation element of the form *L#N* with $N > 0$, then at least *N* local synchronisation interfaces (this one included) should contain the same synchronisation element.
- If a local synchronisation interface contains a synchronisation element of the form *L#0*, then no synchronisation element of the form *L* or *L#N* with same *L* and $N > 0$ should occur in the global synchronisation interface or in the same local synchronisation interface.
- If the global synchronisation interface contains a synchronisation element of the form *L*, then all parallel behaviours must contain a label matching *L*.
- If the global synchronisation interface contains a synchronisation element of the form *L#N*, then at least *N* parallel behaviours must contain a label matching *L*.
- If a local synchronisation interface contains a synchronisation element of the form *L* or *L#N*, then the corresponding parallel behaviours must contain a label matching *L*.

Warnings are issued if those conditions are not satisfied. These checks can be discarded by using the **-nocheck** option of **exp.open**(LOCAL). This is not recommended because errors may lead to

unpredictable behaviours.

Examples:

```
gate par G#2 in
  G4, G1 -> "f1.bcg"
|| G1, G2 -> "f2.bcg"
|| G2, G3 -> "f3.bcg"
|| G3, G4 -> "f4.bcg"
end par
```

represents an automaton such that:

- any two automata among "f1.bcg", "f2.bcg", "f3.bcg", and "f4.bcg" synchronise on transitions whose gate is *G*
- "f1.bcg" and "f2.bcg" synchronise on transitions whose gate is *G1*
- "f2.bcg" and "f3.bcg" synchronise on transitions whose gate is *G2*
- "f3.bcg" and "f4.bcg" synchronise on transitions whose gate is *G3*
- "f4.bcg" and "f1.bcg" synchronise on transitions whose gate is *G4*
- "f1.bcg" executes transitions whose gate is neither *G*, *G1*, *G4*, nor the termination gate, without synchronising with others
- etc.

```
gate par G#2, G#3 in
  "f1.bcg"
|| "f2.bcg"
|| "f3.bcg"
|| "f4.bcg"
end par
```

represents an automaton such that:

- any two or three automata among "f1.bcg", "f2.bcg", "f3.bcg", and "f4.bcg" synchronise on transitions whose gate is *G*
- any of "f1.bcg", "f2.bcg", "f3.bcg", or "f4.bcg" executes transitions whose gate is neither *G* nor the termination gate, without synchronising with others

```
gate par G in
  G1#2, G2 -> "f1.bcg"
|| G1#2 -> "f2.bcg"
|| G1#2 -> "f3.bcg"
|| G2 -> "f4.bcg"
end par
```

represents an automaton such that:

- all automata "f1.bcg", "f2.bcg", "f3.bcg", and "f4.bcg" synchronise on transitions whose gate is *G*
- any two automata among "f1.bcg", "f2.bcg", and "f3.bcg" synchronise on transitions whose gate is *G1*
- "f1.bcg" and "f4.bcg" synchronise on transitions whose gate is *G2*
- "f1.bcg" executes transitions whose gate is neither *G*, *G1*, *G2*, nor the termination gate, without synchronising with others
- etc.

```
gate par
  G#1, G#2 -> "f1.bcg"
|| G#1, G#2 -> "f2.bcg"
|| G#1, G#2 -> "f3.bcg"
end par
```

represents an automaton such that transitions of "f1.bcg", "f2.bcg", and "f3.bcg" whose gate is *G* may either synchronise by pair, or execute asynchronously.

PARALLEL WITH SYNCHRONISATION VECTORS

[gate | label] par using *SV* (, *SV*)* in *B* ("||" *B*)* end par

is parallel composition with *synchronisation vectors*. It denotes the concurrent execution of behaviours following synchronisation rules, expressed using vectors of gates (in "gate" synchronisation mode) or labels (in "label" synchronisation mode).

A synchronisation vector *SV* has the form "*E1* || ... || *En* -> L", or equivalently "*E1* * ... * *En* -> L", where each *Ei* (*i = 1..n*) is either a visible gate without offers (in "gate" synchronisation mode), a visible label, i.e., visible gate with offers (in "label" synchronisation mode), or the special symbol "_" (in both synchronisation modes). Note that gates and labels must not be defined using regular expressions. If it is not specified, the synchronisation mode by default is "gate". See Section SYNTAX OF EXP 2.0 above, which explains how the gate of a label is recognized.

The left-hand side (left of the arrow) of each synchronisation vector must have as many gates (or labels) and "_" symbols as there are behaviours *B* in parallel. A gate (or label) *G* as nth component of a vector left-hand side means that the nth behaviour in the parallel composition can perform a transition whose gate is *G* only if all behaviours can perform simultaneously as specified by the same vector, possibly on different gates, but with same offers (in "gate" synchronisation mode). "_" means that no transition is required for the corresponding behaviour. The resulting transition is labelled with the right-hand side of the vector (in "label" synchronisation mode) or with the right-hand side of the vector, followed by the offers, if any (in "gate" synchronisation mode). The hidden string is forbidden in the left-hand side of a synchronisation vector.

Transitions that do not fit any vector are blocked, except hidden transitions, which always perform asynchronously. In particular, if one wants to let visible transitions perform asynchronously, it is necessary to define vectors whose left-hand side contain only "_" but one visible gate.

Example:

```
gate par using
  G1 || _  || G3 -> G13,
  G1 || G2 || _  -> G12,
  G1 || _  || _  -> G1
  H  || _  || _  -> H
in
  "f1.bcg" || "f2.bcg" || "f3.bcg"
end par
```

represents the automaton defined as follows:

1): if "f1.bcg" can perform a transition whose gate is G1, possibly followed with experiment offers (written *O1...On*), then a transition labelled "G1 *O1... On*" is created in the resulting automaton due to the vector "G1 || _ || _ -> G1"

2): if, additionally to 1), "f3.bcg" can perform a transition labelled "G3 *O1...On*", then a transition labelled "G13 *O1...On*" is created in the resulting automaton due to the vector "G1 || _ || G3 -> G13"

3): if, additionally to 1), "f2.bcg" can perform a transition labelled "G2 *O1...On*", then a transition

labelled "G12 *O1...On*" is created in the resulting automaton due to the vector "G1 || G2 || _ -> G12"

4): if "f1.bcg" can perform a transition whose gate is H, then the same transition is created in the resulting automaton, due to the vector "H || _ || _ -> H"

5): all other transitions of "f1.bcg", "f2.bcg" and "f3.bcg" do not create any transition in the resulting automaton.

LOTOS PARALLEL

B "||" *B*

| *B* "|||" *B*

| *B* "|[" *GL* "]|" *B*

have the same semantics as, respectively:
gate par all in *B* "||" *B* end par
gate par *B* "||" *B* end par
gate par *GL* in *B* "||" *B* end par
Note that in mCRL mode, "||" has a different semantics (see MCRL PARALLEL). Note also that the "|||" operator of CSP has the same semantics as in LOTOS.

CCS PARALLEL

B "|" *B*

is the parallel composition operator of CCS.

Transitions of both behaviours may either perform asynchronously, or may synchronise, resulting in a hidden event.

Synchronisation is possible only if the corresponding transition labels are *co-actions*, i.e., they have the same gate and offers, but one of the gates is prefixed with the co-action prefix. The co-action prefix by default is "'" but can be modified using the **-coaction** option of **exp.open**(LOCAL). Note that (1) the co-action prefix should not contain !, ?, (, space, and tabulation characters, which are used to delimit a gate from its offer, and (2) there should not be any space or tabulation between the co-action prefix and the gate identifier (see Section SYNTAX OF EXP 2.0 above, which explains how the gate of a label is recognized).

For instance, transitions labelled "G" and "'G" may synchronise, as well as transitions labelled "G O" and "'G O". In both cases, the resulting label after synchronisation is the hidden label.

Note that hidden transitions perform asynchronously and that there is no termination label in CCS.

CSP PARALLEL

B "|||" *B*

| *B* "[|" "{" *GL* "}" "|]" *B*

| *B* "[" "{" *GL* "}" "||" "{" *GL* "}" "]" *B*

denote parallel composition in CSP.

- "|||" has the same semantics as in LOTOS (see LOTOS PARALLEL above).
- "[|" "{" *GL* "}" "|]" has the same semantics as LOTOS "|[" *GL* "]|" (see LOTOS PARALLEL above).
- In "*B1* [{*GL1*}||{*GL2*}] *B2*", transitions of *B1* (respectively *B2*) perform asynchronously if their

gate is in *GL1* (respectively *GL2*) but not in *GL2* (respectively *GL1*). Transitions of *B1* and *B2* synchronise if they have the same label, the gate of which is either both in *GL1* and *GL2* or the termination gate (usually written "delta" in CSP). Other transitions are cut off. As usual, hidden transitions perform asynchronously. See Section SYNTAX OF EXP 2.0 above, which explains how the gate of a label is recognized.

MCRL PARALLEL

B "||" *B*

denotes parallel composition in mCRL.

In "*B1* || *B2*", transitions may either perform asynchronously, or they may synchronise following communication rules, that must be defined in the preamble of the EXP file ("comm" keyword). A communication rule has the form "*G1* | *G2* = *G3*", where *G1*, *G2*, and *G3* are gates. Communication rules are commutative, i.e., there is no difference between "*G1* | *G2* = *G3*" and "*G2* | *G1* = *G3*". Such a rule means that if an operand (*B1* or *B2*) may perform a transition whose gate is *G1* and the other operand may perform a transition whose gate is *G2*, and both transitions carry the same experiment offers *O*, then a synchronisation is possible and results in a transition labelled "*G3 O*". See Section SYNTAX OF EXP 2.0 above, which explains how the gate of a label is recognized.

Note that hidden transitions must perform asynchronously, and that there must be at most one communication rule for each pair of gates occurring in the left-hand side of the "=" symbol. For instance, the following is not allowed:
comm G1 | G2 = G3, G2 | G1 = G4 end comm

Note that the reference language must be mCRL in order for the "||" operator to have mCRL semantics instead of LOTOS semantics. If the reference language is different from mCRL, then any "comm" definition is useless and irrelevant.

CCS RESTRICTION

B \ "{" *L* (, *L*)* "}"

| *B* \ *L*

denotes the CCS restriction operator, except when the language option is set to CSP (see Section CSP HIDING below). In the first form, the behaviour *B* is restricted with respect to several labels, whereas in the second form it is restricted with respect to a single label.

CCS restriction is similar to cut (in gate matching mode), except that co-action labels are also affected by the restriction whenever the corresponding action label is affected.

For instance, in "*B* \ *G*", transitions of *B* labelled "*G*", "'*G*" (where "'" stands for the co-action prefix), "*G O*" (where *O* denotes any communication offer), and "'*G* O" are cut off. See Section SYNTAX OF EXP 2.0 above, which explains how the gate of a label is recognized.

Note that labels in the restriction set can be regular expressions denoting gates, but should not have offers and should not start with the co-action prefix.

CSP HIDING

B \ "{" *L* (, *L*)* "}"

| *B* \ *L*

denotes the CSP hiding operator when the language option is set to CSP (otherwise it denotes the CCS restriction operator, see Section CCS RESTRICTION above). Both forms are equivalent to
gate hide *L* (, *L*)* in *B* end hide

gate hide *L* in *B* end hide
respectively.

CCS RELABELING

B "[" *L* / *L* (, *L* / *L*)* "]"

denotes the CCS relabeling operator. It is similar to the renaming operator (in gate matching mode) except that the co-action prefix is preserved by renaming.

For instance, in "*B* [*G1*/*G2*]", the labels "*G2*", "'*G2*" (where "'" stands for the co-action prefix), "*G2 O*" (where *O* denotes any communication offer), and "'*G2 O*" are renamed respectively into "*G1*", "'*G1*", "*G1 O*", and "'*G1 O*". See Section SYNTAX OF EXP 2.0 above, which explains how the gate of a label is recognized.

Note that labels can be regular expressions denoting gates, but should not have offers and should not start with the co-action prefix.

CSP RELABELING

B "[[" *L* <- *L* (, *L* <- *L*)* "]]"

denotes the CSP relabeling operator.

The behaviour
B [[G1 <- G2, ..., G2n+1 <- G2n+2]]
is equivalent to
gate rename G1 -> G2, ..., G2n+1 -> G2n+2 in B

(Note the opposite direction of the arrows.)

Note that, as a consequence, labels can be regular expressions denoting gates, but should not have offers.

LANGUAGE PARAMETERS

There are two means to select a reference language:

- Using **exp.open**(LOCAL) with a command line option (**-lotos**, **-elotos**, **-ccs**, **-csp**, or **-mcrl**), or
- Specifying the language in the EXP file header ("lotos behaviour", "elotos behaviour", etc.). Note that "behaviour" or "behavior" alone is equivalent to "lotos behaviour" when no language option is set. The absence of the "behaviour" or "behavior" keyword means that no reference language is set.

If the EXP file header and the command line option specify different reference languages, then a warning is issued. It is not allowed to specify more than one language option on the command line.

The following table summarizes the default parameter values according to reference languages. Note that these values (in particular lines labelled none and LOTOS) are compatible with **exp.open** version 1.0.

language	hidden event	termin. gate	case sensitivity	sem. of \	sem. of ||
none	"i"	"exit"	sensitive	CCS	LOTOS
LOTOS	"i"	"exit"	insensitive	CCS	LOTOS
E-LOTOS*	"i"	"exit"	sensitive	CCS	LOTOS
CCS	"t"	N/A	insensitive	CCS	LOTOS
CSP	"t"	"delta"	insensitive	CSP	LOTOS
mCRL	"tau"	N/A	sensitive	CCS	mCRL

where N/A stands for *Not Applicable*.

* The parameters associated to E-LOTOS are subject to modifications in the future, depending on implementation choices still to be made.

Note that the hidden string (written either "i", "t", "tau", or anything else) should not be followed by communication offers, whereas the termination string may.

HOW TO CREATE AN EXP FILE

At present, EXP files can be either written by hand or produced automatically by **svl**(LOCAL) from SVL scripts containing some parallel composition operator.

HOW TO READ AN EXP FILE

At present, there is one single CADP tool, **exp.open**(LOCAL), that reads and processes EXP files.

BIBLIOGRAPHY

[Berthomieu-Ribet-Vernadat-04]

B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA - Construction of Abstract State Spaces for Petri Nets and Time Petri Nets. In International Journal of Production Research, Vol. 42, No 14, July 2004.

[Best-Grahlmann-98]

Eike Best and Bernd Grahlmann. "PEP Documentation and User Guide." http://parsys.informatik.uni-oldenburg.de/~pep/paper.html. 1998."

[Bouali-Ressouche-Roy-deSimone-96]

Amar Bouali, Annie Ressouche, Valerie Roy, and Robert de Simone. The Fc2Tools set: a Toolset for the Verification of Concurrent Systems. In R. Alur and T.A. Henzinger, editors, Proceedings of the 8th Conference on Computer-Aided Verification (New Brunswick, New Jersey, USA). Lecture Notes in Computer Science volume 1102, Springer-Verlag, 1996.

[Brookes-Hoare-Roscoe-84]

S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. "A Theory of Communicating Sequential Processes." In Journal of the ACM, vol. 31, number 3, pages 560-599. ACM, 1984.

[Garavel-15-a]

Hubert Garavel. "Nested-Unit Petri Nets: A Structural Means to Increase Efficiency and Scalability of Verification on Elementary Nets." In R. Devillers and A. Valmari, editors, Proceedings of the 36th International Conference on Application and Theory of Petri Nets and Concurrency (Brussels, Belgium). Lecture Notes in Computer Science volume 9115, Springer-Verlag, 2015. Available from http://cadp.inria.fr/publications/Garavel-15-a.html

[Garavel-Sighireanu-99]
Hubert Garavel and Mihaela Sighireanu. "A Graphical Parallel Composition Operator for Process Algebras." In J. Wu, Q. Gao, and S.T. Chanson, editors, Proceedings of the Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification FORTE/PSTV'99 (Beijing, China). Kluwer Academic Publishers, 1999. Available from http://cadp.inria.fr/publications/Garavel-Sighireanu-99.html

[Groote-Ponse-90]
J.F. Groote and A. Ponse. "The syntax and semantics of mCRL." In A. Ponse, C. Verhoef and S.F.M. van Vlijmen, editors, Algebra of Communicating Processes '94, Workshops in Computing Series, Springer-Verlag, pp. 26-62, 1995. Also appeared as: Technical Report CS-R9076, CWI, Amsterdam, 1990.

[ISO-89]
ISO/IEC. "LOTOS --- A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour." International Organization for Standardization --- Information Processing Systems --- Open Systems Interconnection. International Standard number 8807. Geneva, September 1989.

[ISO-01]
ISO/IEC. "Enhancements to LOTOS (E-LOTOS)." International Organization for Standardization --- Information Technology. International Standard number 15437:2001. Geneva, September 2001.

[Lang-05]
Frederic Lang. "EXP.OPEN 2.0: A Flexible Tool Integrating Partial Order, Compositional, and On-the-fly Verification Methods." In J. van de Pol, J. Romijn and G. Smith, editors, Proceedings of the 5th International Conference on Integrated Formal Methods IFM'2005 (Eindhoven, The Netherlands). Lecture Notes in Computer Science volume 3771, Springer-Verlag, 2005. Available from http://cadp.inria.fr/publications/Lang-05.html

[Lang-06]
Frederic Lang. "Refined Interfaces for Compositional Verification." In E. Najm, J.-F. Pradat-Peyre and V. Viguie Donzeau-Gouge, editors, Proceedings of the 26th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems FORTE'2006 (Paris, France). Lecture Notes in Computer Science volume 4229, Springer-Verlag, 2006. Available from http://cadp.inria.fr/publications/Lang-06.html

[Milner-89]
Robin Milner. "Communication and Concurrency." Prentice-Hall, 1989.

[Pace-Lang-Mateescu-03]
Gordon Pace, Frederic Lang, and Radu Mateescu. "Calculating tau-confluence compositionally."

In W.A. Hunt Jr. and F. Somenzi, editors, 15th Computer-Aided Verification conference (CAV 2003), Lecture Notes in Computer Science volume 2725, Springer-Verlag, 2003. Available from http://cadp.inria.fr/publications/Pace-Lang-Mateescu-03.html

SEE ALSO

aldebaran(LOCAL), **aut**(LOCAL), **bcg**(LOCAL), **bcg_io**(LOCAL), **caesar_hide_1**(LOCAL), **caesar_rename_1**(LOCAL), **exp.open**(LOCAL), **lotos.open**(LOCAL), **projector**(LOCAL), **regexp**(LOCAL), **seq**(LOCAL), **svl**(LOCAL)

Additional information is available from the CADP Web page located at http://cadp.inria.fr

Directives for installation are given in files **$CADP/INSTALLATION_*.**

Recent changes and improvements to this software are reported and commented in file **$CADP/HISTORY.**

BUGS

Please report bugs to cadp@inria.fr