Process Library Implementation Notes

Wieger Wesselink

March 20, 2023

1 Process Library Implementation Notes

1.1 Processes

Process expressions in mCRL2 are expressions built according to the following syntax:

expression	C++ equivalent	ATerm grammar
$a(e)$	action(a,e)	Action
$P(e)$	process(P,e)	Process
$P(d := e)$	process_assignment(P,$d := e$)	ProcessAssignment
δ	delta()	Delta
τ	tau()	Tau
$\sum_d x$	sum(d,x)	Sum
$\partial_B(x)$	block(B,x)	Block
$\tau_B(x)$	hide(B,x)	Hide
$\rho_R(x)$	rename(R,x)	Rename
$\Gamma_C(x)$	comm(C,x)	Comm
$\nabla_V(x)$	allow(V,x)	Allow
$x \mid y$	sync(x,y)	Sync
$x ^{\varsigma}\, t$	at_time(x,t)	AtTime
$x \cdot y$	seq(x,y)	Seq
$c \rightarrow x$	if_then(c,x)	IfThen
$c \rightarrow x \diamond y$	if_then_else(c,x,y)	IfThenElse
$x \ll y$	binit(x,y)	BInit
$x \parallel y$	merge(x,y)	Merge
$x \mathbin{⌊\!⌊} y$	lmerge(x,y)	LMerge
$x + y$	choice(x,y)	Choice

where the types of the symbols are as follows:

a, b	strings (action names)
P	a process identifier
e	a sequence of data expressions
d	a sequence of data variables
B	a set of strings (action names)
R	a sequence of rename expressions
C	a sequence of communication expressions
V	a sequence of multi actions
t	a data expression of type real
x, y	process expressions
c	a data expression of type bool

A rename expression is of the form $a \rightarrow b$, with a and b action names. A multi action is of the form $a_1 \mid \cdots \mid a_n$, with a_i actions. A communication expression is of the form $b_1 \mid \cdots \mid b_n \rightarrow b$, with b and b_i action names.

1.1.1 Restrictions

A multi action is a multi set of actions. The left hand sides of the communication expressions in C must be unique. Also the left hand sides of the rename expressions in R must be unique.

1.1.2 Linear process expressions

Linear process expressions are a subset of process expressions satisfying the following grammar:

```
<linear process expression> ::= choice(<linear process expression>, <linear process expression>)
                              | <summand>

<summand>                   ::= sum(<variables>, <alternative>)
                              | <conditional action prefix>
                              | <conditional deadlock>

<conditional action prefix> ::= if_then(<condition>, <action prefix>)
                              | <action prefix>

<action prefix>             ::= seq(<timed multiaction>, <process reference>)
                              | <timed multiaction>

<timed multiaction>         ::= at_time(<multiaction>, <time stamp>)
                              | <multiaction>

<multiaction>               ::= tau()
                              | <action>
                              | sync(<multiaction>, <multiaction>)

<conditional deadlock>      ::= if_then(<condition>, <timed deadlock>)
                              | <timed deadlock>

<timed deadlock>            ::= delta()
                              | at_time(delta(), <time stamp>)

<process reference>         ::= process(<process identifier>, <data expressions>)
                              | process_assignment(<process identifier>, <data assignments>)
```

1.2 Guarded process expressions

We define the predicate *is_guarded* for process expressions as follows: $is\_guarded(p) = is\_guarded(p, \emptyset)$

$$
\begin{array}{lcl}
is\_guarded(a(e), W) & = & true \\
is\_guarded(\delta, W) & = & true \\
is\_guarded(\tau, W) & = & true \\
is\_guarded(P(e), W) & = & \left\{ \begin{array}{ll} false & \text{if } P \in W \\ is\_guarded(p, W \cup \{P\}) & \text{if } P \notin W \end{array} \right. \\
 & & \text{where } P(d) = p \text{ is the equation corresponding to } P(e) \\
is\_guarded(p + q, W) & = & is\_guarded(p, W) \wedge is\_guarded(q, W) \\
is\_guarded(p \cdot q, W) & = & is\_guarded(p, W) \\
is\_guarded(c \to p, W) & = & is\_guarded(p, W) \\
is\_guarded(c \to p \diamond q, W) & = & is\_guarded(p, W) \wedge is\_guarded(q, W) \\
is\_guarded(\Sigma_{d:D}\ p, W) & = & is\_guarded(p, W) \\
is\_guarded(p \,^{\varsigma}\, t, W) & = & is\_guarded(p, W) \\
is\_guarded(p \ll q, W) & = & is\_guarded(p, W) \\
is\_guarded(p \parallel q, W) & = & is\_guarded(p, W) \wedge is\_guarded(q, W) \\
is\_guarded(p \mathbin{\lfloor\!\lfloor} q, W) & = & is\_guarded(p, W) \\
is\_guarded(p \mid q, W) & = & is\_guarded(p, W) \wedge is\_guarded(q, W) \\
is\_guarded(\rho_R(p), W) & = & is\_guarded(p, W) \\
is\_guarded(\partial_B(p), W) & = & is\_guarded(p, W) \\
is\_guarded(\tau_I(p), W) & = & is\_guarded(p, W) \\
is\_guarded(\Gamma_C(p), W) & = & is\_guarded(p, W) \\
is\_guarded(\nabla_V(p), W) & = & is\_guarded(p, W)
\end{array}
$$

N.B. This specification assumes that process names are unique. In mCRL2 process names can be overloaded, therefore in the implemenation W contains *process identifiers* (i.e. both the process name and the sorts of the arguments) instead of process names.

1.3 Alphabet reduction

Alphabet reduction is a preprocessing step for linearization. It is a transformation on process expressions that preserves branching bisimulation.

1.3.1 Notations

In this text action names are represented using $a, b, \ldots$ and multi action names using $\alpha, \beta, \ldots$ So in general we have $\alpha = a_1 \mid \ldots \mid a_n$. In alphabet reduction data parameters play a minor role, therefore we choose a notation in which data parameters are omitted. We use the abbreviation $\overline{a} = a(e_1, \ldots, e_n)$ to denote an action, and $\overline{\alpha} = \overline{a_1} \mid \ldots \mid \overline{a_n}$ to denote a multi action, where $e_1, \ldots, e_n$ are data expressions.Note that a multi action is a multiset (or bag) of actions and a multi action name is a multiset of names. We write $\alpha\beta$ as shorthand for $\alpha \cup \beta$ and $a\beta$ for $\{a\} \cup \beta$. Sets of multi action names are represented using $A, A_1, A_2, \ldots$ A communication C maps multi action names to action names, and is denoted as $\{\alpha_1 \to a_1, \ldots, \alpha_n \to a_n\}$. A renaming R is a substitution on action names, and is denoted as $R = \{a_1 \to b_1, \ldots, a_n \to b_n\}$. A block set B is a set of action names. A hide set I is a set of action names.

1.3.2 Definitions

We define multi actions $\overline{\alpha}$ using the following grammar:

$$\overline{\alpha} := \overline{a} \mathrel{\scriptstyle|} \overline{\alpha} \mid \overline{a},$$

where $\overline{a}$ is an action, and where $\scriptstyle|$ is used to distinguish alternatives.

We define pCRL terms p using the following grammar:

$$p ::= \overline{a} \mathrel{\scriptstyle|} P \mathrel{\scriptstyle|} \delta \mathrel{\scriptstyle|} \tau \mathrel{\scriptstyle|} p + p \mathrel{\scriptstyle|} p \cdot p \mathrel{\scriptstyle|} c \to p \mathrel{\scriptstyle|} c \to p \diamond p \mathrel{\scriptstyle|} \Sigma_{d:D} p \mathrel{\scriptstyle|} p \mathbin{^c} t \mathrel{\scriptstyle|} p \ll p,$$

and parallel mCRL terms q using the following grammar:

$$q ::= p \mathrel{\scriptstyle|} q \parallel q \mathrel{\scriptstyle|} q \mathbin{⌊\!⌊} q \mathrel{\scriptstyle|} q \mid q \mathrel{\scriptstyle|} \rho_R(q) \mathrel{\scriptstyle|} \partial_B(q) \mathrel{\scriptstyle|} \tau_I(q) \mathrel{\scriptstyle|} \Gamma_C(q) \mathrel{\scriptstyle|} \nabla_V(q).$$

Remark 1 Note that there is an unfortunate overload of the |-operator in both multi actions and process expressions. This has consequences for the implementation, since it there is no clean distinction between parallel and non-parallel operators.

Remark 2 The mCRL2 language also has a construct $P(d_{i_1} = e_{i_1}, \ldots, d_{i_k} = e_{i_k})$, but this is just a shorthand notation. Therefore we will ignore it in this text.

1.3.3 Alphabet operations

Let A, A_1 and A_2 be sets of multi action names. Then we define

$$\begin{array}{lcl} A^{\subseteq} & = & \{\alpha \mid \exists \beta . \alpha\beta \in A\} \\ A_1 A_2 & = & \{\alpha\beta \mid \alpha \in A_1 \text{ and } \beta \in A_2\} \\ A_1 \leftarrow\!\!\!\prec A_2 & = & \{\alpha \mid \exists \beta . \alpha\beta \in A_1 \text{ and } \beta \in A_2\} \end{array}$$

Note that β can take the value τ in the definition of $A_1 \leftarrow\!\!\!\prec A_2$, which implies $A_1 \subset A_1 \leftarrow\!\!\!\prec A_2$. The set $A^{\subseteq}$ has an exponential size, so whenever possible it should not be computed explicitly.

Let C be a communication set, then we define

$$\begin{array}{lcl} C(A) & = & \cup_{\alpha \in A} \textsc{Comm}(C, \alpha) \\ C^{-1}(A) & = & \cup_{\alpha \in A} \textsc{CommInverse}(C, \alpha) \\ \mathit{filter}_{\nabla}(C, A) & = & \{\gamma \to c \in C \mid \exists_{\alpha \in A} . \gamma \subset \alpha\} \end{array}$$

where $\textsc{Comm}$ and $\textsc{CommInverse}$ are defined using pseudo code as follows:

```
Comm(C, α)
R := {α}
for γ → c ∈ C do
    if ∃β.α = βγ then R := R ∪ Comm(C, βc)
return R
```

```
CommInverse(C, α1,α2)
R := {α1α2}
for γ → c ∈ C do
    if ∃β.α1 = βc then R := R ∪ CommInverse(C, β,α2γ)
return R
```

Note that $C^{-1}(\alpha) = \textsc{CommInverse}(C,\ \alpha,\tau)$.

Let R be a rename set, then we define

$$\begin{array}{lcl} R(\alpha) & = & \{R(\alpha_i) \mid \alpha_i \in \alpha\} \\ R^{-1}(\alpha) & = & \{\beta \mid R(\beta) = \alpha\} \\ R(A) & = & \{R(\alpha) \mid \alpha \in A\} \\ R^{-1}(A) & = & \{R^{-1}(\alpha) \mid \alpha \in A\} \end{array}$$

Let I be a hide set, then we define

$$\begin{array}{lcl} \tau_I(A) & = & \{\beta \mid \exists_{\alpha \in A, \gamma \in I^*}.\alpha = \beta\gamma \wedge \beta \cap I = \emptyset\} \\ \tau_I^{-1}(A) & = & \partial_I(A)I^* \end{array}$$

Let B be a block set, then we define

$$\partial_B(A) = \{\alpha \in A \mid \alpha \cap B = \emptyset\}$$

We define a mapping *act* that extracts the individual action names of a set of multi action names:

$$\begin{array}{lcl} act\,(a_1 \mid \ldots \mid a_n) & = & \{a_1 \mid \ldots \mid a_n\} \\ act\,(A) & = & \bigcup_{\alpha \in A} act\,(\alpha) \end{array}$$

1.3.4 The mapping α

We define the mapping α as follows. The value $\alpha(p, \emptyset)$ is an over approximation of the alphabet of process expression p.

$$\begin{array}{lcl} \alpha(\overline{a}, W) & = & \{a\} \\ \alpha(P, W) & = & \begin{cases} \emptyset & \text{if } P \in W \\ \alpha(p, W \cup \{P\}) & \text{if } P \notin W, \end{cases} \\ & & \text{where } P = p \text{ is the equation of } P \\ \alpha(\delta, W) & = & \emptyset \\ \alpha(\tau, W) & = & \{\tau\} \\ \alpha(p + q, W) & = & \alpha(p, W) \cup \alpha(q, W) \\ \alpha(p \cdot q, W) & = & \alpha(p, W) \cup \alpha(q, W) \\ \alpha(c \to p, W) & = & \alpha(p, W) \\ \alpha(c \to p \diamond q, W) & = & \alpha(p, W) \cup \alpha(q, W) \\ \alpha(\Sigma_{d:D} p, W) & = & \alpha(p, W) \\ \alpha(p \,{}^{c}\, t, W) & = & \alpha(p, W) \\ \alpha(p \ll q, W) & = & \alpha(p, W) \cup \alpha(q, W) \\ \alpha(p \parallel q, W) & = & \alpha(p, W) \cup \alpha(q, W) \cup \alpha(p, W)\alpha(q, W) \\ \alpha(p \mathbin{|\!\lfloor} q, W) & = & \alpha(p, W) \cup \alpha(q, W) \cup \alpha(p, W)\alpha(q, W) \\ \alpha(p \mid q, W) & = & \alpha(p, W)\alpha(q, W) \\ \alpha(\rho_R(p), W) & = & R(\alpha(p, W)) \\ \alpha(\partial_B(p), W) & = & \partial_B(\alpha(p, W)) \\ \alpha(\tau_I(p), W) & = & \tau_I(\alpha(p, W)) \\ \alpha(\Gamma_C(p), W) & = & C(\alpha(p, W)) \\ \alpha(\nabla_V(p), W) & = & \alpha(p, W) \cap (V \cup \{\tau\}) \end{array}$$

Example 1

If $C = \{a \mid b \to c\}$, then $\alpha(\Gamma_C(a(1) \mid b(2))) = \{a, b, c, a \mid b\}$. Note that the action c does not occur in the transition system of this process expression.

Example 2 In the computation of $\{a_1, a_2, \ldots, a_{20}\} \cap \alpha\,(a_1 \parallel a_2 \parallel \ldots \parallel a_{20})$ the above mentioned optimization is really needed.

1.3.5 Computation of the alphabet

When computing $A \cap \alpha(p, W)$ for some multi action name set A, it may be beneficial to apply an optimization. This is done to keep intermediate expressions small. We introduce $\alpha(p, W, A) = A \cap \alpha(p, W)$, and define it as follows:

$$
\begin{array}{lcl}
\alpha(\overline{a}, W, A) & = & \begin{cases} \{a\} & \text{if } a \in A \\ \emptyset & \text{if } a \notin A \end{cases} \\
\alpha(P, W, A) & = & \begin{cases} \emptyset & \text{if } P \in W \\ \alpha(p, W \cup \{P\}, A) & \text{if } P \notin W, \end{cases} \\
 & & \text{where } P = p \text{ is the equation of } P \\
\alpha(p + q, W, A) & = & \alpha(p, W, A) \cup \alpha(q, W, A) \\
\alpha(p \cdot q, W, A) & = & \alpha(p, W, A) \cup \alpha(q, W, A) \\
\alpha(c \to p, W, A) & = & \alpha(p, W, A) \\
\alpha(c \to p \diamond q, W, A) & = & \alpha(p, W, A) \cup \alpha(q, W, A) \\
\alpha(\Sigma_{d:D} p, W, A) & = & \alpha(p, W, A) \\
\alpha(p \,{}^{\varsigma}\, t, W, A) & = & \alpha(p, W, A) \\
\alpha(p \ll q, W, A) & = & \alpha(p, W, A) \cup \alpha(q, W, A) \\
\alpha(p \parallel q, W, A) & = & \alpha(p, W, A) \cup \alpha(q, W, A) \cup \alpha(p, W, A^{\subseteq})\alpha(q, W, A^{\subseteq}) \\
\alpha(p \mathbin{\parallel\!\!\lfloor} q, W, A) & = & \alpha(p, W, A) \cup \alpha(q, W, A) \cup \alpha(p, W, A^{\subseteq})\alpha(q, W, A^{\subseteq}) \\
\alpha(p \mid q, W, A) & = & \alpha(p, W, A^{\subseteq})\alpha(q, W, A^{\subseteq})
\end{array}
$$

1.3.6 More efficient computation of the alphabet

The computation of $\alpha(p, W, A)$ can be done more efficiently. We define the function $proc(p, W)$ as follows:

$$
\begin{array}{lcl}
proc(\overline{a}, W) & = & \emptyset \\
proc(P, W) & = & \begin{cases} \emptyset & \text{if } P \in W \\ \{P\} \cup proc(p, W) & \text{if } P \notin W \end{cases} \\
proc(p + q, W) & = & proc(p, W) \cup proc(q, W) \\
proc(p \cdot q, W) & = & proc(p, W) \cup proc(q, W) \\
proc(c \to p, W) & = & proc(p, W) \\
proc(c \to p \diamond q, W) & = & proc(p, W) \cup proc(q, W) \\
proc(\Sigma_{d:D} p, W) & = & proc(p, W) \\
proc(p \,{}^{\varsigma}\, t, W) & = & proc(p, W)
\end{array}
$$

Using this function we can change the computation of $\alpha(p, W, A)$ at three places:

$$
\begin{array}{lcl}
\alpha(p + q, W, A) & = & \alpha(p, W, A) \cup \alpha(q, W \cup proc(p, W), A) \\
\alpha(p \cdot q, W, A) & = & \alpha(p, W, A) \cup \alpha(q, W \cup proc(p, W), A) \\
\alpha(c \to p \diamond q, W, A) & = & \alpha(p, W, A) \cup \alpha(q, W \cup proc(p, W), A)
\end{array}
$$

Note that the value $proc(p, W)$ can be computed on the fly during the computation of $\alpha(p, W, A)$.

1.3.7 Bounded alphabet

In practice one often wants to compute $\alpha(p, A) = \alpha(\nabla_A(p))$. This can be computed more efficiently as follows:

$$
\begin{array}{lcl}
\alpha(\overline{a}, A) & = & \begin{cases} \{a\} & \text{if } a \in A \\ \emptyset & \text{if } a \notin A \end{cases} \\
\alpha(P, A) & = & \alpha(p, A), \text{ where } P = p \text{ is the equation of } P \\
\alpha(p + q, A) & = & \alpha(p, A) \cup \alpha(q, A) \\
\alpha(p \cdot q, A) & = & \alpha(p, A) \cup \alpha(q, A) \\
\alpha(c \rightarrow p, A) & = & \alpha(p, A) \\
\alpha(c \rightarrow p \diamond q, A) & = & \alpha(p, A) \cup \alpha(q, A) \\
\alpha(\Sigma_{d:D} p, A) & = & \alpha(p, A) \\
\alpha(p ^{\varsigma} t, A) & = & \alpha(p, A) \\
\alpha(p \ll q, A) & = & \alpha(p, A) \cup \alpha(q, A) \\
\alpha(p \parallel q, A) & = & \alpha(p, A) \cup \alpha(q, A) \cup \alpha(p, A^{\subseteq}) \alpha(q, A \leftarrowtail \alpha(p, A^{\subseteq})) \\
\alpha(p \,\|\!\_\, q, A) & = & \alpha(p, A) \cup \alpha(q, A) \cup \alpha(p, A^{\subseteq}) \alpha(q, A \leftarrowtail \alpha(p, A^{\subseteq})) \\
\alpha(p \mid q, A) & = & \alpha(p, A^{\subseteq}) \alpha(q, A \leftarrowtail \alpha(p, A^{\subseteq})) \\
\alpha(\rho_R(p), A) & = & R(\alpha(p, R^{-1}(A))) \\
\alpha(\partial_B(p), A) & = & \alpha(p, \partial_B(A)) \\
\alpha(\tau_I(p), A) & = & \tau_I(\alpha(p, \tau_I^{-1}(A))) \\
\alpha(\Gamma_C(p), A) & = & C(\alpha(p, C^{-1}(A))) \\
\alpha(\nabla_V(p), A) & = & \alpha(p, A \cap V))
\end{array}
$$

1.3.8 The mappings *push*, $push_\nabla$ and $push_\partial$

We define mappings *push*, $push_\nabla$ and $push_\partial$ such that $push(p)$ is bisimulation equivalent to p, $push_\nabla(A, p)$ is bisimulation equivalent to $\nabla_A(p)$, and $push_\partial(B, p)$ is bisimulation equivalent to $\partial_B(p)$. The goal of these mappings is to push allow and block expressions deeply inside process expressions. It is important to know that an allow set A in the expression $\nabla_A(p)$ implicitly contains the empty multi action τ. Let $\mathcal{E}$ $= \{P_1(d) = p_1, \ldots, P_n(d) = p_n\}$ be a sequence of process equations.

$$
\begin{array}{lcl}
push(p) & = & p \text{ if } p \text{ is a pCRL expression} \\
push(p \parallel q) & = & push\,(p) \parallel push\,(q) \\
push(p \,\|\!\_\, q) & = & push\,(p) \,\|\!\_\, push\,(q) \\
push(p \mid q) & = & push\,(p) \mid push\,(q) \\
push(\rho_R(p)) & = & \rho_R(push\,(p)) \\
push(\partial_B(p)) & = & push_\partial(B, p) \\
push(\tau_I(p)) & = & \tau_I(push\,(p)) \\
push(\Gamma_C(p)) & = & \Gamma_C\,(push\,(p)) \\
push(\nabla_V(p)) & = & push_\nabla(V, p)
\end{array}
$$

We assume that $P^{\nabla}_{A,e}$ is a unique name for every $P \in \{P_1, \ldots, P_n\}$, multi action name set A and sequence of data expressions e.

$$
\begin{array}{lcl}
push_{\nabla}\,(A, \overline{a}) & = & \left\{ \begin{array}{ll} \overline{a} & \text{if } N(\overline{a}) \in A \\ \delta & \text{otherwise} \end{array} \right. \\
push_{\nabla}\,(A, P\,(e)) & = & \begin{array}{l} P^{\nabla}_{A}(e), \text{ where } P(d) = p \text{ is the equation of } P, \text{ and} \\ \text{where } P^{\nabla}_{A}(d) = push_{\nabla}\,(A, p) \text{ is a new equation} \end{array} \\
push_{\nabla}\,(A, \delta) & = & \delta \\
push_{\nabla}\,(A, \tau) & = & \tau \\
push_{\nabla}\,(A, p + q) & = & \nabla_A(p + q) \\
push_{\nabla}\,(A, p \cdot q) & = & \nabla_A(p \cdot q) \\
push_{\nabla}\,(A, c \to p) & = & \nabla_A\,(c \to p) \\
push_{\nabla}\,(A, c \to p \diamond q) & = & \nabla_A\,(c \to p \diamond q) \\
push_{\nabla}\,(A, \Sigma_{d:D} p) & = & \nabla_A\,(\Sigma_{d:D} p) \\
push_{\nabla}\,(A, p \,^{\varsigma}\, t) & = & \nabla_A\,(p \,^{\varsigma}\, t) \\
push_{\nabla}\,(A, p \ll q) & = & \nabla_A\,(p \ll q) \\
push_{\nabla}(A, p \parallel q) & = & \nabla_{\mathsf{A}}(A, p' \parallel q') \text{ where} \left\{ \begin{array}{lcl} p' & = & push_{\nabla}(A^{\subseteq}, p) \\ q' & = & push_{\nabla}(A \longleftarrow \alpha(p'), q) \end{array} \right. \\
push_{\nabla}(A, p \,⌊\!⌊\, q) & = & \nabla_{\mathsf{A}}(A, p' \,⌊\!⌊\, q') \text{ where} \left\{ \begin{array}{lcl} p' & = & push_{\nabla}(A^{\subseteq}, p) \\ q' & = & push_{\nabla}(A \longleftarrow \alpha(p'), q) \end{array} \right. \\
push_{\nabla}(A, p \mid q) & = & \nabla_{\mathsf{A}}(A, p' \mid q') \text{ where} \left\{ \begin{array}{lcl} p' & = & push_{\nabla}(A^{\subseteq}, p) \\ q' & = & push_{\nabla}(A \longleftarrow \alpha(p'), q) \end{array} \right. \\
push_{\nabla}(A, \rho_R(p)) & = & \rho_R(p') \text{ where } p' = push_{\nabla}(R^{-1}(A), p) \\
push_{\nabla}(A, \partial_B(p)) & = & push_{\nabla}(\partial_B(A), p) \\
push_{\nabla}(A, \tau_I(p)) & = & \tau_I(p') \text{ where } p' = push_{\nabla}(\tau_I^{-1}(A), p) \\
push_{\nabla}(A, \Gamma_C(p)) & = & \mathsf{allow}(A, \Gamma_C(p')) \text{ where } p' = push_{\nabla}(C^{-1}(A), p) \\
push_{\nabla}(A, \nabla_V(p)) & = & push_{\nabla}(A \cap V, p),
\end{array}
$$

Optimizations During the computation of $push_{\nabla}$ the following optimizations are applied in the right hand side of each equation:

$$
\begin{array}{lcl}
\nabla_A(p) & = & \left\{ \begin{array}{ll} p & \text{if } (A \cup \{\tau\}) \cap \alpha(p) = \alpha(p) \\ \nabla_{A \cap \alpha(p)}(p) & \text{otherwise} \end{array} \right. \\
\nabla_{\emptyset}(p) & = & \left\{ \begin{array}{ll} \tau & \text{if } p = \tau \\ \delta & \text{otherwise} \end{array} \right. \\
\Gamma_C(p) & = & \Gamma_{filter_{\nabla}(C, \alpha(p))}(p) \\
\delta \mid \delta & = & \delta \\
\delta \parallel \delta & = & \delta
\end{array}
$$

For non pCRL expression the alphabet $\alpha(p)$ is computed on the fly during the computation of $push_{\nabla}\,(A, p)$.

Example 1 Let $P = (a+b) \cdot P$. Then $push_{\nabla}\,(\{a\}, P, \emptyset) = P'$, with $P' = push_{\nabla}\,(\{a\}, (a + b) \cdot P, \{(P, \{a\}, P')\}) = push_{\nabla}\,(\{a\}, (a + b), \{(P, \{a\}, P')\}) \cdot push_{\nabla}\,(\{a\}, P, \{(P, \{a\}, P')\}) = \cdots = a \cdot P'$.

Example 2 Let $P = a \cdot \nabla_{\{a\}}(P)$. Then $push_{\nabla}\,(\{a\}, P, \emptyset) = P'$, with $P' = push_{\nabla}\left(\{a\}, a \cdot \nabla_{\{a\}}(P), \{(P, \{a\}, P')\}\right) = push_{\nabla}\,(\{a\}, a, \{(P, \{a\}, P')\}) \cdot push_{\nabla}\left(\{a\}, \nabla_{\{a\}}(P), \{(P, \{a\}, P')\}\right) = \cdots = a \cdot P'$.

We assume that $P^{\partial}_{A,e}$ is a unique name for every $P \in \{P_1, \ldots, P_n\}$, multi action name set A and sequence of data expressions e.

$$
\begin{array}{lcl}
push_\partial(B,\overline{a}) & = & \left\{\begin{array}{ll}\overline{a} & \text{if } N(\overline{a})\cap B=\emptyset\\ \delta & \text{otherwise}\end{array}\right.\\
push_\partial(B,P(e)) & = & \begin{array}{l}P^\partial_{B,e}(e)\\ \text{where } P(d)=p \text{ is the equation of } P\text{, and}\\ \text{where } P^\partial_{B,e}(d)=push_\partial\,(B,p) \text{ is a new equation}\end{array}\\
push_\partial(B,\delta) & = & \delta\\
push_\partial(B,\tau) & = & \tau\\
push_\partial(B,p+q) & = & push_\partial(B,p)+push_\partial(B,q)\\
push_\partial(B,p\cdot q) & = & push_\partial(B,p)\cdot push_\partial(B,q)\\
push_\partial(B,c\to p) & = & c\to push_\partial(B,p)\\
push_\partial(B,c\to p\diamond q) & = & c\to push_\partial(B,p)\diamond push_\partial(B,q)\\
push_\partial(B,\Sigma_{d:D}p) & = & \Sigma_{d:D}push_\partial(B,p)\\
push_\partial(B,p \,{}^{\varsigma}\, t) & = & push_\partial(B,p) \,{}^{\varsigma}\, t\\
push_\partial(B,p\ll q) & = & push_\partial(B,p)\ll push_\partial(B,q)\\
push_\partial(B,p\parallel q) & = & push_\partial\,(B,p)\parallel push_\partial\,(B,q)\\
push_\partial(B,p ⌊⌊ q) & = & push_\partial\,(B,p) ⌊⌊ push_\partial\,(B,q)\\
push_\partial(B,p\mid q) & = & push_\partial\,(B,p)\mid push_\partial\,(B,q)\\
push_\partial(B,\rho_R(p)) & = & \rho_R\left(R^{-1}\,(B)\,,p\right)\\
push_\partial(B,\partial_{B_1}(p)) & = & push_\partial(B\cup B_1,p)\\
push_\partial(B,\tau_I(p)) & = & \tau_I\,(push_\partial\,(B\setminus I,p))\\
push_\partial(B,\Gamma_C(p)) & = & \mathsf{block}(B,\Gamma_C\,(push_\partial\,(B',p)) \text{ where } B'=B\setminus\{b\in B\mid \exists_{\gamma\to c\in C}.b\in\gamma\wedge c\notin B\}\\
push_\partial(B,\nabla_V(p)) & = & push_\nabla(\partial_B(A),p,\emptyset),
\end{array}
$$

where

$$
\mathsf{block}(B,p) \quad = \quad \left\{\begin{array}{ll} p & \text{if } B=\emptyset\\ \partial_B(p) & \text{otherwise}\end{array}\right.
$$

Example 3 The presence of $R^{-1}(\partial_B(A))$ instead of just $R^{-1}(A)$ in the right hand side of the rename operator is explained by the example $push_\nabla(\{b\},\rho_{\{b\to c\}}b)$. We see that $\rho_{\{b\to c\}}push_\nabla(R^{-1}(A),p)=\rho_{\{b\to c\}}push_\nabla(\{b\},b)=\rho_{\{b\to c\}}b=c$, which is clearly the wrong answer.

1.3.9 Allow sets

There are two rules in the definition of $push_\nabla$ where the allow set can/should not be computed explicitly. The computation of $push_\nabla(A,p\parallel q)$ involves computation of $push_\nabla(p,A^{\subseteq})$. We want to avoid the computation of $A^{\subseteq}$, since it can become very large. The computation of $push_\nabla(A,\tau_I(p))$ involves computation of $push_\nabla(p,\tau_I^{-1}(A))$. The set $\tau_I^{-1}(A)=AI^*$ is infinite.

In the implementation we use allow sets of the form $A^{\subseteq}I^*$, where A is a set of multi action names and I is a set of action names. The $^{\subseteq}$ is optional and I may be empty. Such an allow set is stored as two sets A and I, together with an attribute that tells if $^{\subseteq}$ is appicable. We need to show that allow sets are closed under the operations in $push_{\nabla}$.

$$\begin{array}{lcl}
\partial_B(A^{\subseteq}I^*) & = & \tau_B(A)^{\subseteq}\tau_B(I)^* \\
\tau_{I_1}^{-1}\left(A^{\subseteq}I^*\right) & = & \partial_{I_1}(A^{\subseteq})\left(I \cup I_1\right)^* \\
\left(A^{\subseteq}I^*\right) \cap V & = & \{\beta \in V \mid \exists_{\alpha \in A}.\tau_I(\beta) \sqsubseteq \alpha\} \\
R^{-1}\left(A^{\subseteq}I^*\right) & = & R^{-1}\left(A^{\subseteq}\right)R^{-1}\left(I\right)^* \\
C^{-1}\left(A^{\subseteq}I^*\right) & \subseteq & C^{-1}\left(A\right)^{\subseteq} act\left(C^{-1}\left(I\right)\right)^* \\
\left(A^{\subseteq}I^*\right) \leftarrowtail A_1 & = & A^{\subseteq}I^* \\
\left(A^{\subseteq}I^*\right)^{\subseteq} & = & A^{\subseteq}I^* \\
\partial_B(AI^*) & = & \partial_B(A)\tau_B(I)^* \\
\tau_{I_1}^{-1}\left(AI^*\right) & = & \partial_{I_1}(A)\left(I \cup I_1\right)^* \\
\left(AI^*\right) \cap V & = & \{\beta \in V \mid \exists_{\alpha \in A}.\tau_I(\beta) = \alpha\} \\
R^{-1}\left(AI^*\right) & = & R^{-1}\left(A\right)R^{-1}\left(I\right)^* \\
C^{-1}\left(AI^*\right) & \subseteq & C^{-1}\left(A\right) act\left(C^{-1}\left(I\right)\right)^* \\
\left(AI^*\right)^{\subseteq} & = & A^{\subseteq}I^*
\end{array}$$

where we used the following properties:

$$\begin{array}{lcl}
\partial_B\left(A_1A_2\right) & = & \partial_B\left(A_1\right)\partial_B\left(A_1\right) \\
\partial_B\left(A^{\subseteq}\right) & = & \tau_B(A)^{\subseteq} \\
R^{-1}\left(A_1A_2\right) & = & R^{-1}\left(A_1\right)R^{-1}\left(A_2\right) \\
R^{-1}\left(A^*\right) & = & R^{-1}\left(A\right)^* \\
C^{-1}\left(A^{\subseteq}\right) & \subseteq & C^{-1}\left(A\right)^{\subseteq} \\
C^{-1}\left(A_1A_2\right) & = & C^{-1}\left(A_1\right)C^{-1}\left(A_2\right) \\
C^{-1}\left(A^*\right) & = & C^{-1}\left(A\right)^* \\
A^{\subseteq} \leftarrowtail A_1 & = & A^{\subseteq}
\end{array}$$

Note that in case of the communication we only have an inclusion relation instead of equality. This is done to stay within the format $A^{\subseteq}I^*$. As a consequence the implementation uses an over-approximation of $C^{-1}\left(A^{\subseteq}I^*\right)$ and $C^{-1}\left(AI^*\right)$. Furthermore note that the property $R^{-1}\left(A^{\subseteq}\right) = R^{-1}\left(A\right)^{\subseteq}$ does not hold. A counter example is $R = \{b \to a\}$ and $A = \{a, b \mid c\}$. In that case we have $R^{-1}\left(A^{\subseteq}\right) = \{a, b, c\}^{\subseteq}$ and $R^{-1}\left(A\right)^{\subseteq} = \{a, b\}^{\subseteq}$. Another property that was initially assumed, but that does not hold is $\left(AI^*\right) \leftarrowtail A_1 = \left(A \leftarrowtail \tau_I(A_1)\right)I^*$.

1.4 Optimization for $push_{\nabla}$

In some cases the $push_{\nabla}$ operator produces expressions that are too large. This section proposes an optimization for the case $push_{\nabla}(A, \Gamma_C(p))$ that can help to prevent this problem for certain practical cases.

$$push_{\nabla}(A, \Gamma_C(p)) \quad = \quad \begin{cases} \mathsf{allow}(A, \Gamma_{C \setminus C'}(push_{\nabla\Gamma}(A', C', p))) & \text{if } C \neq C' \\ push_{\nabla\Gamma}(A, C, p)) & \text{otherwise,} \end{cases}$$

with $C' = \{\beta \to b \in C \mid b \notin \bigcup_{\beta' \to b' \in C} \beta'\}$ and $A' = ((C \setminus C')(A))^{\subseteq}$ and

$$\begin{array}{lcl}
push_{\nabla\Gamma}(A, C, p \parallel q) & = & \mathsf{allow}\left(A, \Gamma_C\left(\mathsf{allow}\left(C^{-1}(A), p' \parallel q'\right)\right)\right) \text{ where } \begin{cases} p' & = & push_{\nabla\Gamma}(A', C, p) \\ q' & = & push_{\nabla\Gamma}(A'', C, q) \\ A' & = & C^{-1}(A)^{\subseteq} \setminus \left(C^{-1}(A) \setminus A\right) \\ A'' & = & \left(C^{-1}(A) \leftarrow\!\!\!\prec \alpha(p')\right) \setminus \left(C^{-1}(A) \setminus A\right) \end{cases} \\
push_{\nabla\Gamma}(A, C, p \mathbin{\lfloor\!\lfloor} q) & = & \mathsf{allow}\left(A, \Gamma_C\left(\mathsf{allow}\left(C^{-1}(A), p' \mathbin{\lfloor\!\lfloor} q'\right)\right)\right) \text{ where } \begin{cases} p' & = & push_{\nabla\Gamma}(A', C, p) \\ q' & = & push_{\nabla\Gamma}(A'', C, q) \\ A' & = & C^{-1}(A)^{\subseteq} \setminus \left(C^{-1}(A) \setminus A\right) \\ A'' & = & \left(C^{-1}(A) \leftarrow\!\!\!\prec \alpha(p')\right) \setminus \left(C^{-1}(A) \setminus A\right) \end{cases} \\
push_{\nabla\Gamma}(A, C, p \mid q) & = & \mathsf{allow}\left(A, \Gamma_C\left(\mathsf{allow}\left(C^{-1}(A), p' \mid q'\right)\right)\right) \text{ where } \begin{cases} p' & = & push_{\nabla\Gamma}(A', C, p) \\ q' & = & push_{\nabla\Gamma}(A'', C, q) \\ A' & = & C^{-1}(A)^{\subseteq} \setminus \left(C^{-1}(A) \setminus A\right) \\ A'' & = & \left(C^{-1}(A) \leftarrow\!\!\!\prec \alpha(p')\right) \setminus \left(C^{-1}(A) \setminus A\right) \end{cases} \\
push_{\nabla\Gamma}(A, C, \partial_B(p)) & = & push_{\nabla\Gamma}(\partial_B(A), C, p) \\
push_{\nabla\Gamma}(A, C, \nabla_V(p)) & = & push_{\nabla\Gamma}(A \cap V, C, p) \\
push_{\nabla\Gamma}(A, C, p) & = & \mathsf{allow}(A, \Gamma_C(p')) \text{ where } p' = push_{\nabla}(C^{-1}(A), p) \text{ for all other cases of } p
\end{array}$$

Note that in this case the allow set A has the general shape $\left(A_1^{\subseteq} \setminus A_2^{\subseteq}\right) I^*$ (?), with the subset operator $\subseteq$ optional, and with I possibly empty. To implement this optimization, it needs to be investigated if such a set A is closed under the operations $\partial_B(A)$, $\tau_{I_1}^{-1}(A)$, $A \cap V$, $R^{-1}(A)$, $C^{-1}(A)$, $A \leftarrow\!\!\!\prec A_1$, $A^{\subseteq}$ and $C(A)$.