1 Process Library Implementation Notes

1.1 Processes

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

<table>
<thead>
<tr>
<th>expression</th>
<th>C++ equivalent</th>
<th>ATerm grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>a(e)</td>
<td>action(a,e)</td>
<td>Action</td>
</tr>
<tr>
<td>P(e)</td>
<td>process(P,e)</td>
<td>Process</td>
</tr>
<tr>
<td>P(d := e)</td>
<td>process_assignment(P,d := e)</td>
<td>ProcessAssignment</td>
</tr>
<tr>
<td>δ</td>
<td>delta()</td>
<td>Delta</td>
</tr>
<tr>
<td>τ</td>
<td>tau()</td>
<td>Tau</td>
</tr>
<tr>
<td>∑d x</td>
<td>sum(d,x)</td>
<td>Sum</td>
</tr>
<tr>
<td>δB(x)</td>
<td>block(B,x)</td>
<td>Block</td>
</tr>
<tr>
<td>τB(x)</td>
<td>hide(B,x)</td>
<td>Hide</td>
</tr>
<tr>
<td>ρR(x)</td>
<td>rename(R,x)</td>
<td>Rename</td>
</tr>
<tr>
<td>ΓC(x)</td>
<td>commu(C,x)</td>
<td>Comm</td>
</tr>
<tr>
<td>∩V(x)</td>
<td>allow(V,x)</td>
<td>Allow</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>sync(x,y)</td>
</tr>
<tr>
<td>x \cdot y</td>
<td>seq(x,y)</td>
<td>Seq</td>
</tr>
<tr>
<td>c \rightarrow x</td>
<td>if then(c,x)</td>
<td>IfThen</td>
</tr>
<tr>
<td>c \rightarrow x \circ y</td>
<td>if then else(c,x,y)</td>
<td>IfThenElse</td>
</tr>
<tr>
<td>x \ll y</td>
<td>binit(x,y)</td>
<td>BInit</td>
</tr>
<tr>
<td>x \parallel y</td>
<td>merge(x,y)</td>
<td>Merge</td>
</tr>
<tr>
<td>x \parallel y</td>
<td>lmerge(x,y)</td>
<td>LMerge</td>
</tr>
<tr>
<td>x + y</td>
<td>choice(x,y)</td>
<td>Choice</td>
</tr>
</tbody>
</table>

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 | \cdots | a_n, with a_i actions. A communication expression is of the form b_1 | \cdots | 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:

\[
\text{<linear process expression>} ::= \text{choice(<linear process expression>, <linear process expression>)} \\
| \text{<summand>}
\]

\[
\text{<summand>} ::= \text{sum(<variables>, <alternative>)} \\
| \text{<conditional action prefix>}
| \text{<conditional deadlock>}
\]

\[
\text{<conditional action prefix>} ::= \text{if_then(<condition>, <action prefix>)} \\
| \text{<action prefix>}
\]

\[
\text{<action prefix>} ::= \text{seq(<timed multiaction>, <process reference>)} \\
| \text{<timed multiaction>}
\]

\[
\text{<timed multiaction>} ::= \text{at_time(<multiaction>, <time stamp>)} \\
| \text{<multiaction>}
\]

\[
\text{<multiaction>} ::= \text{tau()} \\
| \text{<action>}
| \text{sync(<multiaction>, <multiaction>)}
\]

\[
\text{<conditional deadlock>} ::= \text{if_then(<condition>, <timed deadlock>)} \\
| \text{<timed deadlock>}
\]

\[
\text{<timed deadlock>} ::= \text{delta()} \\
| \text{at_time(delta(), <time stamp>)}
\]

\[
\text{<process reference>} ::= \text{process(<process identifier>, <data expressions>)} \\
| \text{process_assignment(<process identifier>, <data assignments>)}
\]
1.2 Guarded process expressions

We define the predicate \texttt{is\_guarded} for process expressions as follows: \texttt{is\_guarded}(p) = \texttt{is\_guarded}(p, \emptyset)

\[
\begin{align*}
\texttt{is\_guarded}(a(e), W) &= \text{true} \\
\texttt{is\_guarded}(\delta, W) &= \text{true} \\
\texttt{is\_guarded}(\tau, W) &= \text{true} \\
\texttt{is\_guarded}(P(e), W) &= \left\{ \begin{array}{ll}
\text{false} & \text{if } P \in W \\
\texttt{is\_guarded}(P, W \cup \{P\}) & \text{if } P \notin W
\end{array} \right. \\
\texttt{is\_guarded}(p + q, W) &= \texttt{is\_guarded}(p, W) \land \texttt{is\_guarded}(q, W) \\
\texttt{is\_guarded}(p \cdot q, W) &= \texttt{is\_guarded}(p, W) \\
\texttt{is\_guarded}(c \rightarrow p, W) &= \texttt{is\_guarded}(p, W) \\
\texttt{is\_guarded}(c \rightarrow q \bowtie q, W) &= \texttt{is\_guarded}(p, W) \land \texttt{is\_guarded}(q, W) \\
\texttt{is\_guarded}((\Sigma_{d:D} p), W) &= \texttt{is\_guarded}(p, W) \\
\texttt{is\_guarded}(p \cdot t, W) &= \texttt{is\_guarded}(p, W) \\
\texttt{is\_guarded}(p \bowtie q, W) &= \texttt{is\_guarded}(p, W) \\
\texttt{is\_guarded}(p \parallel q, W) &= \texttt{is\_guarded}(p, W) \land \texttt{is\_guarded}(q, W) \\
\texttt{is\_guarded}(p \mid q, W) &= \texttt{is\_guarded}(p, W) \land \texttt{is\_guarded}(q, W) \\
\texttt{is\_guarded}(\rho_R(p), W) &= \texttt{is\_guarded}(p, W) \\
\texttt{is\_guarded}(\partial_B(p), W) &= \texttt{is\_guarded}(p, W) \\
\texttt{is\_guarded}(\tau_I(p), W) &= \texttt{is\_guarded}(p, W) \\
\texttt{is\_guarded}(\Gamma_C(p), W) &= \texttt{is\_guarded}(p, W) \\
\texttt{is\_guarded}(\nabla_V(p), W) &= \texttt{is\_guarded}(p, W)
\end{align*}
\]

N.B. This specification assumes that process names are unique. In mCRL2 process names can be overloaded, therefore in the implementation \(W\) contains \textit{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 $\pi = a(e_1, \ldots, e_n)$ to denote an action, and $\pi = \pi_1 \mid \ldots \mid \pi_n$ 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 $\mathcal{A}$, $\mathcal{A}_1$, $\mathcal{A}_2$, $\ldots$.

A communication $C$ maps multi action names to action names, and is denoted as $\{\alpha_1 \rightarrow a_1, \ldots, \alpha_n \rightarrow a_n\}$. A renaming $R$ is a substitution on action names, and is denoted as $R = \{a_1 \rightarrow b_1, \ldots, a_n \rightarrow b_n\}$. 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 $\alpha$ using the following grammar:

$$\alpha ::= a p \mid \alpha$$

where $a$ is an action, and where $p$ is used to distinguish alternatives.

We define pCRL terms $p$ using the following grammar:

$$p ::= a p \mid P \mid \delta \mid \tau \mid p + p \cdot p \mid \tau \mid p \rightarrow p \mid p \circ p \mid \Sigma_{d \in D} p \mid p \uparrow t \mid p \ll p,$$

and parallel mCRL terms $q$ using the following grammar:

$$q ::= p \parallel q \parallel q \parallel q \mid q \parallel p_R(q) \mid \partial_B(q) \mid \tau I(q) \mid \Gamma_C(q) \mid \nabla_V(q).$$

 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 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

$$A^\subseteq = \{\alpha \mid \exists \beta. \alpha \beta \in A\}$$
$$A_1 A_2 = \{\alpha \beta \mid \alpha \in A_1 \text{ and } \beta \in A_2\}$$
$$A_1 \leftarrow A_2 = \{\alpha \mid \exists \beta. \alpha \beta \in A_1 \text{ and } \beta \in A_2\}$$

Note that $\beta$ can take the value $\tau$ in the definition of $A_1 \leftarrow A_2$, which implies $A_1 \subseteq A_1 \leftarrow A_2$. 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

$$C(A) = \cup_{\alpha \in A} \text{COMM}(C, \alpha)$$
$$C^{-1}(A) = \cup_{\alpha \in A} \text{COMM}^{-1}(C, \alpha)$$
$$\text{filter}_C(C, A) = \{\gamma \rightarrow c \in C \mid \exists \alpha \in A. \gamma \subset \alpha\}$$
where $\text{Comm}$ and $\text{CommInverse}$ are defined using pseudo code as follows:

\[
\text{Comm}(C, \alpha) \\
R := \{\alpha\} \\
\text{for } \gamma \rightarrow c \in C \text{ do} \\
\quad \text{if } \exists \beta.\alpha = \beta\gamma \text{ then } R := R \cup \text{Comm}(C, \beta c) \\
\text{return } R \\
\]

\[
\text{CommInverse}(C, \alpha_1, \alpha_2) \\
R := \{\alpha_1\alpha_2\} \\
\text{for } \gamma \rightarrow c \in C \text{ do} \\
\quad \text{if } \exists \beta.\alpha_1 = \beta c \text{ then } R := R \cup \text{CommInverse}(C, \beta, \alpha_2\gamma) \\
\text{return } R \\
\]

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

Let $R$ be a rename set, then we define

\[
R(\alpha) = \{R(\alpha_i) \mid \alpha_i \in \alpha\} \\
R^{-1}(\alpha) = \{\beta \mid R(\beta) = \alpha\} \\
R(A) = \{R(\alpha) \mid \alpha \in A\} \\
R^{-1}(A) = \{R^{-1}(\alpha) \mid \alpha \in A\} \\
\]

Let $I$ be a hide set, then we define

\[
\tau_I(A) = \{\beta \mid \exists \alpha \in A, \gamma \in I, \alpha = \beta\gamma \land \beta \cap I = \emptyset\} \\
\tau_I^{-1}(A) = \partial_I(A)^* \\
\]

Let $B$ be a block set, then we define

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

We define a mapping $\text{act}$ that extracts the individual action names of a set of multi action names:

\[
\text{act}(a_1 \mid \ldots \mid a_n) = \{a_1 \mid \ldots \mid a_n\} \\
\text{act}(A) = \bigcup_{\alpha \in A} \text{act}(\alpha) \\
\]

### 1.3.4 The mapping $\alpha$

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

\[
\alpha(\pi, W) = \{a\} \\
\alpha(P, W) = \{\emptyset\} \text{ if } P \in W \\
\quad \text{where } P = p \text{ is the equation of } P \\
\alpha(\tau, W) = \{\tau\} \\
\alpha(p + q, W) = \alpha(p, W) \cup \alpha(q, W) \\
\alpha(p \cdot q, W) = \alpha(p, W) \cup \alpha(q, W) \\
\alpha(c \rightarrow p, W) = \alpha(p, W) \\
\alpha(c \rightarrow p \circ q, W) = \alpha(p, W) \cup \alpha(q, W) \\
\alpha(\Sigma_{t,d} p, W) = \alpha(p, W) \\
\alpha(p \cdot t, W) = \alpha(p, W) \\
\alpha(p \ll q, W) = \alpha(p, W) \cup \alpha(q, W) \\
\alpha(p \parallel q, W) = \alpha(p, W) \cup \alpha(q, W) \cup \alpha(p, W)\alpha(q, W) \\
\alpha(p \mid q, W) = \alpha(p, W) \cup \alpha(q, W) \cup \alpha(p, W)\alpha(q, W) \\
\alpha(p \mid q, W) = \alpha(p, W) \cup \alpha(q, W) \cup \alpha(p, W)\alpha(q, W) \\
\alpha(p \parallel q, W) = \alpha(p, W) \cup \alpha(q, W) \cup \alpha(p, W)\alpha(q, W) \\
\alpha(p \parallel q, W) = \alpha(p, W) \cup \alpha(q, W) \cup \alpha(p, W)\alpha(q, W) \\
\alpha(p, W) = R(\alpha(p, W)) \\
\alpha(\partial_D(p, W)) = \partial_D(\alpha(p, W)) \\
\alpha(\partial_I(p, W)) = \tau_I(\alpha(p, W)) \\
\alpha(\Gamma_C(p, W)) = C(\alpha(p, W)) \\
\alpha(\nabla_V(p, W)) = \alpha(p, W) \cap (V \cup \{\tau\}) \\
\]
Example 1
If $C = \{a \mid b \rightarrow c\}$, then $\alpha(T_C(a(1) \mid b(2))) = \{a, b, c, a \mid b\}$. 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:

\[
\alpha(\pi, W, A) = \begin{cases} 
\{a\} & \text{if } a \in A \\
\emptyset & \text{if } a \notin A
\end{cases}
\]

\[
\alpha(P, W, A) = \begin{cases} 
\emptyset & \text{if } P \in W \\
\alpha(p, W) & \text{if } P \notin W
\end{cases}
\]

\[
\alpha(p + q, W, A) = \alpha(p, W, A) \cup \alpha(q, W, A)
\]

\[
\alpha(p \cdot q, W, A) = \alpha(p, W, A) \cup \alpha(q, W, A)
\]

\[
\alpha(c \rightarrow p, W, A) = \alpha(p, W, A)
\]

\[
\alpha(c \rightarrow p \circ q, W, A) = \alpha(p, W, A) \cup \alpha(q, W, A)
\]

\[
\alpha(\Sigma_{e,D}p, W, A) = \alpha(p, W, A)
\]

\[
\alpha(p \cdot t, W, A) = \alpha(p, W, A)
\]

\[
\alpha(p \ll q, W, A) = \alpha(p, W, A) \cup \alpha(q, W, A)
\]

\[
\alpha(p \parallel q, W, A) = \alpha(p, W, A) \cup \alpha(q, W, A) \cup \alpha(p, W, A^\leq) \alpha(q, W, A^\leq)
\]

\[
\alpha(p \parallel q, W, A) = \alpha(p, W, A) \cup \alpha(q, W, A) \cup \alpha(p, W, A^\leq) \alpha(q, W, A^\leq)
\]

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 $\text{proc}(p, W)$ as follows:

\[
\text{proc}(\pi, W) = \emptyset
\]

\[
\text{proc}(P, W) = \begin{cases} 
\emptyset & \text{if } P \in W \\
\text{proc}(P) \cup \text{proc}(q, W) & \text{if } P \notin W
\end{cases}
\]

\[
\text{proc}(p + q, W) = \text{proc}(p, W) \cup \text{proc}(q, W)
\]

\[
\text{proc}(p \cdot q, W) = \text{proc}(p, W) \cup \text{proc}(q, W)
\]

\[
\text{proc}(c \rightarrow p, W) = \text{proc}(p, W)
\]

\[
\text{proc}(c \rightarrow p \circ q, W) = \text{proc}(p, W) \cup \text{proc}(q, W)
\]

\[
\text{proc}(\Sigma_{e,D}p, W) = \text{proc}(p, W)
\]

\[
\text{proc}(p \cdot t, W) = \text{proc}(p, W)
\]

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

\[
\alpha(p + q, W, A) = \alpha(p, W, A) \cup \alpha(q, W \cup \text{proc}(p, W), A)
\]

\[
\alpha(p \cdot q, W, A) = \alpha(p, W, A) \cup \alpha(q, W \cup \text{proc}(p, W), A)
\]

\[
\alpha(c \rightarrow p \circ q, W, A) = \alpha(p, W, A) \cup \alpha(q, W \cup \text{proc}(p, W), A)
\]

Note that the value $\text{proc}(p, W)$ 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:
1.3.8 The mappings push, push_{\nabla} and push_{\partial}

We define mappings \textit{push}, \textit{push}_{\nabla} and \textit{push}_{\partial} such that \textit{push}(p) is bisimulation equivalent to \( P \), \textit{push}_{\nabla}(A, p) is bisimulation equivalent to \( \nabla_{A}(p) \), and \textit{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 \( \tau \). Let \( \mathcal{E} = \{ P_{1}(d) = p_{1}, \ldots, P_{n}(d) = p_{n} \} \) be a sequence of process equations.

\[
\begin{align*}
\alpha(\pi, A) &= \left\{ \{a\} \text{ if } a \in A \right\} \\
\alpha(P, A) &= \alpha(p, A), \text{ where } P = p \text{ is the equation of } P \\
\alpha(p + q, A) &= \alpha(p, A) \cup \alpha(q, A) \\
\alpha(p \cdot q, A) &= \alpha(p, A) \cup \alpha(q, A) \\
\alpha(c \rightarrow p, A) &= \alpha(p, A) \\
\alpha(c \rightarrow p \circ q, A) &= \alpha(p, A) \cup \alpha(q, A) \\
\alpha(\Sigma_{d:B}p, A) &= \alpha(p, A) \\
\alpha(p \cdot t, A) &= \alpha(p, A) \\
\alpha(p \ll q, A) &= \alpha(p, A) \cup \alpha(q, A) \\
\alpha(p \parallel q, A) &= \alpha(p, A) \cup \alpha(q, A) \cup \alpha(p, A^{\subseteq})\alpha(q, A \leftarrow \alpha(p, A^{\subseteq})) \\
\alpha(p \mid q, A) &= \alpha(p, A^{\subseteq})\alpha(q, A \leftarrow \alpha(p, A^{\subseteq})) \\
\alpha(p_{R}(p), A) &= R(\alpha(p, R^{-1}(A))) \\
\alpha(\partial_{B}(p), A) &= \alpha(p, \partial_{B}(A)) \\
\alpha(\tau_{I}(p), A) &= \tau_{I}(\alpha(p, \tau_{I}^{-1}(A))) \\
\alpha(\Gamma_{C}(p), A) &= C(\alpha(p, C^{-1}(A))) \\
\alpha(\nabla_{V}(p), A) &= \alpha(p, A \cap V) \\
\end{align*}
\]

\[
\begin{align*}
\text{push}(p) &= p \text{ if } p \text{ is a \textit{pCRL} expression} \\
\text{push}(p \parallel q) &= \text{push}(p) \parallel \text{push}(q) \\
\text{push}(p \parallel q) &= \text{push}(p) \parallel \text{push}(q) \\
\text{push}(p \mid q) &= \text{push}(p) \mid \text{push}(q) \\
\text{push}(p_{R}(p)) &= \rho_{R}(\text{push}(p)) \\
\text{push}(\partial_{B}(p)) &= \text{push}_{\partial}(B, p) \\
\text{push}(\tau_{I}(p)) &= \tau_{I}(\text{push}(p)) \\
\text{push}(\Gamma_{C}(p)) &= \Gamma_{C}(\text{push}(p)) \\
\text{push}(\nabla_{V}(p)) &= \text{push}_{\nabla}(V, p) \\
\end{align*}
\]
We assume that $P_{A,e}^\Sigma$ is a unique name for every $P \in \{P_1, \ldots, P_n\}$, multi action name set $A$ and sequence of data expressions $e$.

\[
push_V(A, \pi) = \begin{cases} \pi & \text{if } N(\pi) \in A \\ \delta & \text{otherwise} \end{cases}
\]

\[
push_V(A, P(e)) = P_V^\Sigma(e), \text{ where } P(d) = p \text{ is the equation of } P, \text{ and where } P_A^\Sigma(d) = push_V(A, p) \text{ is a new equation}
\]

\[
push_V(A, \delta) = \delta
\]

\[
push_V(A, \tau) = \tau
\]

\[
push_V(A, p + q) = \nabla_A(p + q)
\]

\[
push_V(A, p \cdot q) = \nabla_A(p \cdot q)
\]

\[
push_V(A, c \to p) = \nabla_A(c \to p)
\]

\[
push_V(A, c \to p \circ q) = \nabla_A(c \to p \circ q)
\]

\[
push_V(A, \Sigma, Dp) = \nabla_A(\Sigma, Dp)
\]

\[
push_V(A, p^* t) = \nabla_A(p^* t)
\]

\[
push_V(A, p \leq q) = \nabla_A(p \leq q)
\]

\[
push_V(A, p \parallel q) = \nabla_A(A, p' \parallel q') \text{ where } \begin{cases} p' = push_V(A^\subseteq, p) \\ q' = push_V(A^\subseteq, p) \end{cases}
\]

\[
push_V(A, p \parallel q) = \nabla_A(A, p' \parallel q') \text{ where } \begin{cases} p' = push_V(A^\subseteq, p) \\ q' = push_V(A^\subseteq, p) \end{cases}
\]

\[
push_V(A, p \mid q) = \nabla_A(A, p \mid q') \text{ where } \begin{cases} p' = push_V(A^\subseteq, p) \\ q' = push_V(A^\subseteq, p) \end{cases}
\]

\[
push_V(A, \rho_R(p)) = \rho_R(p') \text{ where } p' = push_V(R^{-1}(A), p)
\]

\[
push_V(A, \partial_B(p)) = push_V(\partial_B(A), p)
\]

\[
push_V(A, \tau_I(p)) = \tau_I(p') \text{ where } p' = push_V(\tau_I^{-1}(A), p)
\]

\[
push_V(A, \Gamma_C(p)) = \allow(A, \Gamma_C(p')) \text{ where } p' = push_V(C^{-1}(A), p)
\]

\[
push_V(A, \nabla_V(p)) = push_V(A \cap V, p)
\]

**Optimizations** During the computation of $push_V$ the following optimizations are applied in the right hand side of each equation:

\[
\nabla_A(p) = \begin{cases} p & \text{if } (A \cup \{\tau\}) \cap \alpha(p) = \alpha(p) \\ \nabla_{A \cap \alpha(p)}(p) & \text{otherwise} \end{cases}
\]

\[
\nabla_\emptyset(p) = \begin{cases} \tau & \text{if } p = \tau \\ \delta & \text{otherwise} \end{cases}
\]

\[
\Gamma_C(p) = \Gamma_{filter_C(C, \alpha(p))}(p)
\]

\[
\delta \mid \delta = \delta
\]

\[
\delta \parallel \delta = \delta
\]

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

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

**Example 2** Let $P = a \cdot \nabla_{\{a\}}(P)$. Then $push_V(\{a\}, P, \emptyset) = P'$, with $P' = push_V(\{a\}, a \cdot \nabla_{\{a\}}(P), \{(P, \{a\}, P')\}) = push_V(\{a\}, a, ((P, \{a\}, P')) \cdot push_V(\{a\}, \nabla_{\{a\}}(P), \{(P, \{a\}, P')\}) = \cdots = a \cdot P'$.

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

\[
push_\delta(B, \overline{\alpha}) = \begin{cases} \alpha & \text{if } N(\overline{\alpha}) \cap B = \emptyset \\ \delta & \text{otherwise} \end{cases}
\]

\[
push_\delta(B, P(e)) = \begin{cases} \text{where } P(d) = p \text{ is the equation of } P, & \text{and} \\ \text{where } P_{B,e}(d) = push_\delta(B, p) \text{ is a new equation} \end{cases}
\]

\[
push_\delta(B, \delta) = \delta
\]

\[
push_\delta(B, \tau) = \tau
\]

\[
push_\delta(B, p + q) = push_\delta(B, p) + push_\delta(B, q)
\]

\[
push_\delta(B, p \cdot q) = push_\delta(B, p) \cdot push_\delta(B, q)
\]

\[
push_\delta(B, c \rightarrow p) = c \rightarrow push_\delta(B, p)
\]

\[
push_\delta(B, c \rightarrow p \circ q) = c \rightarrow push_\delta(B, p) \circ push_\delta(B, q)
\]

\[
push_\delta(B, \Sigma_d \cdot p) = \Sigma_d \cdot push_\delta(B, p)
\]

\[
push_\delta(B, \rho R(p)) = \rho_R \left( R^{-1}(B), p \right)
\]

\[
push_\delta(B, \rho B_1(p)) = push_\delta(B \cup B_1, p)
\]

\[
push_\delta(B, \tau I(p)) = \tau I \left( push_\delta(B \setminus I, p) \right)
\]

\[
push_\delta(B, \Gamma_C(p)) = block(B, \Gamma_C \left( push_\delta(B', p) \right)) \text{ where } B' = B \setminus \{ b \in B \mid \exists_{\gamma \rightarrow c \in C} b \in \gamma \land c \notin B \}
\]

\[
push_\delta(B, \nabla V(p)) = push_V(\partial_B(A), p, \emptyset),
\]

where

\[
\text{block}(B, p) = \begin{cases} p & \text{if } B = \emptyset \\ \partial_B(p) & \text{otherwise} \end{cases}
\]

**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 \( push_V(\{b\}, \rho_{\{b \rightarrow c\}} b) \). We see that \( \rho_{\{b \rightarrow c\}} push_V(R^{-1}(A), p) = \rho_{\{b \rightarrow c\}} push_V(\{b\}, b) = \rho_{\{b \rightarrow c\}} b = c \), which is clearly the wrong answer.

### 1.3.9 Allow sets

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

In the implementation we use allow sets of the form \( A \subseteq \Gamma^* \), 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 applicable. We need to show that allow sets are closed
under the operations in $\text{push}_\mathcal{V}$.

\[
\begin{align*}
\partial_B(A^\subseteq I^*) & = \tau_B(A)^\subseteq \tau_B(I)^* \\
\tau^{-1}_{I_1}(A^\subseteq I^*) & = \partial_I(A^\subseteq) (I \cup I_1)^* \\
(A^\subseteq I^*) \cap V & = \{ \beta \in V \mid \exists \alpha \in A, \tau_I(\beta) \sqsubseteq \alpha \} \\
R^{-1}(A^\subseteq I^*) & = R^{-1}(A^\subseteq) R^{-1}(I)^* \\
C^{-1}(A^\subseteq I^*) \subseteq C^{-1}(A)^\subseteq \text{act} (C^{-1}(I))^* \\
(A^\subseteq I^*) \lhd A_1 & = A^\subseteq I^* \\
(A^\subseteq I^*) \subseteq & = A^\subseteq I^* \\
\partial_B(AI^*) & = \partial_B(A) \tau_B(I)^* \\
\tau^{-1}_{I_1}(AI^*) & = \partial_I(A)(I \cup I_1)^* \\
(AI^*) \cap V & = \{ \beta \in V \mid \exists \alpha \in A, \tau_I(\beta) = \alpha \} \\
R^{-1}(AI^*) & = R^{-1}(A) R^{-1}(I)^* \\
C^{-1}(AI^*) \subseteq C^{-1}(A) \text{act} (C^{-1}(I))^* \\
(AI^*) \subseteq & = A^\subseteq I^*
\end{align*}
\]

where we used the following properties:

\[
\begin{align*}
\partial_B(A_1A_2) & = \partial_B(A_1) \partial_B(A_2) \\
\partial_B(A^\subseteq) & = \tau_B(A)^\subseteq \\
R^{-1}(A_1A_2) & = R^{-1}(A_1) R^{-1}(A_2) \\
R^{-1}(A^\subseteq) & = R^{-1}(A)^* \\
C^{-1}(A^\subseteq) \subseteq & = C^{-1}(A)^\subseteq \\
C^{-1}(A_1A_2) & = C^{-1}(A_1) C^{-1}(A_2) \\
C^{-1}(A^\subseteq) & = C^{-1}(A)^* \\
A^\subseteq \lhd A_1 & = A^\subseteq
\end{align*}
\]

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 $C^{-1}(A^\subseteq I^*)$ and $C^{-1}(AI^*)$. Furthermore note that the property $R^{-1}(A^\subseteq) = R^{-1}(A)^\subseteq$ does not hold. A counter example is $R = \{ b \rightarrow a \}$ and $A = \{ a, b \mid c \}$. In that case we have $R^{-1}(A^\subseteq) = \{ a, b, c \}^\subseteq$ and $R^{-1}(A)^\subseteq = \{ a, b \}^\subseteq$. Another property that was initially assumed, but that does not hold is $(AI^*) \lhd A_1 = (A \leftarrow \tau_I(A_1)) I^*$.  

11
1.4 Optimization for $\text{push}_\nabla$

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

$$\text{push}_\nabla(A, \Gamma_C(p)) = \begin{cases} \text{allow}(A, \Gamma_C \cap C'(\text{push}_\nabla(A', C', p))) & \text{if } C \neq C' \\ \text{push}_\nabla(A, C, p) & \text{otherwise}, \end{cases}$$

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

$$\text{push}_\nabla(A, C, p \parallel q) = \text{allow}(A, \Gamma_C(\text{allow}(C^{-1}(A), p' \parallel q'))),$$

$$\text{push}_\nabla(A, C, p \parallel q) = \text{allow}(A, \Gamma_C(\text{allow}(C^{-1}(A), p' \parallel q'))),$$

$$\text{push}_\nabla(A, C, p \mid q) = \text{allow}(A, \Gamma_C(\text{allow}(C^{-1}(A), p' \mid q'))),$$

$$\text{push}_\nabla(A, C, \partial_B(p)) = \text{push}_\nabla(\partial_B(A), C, p),$$

$$\text{push}_\nabla(A, C, \nabla_V(p)) = \text{push}_\nabla(A \cap V, C, p),$$

$$\text{push}_\nabla(A, C, p) = \text{allow}(A, \Gamma_C(p'))$$

where $p' = \text{push}_\nabla(A', C, p)$ and $q' = \text{push}_\nabla(A'', C, q)$.

Note that in this case the allow set $A$ has the general shape $(A_1 \subseteq A_2^\subseteq)^* (?)$, 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 $\partial_B(A)$, $\tau_{\nabla_V}^{-1}(A)$, $A \cap V$, $R^{-1}(A)$, $C^{-1}(A)$, $A \leftarrow A_1$, $A^\subseteq$ and $C(A)$. 

12