

# PBES Implementation Notes

Wieger Wesselink

December 22, 2008

This document contains details about data structures and algorithms of the PBES Library of the mCRL2 toolset.

## 1 Definitions

Parameterised Boolean Equation Systems (PBESs) are empty (denoted  $\epsilon$ ) or finite sequences of fixed point equations, where each equation is of the form  $(\mu X(d:D) = \phi)$  or  $(\nu X(d:D) = \phi)$ . The left-hand side of each equation consists of a *fixed point symbol*, where  $\mu$  indicates a least and  $\nu$  a greatest fixed point, and a sorted predicate variable  $X$  of sort  $D \rightarrow B$ , taken from some countable domain of sorted predicate variables  $\mathcal{X}$ . The right-hand side of each equation is a predicate formula as defined below.

**Definition 1** Predicate formulae  $\phi$  are defined by the following grammar:

$$\phi ::= b \mid X(e) \mid \neg\phi \mid \phi \oplus \phi \mid \mathbf{Q}d : D. \phi$$

where  $\oplus \in \{\wedge, \vee, \Rightarrow\}$ ,  $\mathbf{Q} \in \{\forall, \exists\}$ ,  $b$  is a data term of sort  $\mathbf{B}$ ,  $X$  is a predicate variable,  $d$  is a data variable of sort  $D$  and  $e$  is a vector of data terms.

The set of predicate variables that occur in a predicate formula  $\phi$ , denoted by  $\text{occ}$ , is defined recursively as follows, for any formulae  $\phi_1, \phi_2$ :

$$\begin{array}{ll} \text{occ}(b) & =_{def} \emptyset \\ \text{occ}(\phi_1 \oplus \phi_2) & =_{def} \text{occ}(\phi_1) \cup \text{occ}(\phi_2) \end{array} \quad \begin{array}{ll} \text{occ}(X(e)) & =_{def} \{X\} \\ \text{occ}(\mathbf{Q}d : D. \phi_1) & =_{def} \text{occ}(\phi_1). \end{array}$$

Extended to equation systems,  $\text{occ}(\mathcal{E})$  is the union of all variables occurring at the right-hand side of equations in  $\mathcal{E}$ . Likewise, the set of predicate variable instantiations that occur in a predicate formula  $\phi$  is denoted by  $\text{iocc}$ , and is defined recursively as follows

$$\begin{array}{ll} \text{iocc}(b) & =_{def} \emptyset \\ \text{iocc}(\phi_1 \oplus \phi_2) & =_{def} \text{iocc}(\phi_1) \cup \text{iocc}(\phi_2) \end{array} \quad \begin{array}{ll} \text{iocc}(X(e)) & =_{def} \{X(e)\} \\ \text{iocc}(\mathbf{Q}d : D. \phi_1) & =_{def} \text{iocc}(\phi_1). \end{array}$$

For any equation system  $\mathcal{E}$ , the set of *binding predicate variables*,  $\text{bnd}(\mathcal{E})$ , is the set of variables occurring at the left-hand side of some equation in  $\mathcal{E}$ . Formally, we define:

$$\begin{array}{ll} \text{bnd}(\epsilon) & =_{def} \emptyset \\ \text{occ}(\epsilon) & =_{def} \emptyset \end{array} \quad \begin{array}{ll} \text{bnd}((\sigma X(d:D) = \phi) \mathcal{E}) & =_{def} \text{bnd}(\mathcal{E}) \cup \{X\} \\ \text{occ}((\sigma X(d:D) = \phi) \mathcal{E}) & =_{def} \text{occ}(\mathcal{E}) \cup \text{occ}(\phi). \end{array}$$

Let  $\text{dvar}(d)$  be the set of *free data variables* occurring in a data term  $d$ . The function  $\text{dvar}$  is extended to predicate formulae using

$$\begin{array}{ll} \text{dvar}(X(e)) & =_{def} \text{dvar}(e) \\ \text{dvar}(\phi_1 \oplus \phi_2) & =_{def} \text{dvar}(\phi_1) \cup \text{iocc}(\phi_2). \end{array} \quad \text{dvar}(\mathbf{Q}d : D. \phi_1) =_{def} \text{dvar}(\phi_1) \setminus \text{dvar}(d).$$

The set of freely occurring predicate variables in  $\mathcal{E}$ , denoted  $\text{pvar}(\mathcal{E})$  is defined as  $\text{occ}(\mathcal{E}) \setminus \text{bnd}(\mathcal{E})$ . An equation system  $\mathcal{E}$  is said to be *well-formed* iff every binding predicate variable occurs at the left-hand side of precisely one equation of  $\mathcal{E}$ . We only consider well-formed equation systems in this paper.

An equation system  $\mathcal{E}$  is called *closed* if  $\text{pvar}(\mathcal{E}) = \emptyset$  and *open* otherwise. An equation  $(\sigma X(d : D) = \phi)$ , where  $\sigma$  denotes either the fixed point sign  $\mu$  or  $\nu$ , is called *data-closed* if the set of data variables that occur freely in  $\phi$  is contained in the set of variables induced by the vector of variables  $d$ . An equation system is called *data-closed* iff each of its equations is data-closed.

**Definition 2** Action formulae  $\alpha$  are defined by the following grammar:

$$\alpha ::= b \mid \neg\alpha \mid \alpha \oplus \alpha \mid \mathbf{Q}d : D.\alpha \mid a(d) \mid \alpha^t$$

where  $\oplus \in \{\wedge, \vee, \Rightarrow\}$ ,  $\mathbf{Q} \in \{\forall, \exists\}$ ,  $b$  is a data term of sort  $\mathbf{B}$ ,  $X$  is a predicate variable,  $d$  is a data variable of sort  $D$  and  $a$  is an action label.

**Definition 3** State formulae  $\phi$  are defined by the following grammar:

$$\phi ::= b \mid X(e) \mid \neg\phi \mid \phi \oplus \phi \mid \mathbf{Q}d : D.\phi \mid \langle \alpha \rangle \phi \mid [\alpha] \phi \mid \Delta \mid \Delta(t) \mid \nabla \mid \nabla(t) \mid \sigma X(d:D) := e$$

where  $\oplus \in \{\wedge, \vee, \Rightarrow\}$ ,  $\mathbf{Q} \in \{\forall, \exists\}$ ,  $\sigma \in \{\mu, \nu\}$ ,  $b$  is a data term of sort  $\mathbf{B}$ ,  $X$  is a predicate variable,  $d$  is a data variable of sort  $D$  and  $e$  is a vector of data terms and  $\alpha$  is an action formula.

## 1.1 Monotonicity

**Definition 4** A state formula is called *monotonous* if it can be rewritten such that propositional variables are not inside the scope of a negation or an implication. More formally, a state formula  $\varphi$  is *monotonous* if  $m(\varphi) = \text{true}$ , where  $m$  is defined as follows. This definition applies to predicate formulae as well.

$m(\neg b)$	$=_{def}$	<b>true</b>
$m(\neg\neg\varphi)$	$=_{def}$	$m(\varphi)$
$m(\neg(\varphi \wedge \psi))$	$=_{def}$	$m(\neg\varphi) \wedge m(\neg\psi)$
$m(\neg(\varphi \vee \psi))$	$=_{def}$	$m(\neg\varphi) \wedge m(\neg\psi)$
$m(\neg(\varphi \Rightarrow \psi))$	$=_{def}$	$m(\varphi) \wedge m(\neg\psi)$
$m(\neg\forall d:D.\varphi)$	$=_{def}$	$m(\neg\varphi)$
$m(\neg\exists d:D.\varphi)$	$=_{def}$	$m(\neg\varphi)$
$m(\neg[\alpha]\varphi)$	$=_{def}$	$m(\neg\varphi)$
$m(\neg\langle\alpha\rangle\varphi)$	$=_{def}$	$m(\neg\varphi)$
$m(\neg\nabla)$	$=_{def}$	<b>true</b>
$m(\neg\nabla(t))$	$=_{def}$	<b>true</b>
$m(\neg\Delta)$	$=_{def}$	<b>true</b>
$m(\neg\Delta(t))$	$=_{def}$	<b>true</b>
$m(\neg X(e))$	$=_{def}$	<b>false</b>
$m(\neg\mu X(d:D := e).\varphi)$	$=_{def}$	$m(\neg\varphi[X := \neg X])$
$m(\neg\nu X(d:D := e).\varphi)$	$=_{def}$	$m(\neg\varphi[X := \neg X])$
$m(b)$	$=_{def}$	<b>true</b>
$m(\varphi \wedge \psi)$	$=_{def}$	$m(\varphi) \wedge m(\psi)$
$m(\varphi \vee \psi)$	$=_{def}$	$m(\varphi) \wedge m(\psi)$
$m(\varphi \Rightarrow \psi)$	$=_{def}$	$m(\neg\varphi) \wedge m(\psi)$
$m(\forall d:D.\varphi)$	$=_{def}$	$m(\varphi)$
$m(\exists d:D.\varphi)$	$=_{def}$	$m(\varphi)$
$m([\alpha]\varphi)$	$=_{def}$	$m(\varphi)$
$m(\langle\alpha\rangle\varphi)$	$=_{def}$	$m(\varphi)$
$m(\nabla)$	$=_{def}$	<b>true</b>
$m(\nabla(t))$	$=_{def}$	<b>true</b>
$m(\Delta)$	$=_{def}$	<b>true</b>
$m(\Delta(t))$	$=_{def}$	<b>true</b>
$m(X(e))$	$=_{def}$	<b>true</b>
$m(\mu X(d:D := e).\varphi)$	$=_{def}$	$m(\varphi)$
$m(\nu X(d:D := e).\varphi)$	$=_{def}$	$m(\varphi)$

## 1.2 Normalization

The normalization function  $h$  is a function that eliminates implications from a state formula  $\varphi$ , and that 'pushes' negations inwards to the level of data expressions. A precondition of  $h$  is that  $\varphi$  is monotonous. If this is not the case, during the computation a term  $\neg X(e)$  will be encountered.

$h(\neg b)$	$=_{def}$	$\neg b$
$h(\neg\neg\varphi)$	$=_{def}$	$h(\varphi)$
$h(\neg(\varphi \wedge \psi))$	$=_{def}$	$h(\neg\varphi) \vee h(\neg\psi)$
$h(\neg(\varphi \vee \psi))$	$=_{def}$	$h(\neg\varphi) \wedge h(\neg\psi)$
$h(\neg(\varphi \Rightarrow \psi))$	$=_{def}$	$h(\varphi) \wedge h(\neg\psi)$
$h(\neg\forall d:D.\varphi)$	$=_{def}$	$\exists d:D.h(\neg\varphi)$
$h(\neg\exists d:D.\varphi)$	$=_{def}$	$\forall d:D.h(\neg\varphi)$
$h(\neg[\alpha]\varphi)$	$=_{def}$	$[\alpha]h(\neg\varphi)$
$h(\neg\langle\alpha\rangle\varphi)$	$=_{def}$	$\langle\alpha\rangle h(\neg\varphi)$
$h(\neg\nabla)$	$=_{def}$	$\Delta$
$h(\neg\nabla(t))$	$=_{def}$	$\Delta(t)$
$h(\neg\Delta)$	$=_{def}$	$\nabla$
$h(\neg\Delta(t))$	$=_{def}$	$\nabla(t)$
$h(\neg X(e))$	$=_{def}$	<i>undefined</i>
$h(\neg\nu X(d:D := e). \varphi)$	$=_{def}$	$\nu X(d:D := e). h(\neg\varphi[X := \neg X])$
$h(\neg\mu X(d:D := e). \varphi)$	$=_{def}$	$\mu X(d:D := e). h(\neg\varphi[X := \neg X])$
$h(b)$	$=_{def}$	$b$
$h(\varphi \wedge \psi)$	$=_{def}$	$h(\varphi) \wedge h(\psi)$
$h(\varphi \vee \psi)$	$=_{def}$	$h(\varphi) \vee h(\psi)$
$h(\varphi \Rightarrow \psi)$	$=_{def}$	$h(\neg\varphi) \vee h(\psi)$
$h(\mathbf{Q}d:D.\varphi)$	$=_{def}$	$\forall d:D.h(\varphi)$
$h([\alpha]\varphi)$	$=_{def}$	$[\alpha]h(\varphi)$
$h(\langle\alpha\rangle\varphi)$	$=_{def}$	$\langle\alpha\rangle h(\varphi)$
$h(\nabla)$	$=_{def}$	$\nabla$
$h(\nabla(t))$	$=_{def}$	$\nabla(t)$
$h(\Delta)$	$=_{def}$	$\Delta$
$h(\Delta(t))$	$=_{def}$	$\Delta(t)$
$h(X(d))$	$=_{def}$	$X(d)$
$h(\sigma X(d:D := e). \varphi)$	$=_{def}$	$\sigma X(d:D := e). h(\varphi)$

### 1.3 The predicate formula normal form (PFNF)

**Definition 5** A predicate formula is said to be in Predicate Formula Normal Form (PFNF) if it has the following form:

$$\mathbf{Q}_1 v_1 : V_1 \cdots \mathbf{Q}_n v_n : V_n \cdot h \wedge \bigwedge_{i \in I} \left( g_i \implies \bigvee_{j \in J_i} X^j(e^j) \right)$$

where  $X^j \in \chi$  ( $\chi$  is a countable of sorted predicate variables),  $\mathbf{Q}_i \in \{\forall, \exists\}$ ,  $I$  is a (possibly empty) finite index set, each  $J_i$  is a non-empty finite index set, and  $h$  and every  $g_i$  are simple formulae.

Note that here  $J_i$  is used to index a set of occurrences of not necessarily different variables. For instance,  $(n > 0 \implies X(3) \vee X(5) \vee Y(6))$  is a formula complying to the definition of PFNF. So long as it does not lead to confusion, we stick to the convention to drop the typing of the quantified variables  $v_i$ . An algorithm to compute a PFNF is:

$$\begin{aligned} p(c) &=_{def} c \\ p(X(d)) &=_{def} X(d) \\ p(\forall x:D.\varphi) &=_{def} \forall x:D.p(\varphi) \\ p(\exists x:D.\varphi) &=_{def} \exists x:D.p(\varphi) \\ \\ p(\varphi \wedge \psi) &=_{def} \mathbf{Q}_1^\varphi \cdots \mathbf{Q}_{n^\varphi}^\varphi \mathbf{Q}_1^\psi \cdots \mathbf{Q}_{n^\psi}^\psi \cdot (h^\varphi \wedge h^\psi) \\ &\quad \wedge \bigwedge_{i \in I^\varphi \cup I^\psi} \left( g_i \implies \bigvee_{j \in J_i} X^j(e^j) \right) \\ \\ p(\varphi \vee \psi) &=_{def} \mathbf{Q}_1^\varphi \cdots \mathbf{Q}_{n^\varphi}^\varphi \mathbf{Q}_1^\psi \cdots \mathbf{Q}_{n^\psi}^\psi \cdot (h^\varphi \vee h^\psi) \\ &\quad \wedge \bigwedge_{i \in I^\varphi} \left( (-h^\psi \wedge g_i) \implies \bigvee_{j \in J_i} X^j(e^j) \right) \\ &\quad \wedge \bigwedge_{i \in I^\psi} \left( (-h^\varphi \wedge g_i) \implies \bigvee_{j \in J_i} X^j(e^j) \right) \\ &\quad \wedge \bigwedge_{i \in I^\varphi, k \in I^\psi} \left( (g_i \wedge g_k) \implies \bigvee_{j \in J_i, m \in J_k} X^j(e^j) \vee X^m(e^m) \right) \end{aligned}$$

where

$$\begin{aligned} p(\varphi) &= \mathbf{Q}_1^\varphi \cdots \mathbf{Q}_{n^\varphi}^\varphi \cdot h^\varphi \wedge \bigwedge_{i \in I^\varphi} \left( g_i \implies \bigvee_{j \in J_i} X^j(e^j) \right) \\ p(\psi) &= \mathbf{Q}_1^\psi \cdots \mathbf{Q}_{n^\psi}^\psi \cdot h^\psi \wedge \bigwedge_{i \in I^\psi} \left( g_i \implies \bigvee_{j \in J_i} X^j(e^j) \right), \end{aligned}$$

under the assumption that  $I^\varphi$  and  $I^\psi$  are disjoint, and  $v_i^\varphi \neq v_j^\psi$  for all  $i, j$ .

## 2 Transforming a state formula to a PBES

In this section we define the algorithm `pbcs_translate` that generates a PBES from a state formula and an LPD. Let  $\langle D_p, d_0, P \rangle$  be the LPD given by

$$\begin{aligned} \mathbf{proc} \ P(x:D_p) &= \sum_{i \in I} \sum_{y:E_i} c_i(x, y) \rightarrow a_i(f_i(x, y)) \cdot t_i(x, y) \cdot P(g_i(x, y)) \\ &+ \sum_{j \in J} \sum_{y:E_j} c_j(x, y) \rightarrow \delta \cdot t_j(x, y); \end{aligned}$$

where  $a_i(f_i(x, y))$  is a multiset of actions. Then we define

$$\mathbf{pbcs\_translate}(\sigma X(x_f : D_f := d). \varphi, \langle D_p, d_0, P \rangle) = \mathbf{E}(\varphi),$$

where the function  $\mathbf{E}$  is inductively defined using the tables below. The function  $\varphi$  has to be in positive normal form, i.e. it may not contain any  $\neg$  or  $\Rightarrow$  symbols. This is done using the function  $h$ , as given below. There is also an untimed variant of the algorithm, which can be obtained by removing all time references. A formula  $\varphi$  not of the form  $\sigma X(x_f : D_f := d). \varphi$  is first translated into  $\nu X(). \varphi$ . We assume that  $T : \mathbb{R}$  is a unique fresh time variable that is generated by the algorithm.

Let  $a = \{a_1, \dots, a_n\}$  and  $b = \{b_1, \dots, b_n\}$  be two multi actions. Let  $A$  be the set of all permutations  $[i_1, \dots, i_n]$  of  $[1, \dots, n]$  such that  $\text{name}(a_{i_k}) = \text{name}(b_{i_k})$  for  $k = 1 \dots n$ . Then we define the function  $\mathbf{Sat}$  as follows:

$$\begin{aligned} \mathbf{Sat}(a \cdot t, b) &=_{def} \begin{cases} \bigvee_{[i_1, \dots, i_n] \in A} \bigwedge_{k=1 \dots n} (a_{i_k} = b_{i_k}) & \text{if } A \neq \emptyset \\ false & \text{otherwise} \end{cases} \\ \mathbf{Sat}(a \cdot t, c) &=_{def} c \\ \mathbf{Sat}(a \cdot t, \alpha \cdot u) &=_{def} \mathbf{Sat}(a \cdot t, \alpha) \wedge t \approx u \\ \mathbf{Sat}(a \cdot t, \neg \alpha) &=_{def} \neg \mathbf{Sat}(a \cdot t, \alpha) \\ \mathbf{Sat}(a \cdot t, \alpha \wedge \beta) &=_{def} \mathbf{Sat}(a \cdot t, \alpha) \wedge \mathbf{Sat}(a \cdot t, \beta) \\ \mathbf{Sat}(a \cdot t, \alpha \vee \beta) &=_{def} \mathbf{Sat}(a \cdot t, \alpha) \vee \mathbf{Sat}(a \cdot t, \beta) \\ \mathbf{Sat}(a \cdot t, \alpha \Rightarrow \beta) &=_{def} \mathbf{Sat}(a \cdot t, \alpha) \Rightarrow \mathbf{Sat}(a \cdot t, \beta) \\ \mathbf{Sat}(a \cdot t, \forall x:D. \alpha) &=_{def} \forall y:D. (\mathbf{Sat}(a \cdot t, \alpha[x := y])) \\ \mathbf{Sat}(a \cdot t, \exists x:D. \alpha) &=_{def} \exists y:D. (\mathbf{Sat}(a \cdot t, \alpha[x := y])) \end{aligned}$$

$$\begin{aligned} \mathbf{Par}_{X,l}(c) &=_{def} \square \\ \mathbf{Par}_{X,l}(\neg \varphi) &=_{def} \mathbf{Par}_{X,l}(\varphi) \\ \mathbf{Par}_{X,l}(\varphi \wedge \psi) &=_{def} \mathbf{Par}_{X,l}(\varphi) + + \mathbf{Par}_{X,l}(\psi) \\ \mathbf{Par}_{X,l}(\varphi \vee \psi) &=_{def} \mathbf{Par}_{X,l}(\varphi) + + \mathbf{Par}_{X,l}(\psi) \\ \mathbf{Par}_{X,l}(\varphi \Rightarrow \psi) &=_{def} \mathbf{Par}_{X,l}(\varphi) + + \mathbf{Par}_{X,l}(\psi) \\ \mathbf{Par}_{X,l}([\alpha] \varphi) &=_{def} \mathbf{Par}_{X,l}(\varphi) \\ \mathbf{Par}_{X,l}(\langle \alpha \rangle \varphi) &=_{def} \mathbf{Par}_{X,l}(\varphi) \\ \mathbf{Par}_{X,l}(\forall x:D. \varphi) &=_{def} \mathbf{Par}_{X,l++[x:D]}(\varphi) \\ \mathbf{Par}_{X,l}(\exists x:D. \varphi) &=_{def} \mathbf{Par}_{X,l++[x:D]}(\varphi) \\ \mathbf{Par}_{X,l}(Y(d_f)) &=_{def} \square \\ \mathbf{Par}_{X,l}(\sigma Y(x_f:D_f := d). \varphi) &=_{def} \begin{cases} l & \text{if } Y = X \\ \mathbf{Par}_{X,l++[x_f:D_f]}(\varphi) & \text{if } Y \neq X \end{cases} \\ \mathbf{Par}_{X,l}(\nabla(t)) &=_{def} \square \\ \mathbf{Par}_{X,l}(\Delta(t)) &=_{def} \square \end{aligned}$$

$\mathbf{RHS}(c)$	$=_{def}$	$c$
$\mathbf{RHS}(\varphi \wedge \psi)$	$=_{def}$	$\mathbf{RHS}(\varphi) \wedge \mathbf{RHS}(\psi)$
$\mathbf{RHS}(\varphi \vee \psi)$	$=_{def}$	$\mathbf{RHS}(\varphi) \vee \mathbf{RHS}(\psi)$
$\mathbf{RHS}(\varphi \Rightarrow \psi)$	$=_{def}$	$\mathbf{RHS}(\neg\varphi) \vee \mathbf{RHS}(\psi)$
$\mathbf{RHS}(\forall x:D.\varphi)$	$=_{def}$	$\forall x:D.\mathbf{RHS}(\varphi)$
$\mathbf{RHS}(\exists x:D.\varphi)$	$=_{def}$	$\exists x:D.\mathbf{RHS}(\varphi)$
$\mathbf{RHS}([\alpha]\varphi)$	$=_{def}$	$\bigwedge_{i \in I} \forall y:E_i ((\mathbf{Sat}_{true}(a_i(f_i(x_p, y))^{t_i(x_p, y)}, \alpha) \wedge c_i(x_p, y) \wedge t_i(x_p, y) > T) \Rightarrow \mathbf{RHS}(\varphi)[T := t_i(x_p, y)] [x_p := g_i(x_p, y)])$
$\mathbf{RHS}(\langle \alpha \rangle \varphi)$	$=_{def}$	$\bigvee_{i \in I} \exists y:E_i ((\mathbf{Sat}_{true}(a_i(f_i(x_p, y))^{t_i(x_p, y)}, \alpha) \wedge c_i(x_p, y) \wedge t_i(x_p, y) > T \wedge \mathbf{RHS}(\varphi)[T := t_i(x_p, y)] [x_p := g_i(x_p, y)])$
$\mathbf{RHS}(X(d))$	$=_{def}$	$\tilde{X}(T, d, x_p, \mathbf{Par}_{X, \square}(\varphi_0))$
$\mathbf{RHS}(\sigma X(x_f:D_f := d). \varphi)$	$=_{def}$	$\tilde{X}(T, d, x_p, \mathbf{Par}_{X, \square}(\varphi_0))$
$\mathbf{RHS}(\nabla(t))$	$=_{def}$	$(\bigwedge_{i \in I \cup J} \forall y:E_i ((\neg c_i(x_p, y) \vee t > t_k(x_p, y))) \wedge t > T$
$\mathbf{RHS}(\Delta(t))$	$=_{def}$	$(\bigvee_{i \in I \cup J} \exists y:E_i ((c_i(x_p, y) \wedge t \leq t_k(x_p, y))) \vee t \leq T$
$\mathbf{RHS}(\neg c)$	$=_{def}$	$\neg \mathbf{RHS}(\neg c) = \neg c$
$\mathbf{RHS}(\neg \neg \varphi)$	$=_{def}$	$\mathbf{RHS}(\varphi)$
$\mathbf{RHS}(\neg(\varphi \wedge \psi))$	$=_{def}$	$\mathbf{RHS}(\neg\varphi) \vee \mathbf{RHS}(\neg\psi)$
$\mathbf{RHS}(\neg(\varphi \vee \psi))$	$=_{def}$	$\mathbf{RHS}(\neg\varphi) \wedge \mathbf{RHS}(\neg\psi)$
$\mathbf{RHS}(\neg(\varphi \Rightarrow \psi))$	$=_{def}$	$\mathbf{RHS}(\varphi) \wedge \mathbf{RHS}(\neg\psi)$
$\mathbf{RHS}(\neg(\forall x:D.\varphi))$	$=_{def}$	$\exists x:D.\mathbf{RHS}(\neg\varphi)$
$\mathbf{RHS}(\neg(\exists x:D.\varphi))$	$=_{def}$	$\forall x:D.\mathbf{RHS}(\neg\varphi)$
$\mathbf{RHS}(\neg([\alpha]\varphi))$	$=_{def}$	$\mathbf{RHS}(\langle \alpha \rangle (\neg\varphi))$
$\mathbf{RHS}(\neg(\langle \alpha \rangle \varphi))$	$=_{def}$	$\mathbf{RHS}([\alpha](\neg\varphi))$
$\mathbf{RHS}(\neg X(d))$	$=_{def}$	$\mathbf{RHS}(X(d))$
$\mathbf{RHS}(\neg(\sigma X(x_f:D_f := d). \varphi))$	$=_{def}$	$\mathbf{RHS}(\tilde{\sigma} X(x_f:D_f := d). (\neg\varphi[X := \neg X])) = \mathbf{RHS}((\sigma X(x_f:D_f := d). \neg\varphi))$
$\mathbf{RHS}(\neg \nabla(t))$	$=_{def}$	$\mathbf{RHS}(\Delta(t))$
$\mathbf{RHS}(\neg \Delta(t))$	$=_{def}$	$\mathbf{RHS}(\nabla(t))$

$$\begin{array}{lll}
\mathbf{E}(c) & =_{def} & \epsilon \\
\mathbf{E}(\varphi \wedge \psi) & =_{def} & \mathbf{E}(\varphi)\mathbf{E}(\psi) \\
\mathbf{E}(\varphi \vee \psi) & =_{def} & \mathbf{E}(\varphi)\mathbf{E}(\psi) \\
\mathbf{E}(\varphi \Rightarrow \psi) & =_{def} & \mathbf{E}(\neg\varphi)\mathbf{E}(\psi) \\
\mathbf{E}(\forall x:D.\varphi) & =_{def} & \mathbf{E}(\varphi) \\
\mathbf{E}(\exists x:D.\varphi) & =_{def} & \mathbf{E}(\varphi) \\
\mathbf{E}([\alpha]\varphi) & =_{def} & \mathbf{E}(\varphi) \\
\mathbf{E}(\langle\alpha\rangle\varphi) & =_{def} & \mathbf{E}(\varphi) \\
\mathbf{E}(\nabla) & =_{def} & \epsilon \\
\mathbf{E}(\nabla(t)) & =_{def} & \epsilon \\
\mathbf{E}(\Delta) & =_{def} & \epsilon \\
\mathbf{E}(\Delta(t)) & =_{def} & \epsilon \\
\mathbf{E}(X(d)) & =_{def} & \epsilon \\
\mathbf{E}(\sigma X(x_f:D_f := d). \varphi) & =_{def} & (\sigma \tilde{X}(T : \mathbb{R}, x_f:D_f, x_p:D_p, \mathbf{Par}_{X,[]}(\varphi_0)) = \mathbf{RHS}(\varphi) ) \mathbf{E}(\varphi) \\
\mathbf{E}(\neg c) & =_{def} & \epsilon \\
\mathbf{E}(\neg\neg\varphi) & =_{def} & \mathbf{E}(\varphi) \\
\mathbf{E}(\neg(\varphi \wedge \psi)) & =_{def} & \mathbf{E}(\neg\varphi)\mathbf{E}(\neg\psi) \\
\mathbf{E}(\neg(\varphi \vee \psi)) & =_{def} & \mathbf{E}(\neg\varphi)\mathbf{E}(\neg\psi) \\
\mathbf{E}(\neg(\varphi \Rightarrow \psi)) & =_{def} & \mathbf{E}(\varphi)\mathbf{E}(\neg\psi) \\
\mathbf{E}(\neg(\forall x:D.\varphi)) & =_{def} & \mathbf{E}(\neg\varphi) \\
\mathbf{E}(\neg(\exists x:D.\varphi)) & =_{def} & \mathbf{E}(\neg\varphi) \\
\mathbf{E}(\neg([\alpha]\varphi)) & =_{def} & \mathbf{E}(\neg\varphi) \\
\mathbf{E}(\neg(\langle\alpha\rangle\varphi)) & =_{def} & \mathbf{E}(\neg\varphi) \\
\mathbf{E}(\neg\nabla) & =_{def} & \epsilon \\
\mathbf{E}(\neg\nabla(t)) & =_{def} & \epsilon \\
\mathbf{E}(\neg\Delta) & =_{def} & \epsilon \\
\mathbf{E}(\neg\Delta(t)) & =_{def} & \epsilon \\
\mathbf{E}(\neg X(d)) & =_{def} & \epsilon \\
\mathbf{E}(\neg\sigma X(x_f:D_f := d). \varphi) & =_{def} & (\tilde{\sigma}\tilde{X}(T : \mathbb{R}, x_f:D_f, x_p:D_p, \mathbf{Par}_{X,[]}(\varphi_0)) = \mathbf{RHS}(\neg\varphi) [X := \neg X]) \mathbf{E}(\neg\varphi),
\end{array}$$

where  $\tilde{\sigma} = \mu$  if  $\sigma = \nu$  and  $\tilde{\sigma} = \nu$  if  $\sigma = \mu$  and  $\tilde{X}$  is a fresh predicate variable.

### 3 Bisimulation algorithms

Let

$$\begin{aligned} M(d) &= \sum_{i \in I_M} \sum_{e: E_i} c_i(d, e) \rightarrow a_i(d, e) \cdot M(g_i(d, e)) \\ S(d) &= \sum_{i \in I_S} \sum_{e: E_i} c_i(d, e) \rightarrow a_i(d, e) \cdot M(g_i(d, e)) \end{aligned}$$

be two linear processes, such that  $I_M \cap I_S = \emptyset$ .  $M$  is called the model and  $S$  the specification. The expression  $a_i(d, e)$  can be a multi-action, or have the special value  $\tau$ . We assume that there are no  $\delta$  summands. We define four pbes equation systems that express some kind of bisimulation equivalence between  $M$  and  $S$ .

**Branching Bisimulation**  $brbsim(M, S) = \nu E_2 \mu E_1$ , where

$$\begin{aligned} E_2 &:= \{X^{M,S}(d, d') = match^{M,S}(d, d') \wedge match^{S,M}(d', d), \\ &\quad X^{S,M}(d', d) = X^{M,S}(d, d')\} \\ E_1 &:= \{Y_i^{M,S}(d, d', e) = close_i^{M,S}(d, d', e) | i \in I_M, \\ &\quad Y_i^{S,M}(d', d, e) = close_i^{S,M}(d', d, e) | i \in I_S\} \end{aligned}$$

with for all  $i \in I_p$  and  $(p, q) \in \{(M, S), (S, M)\}$ :

$$\begin{aligned} match^{p,q}(d, d') &= \bigwedge_{i \in I_p} \forall e: E_i. (c_i(d, e) \Rightarrow Y_i^{p,q}(d, d', e)) \\ close_i^{p,q}(d, d', e) &= \bigvee_{\{j \in I_q | a_j = \tau\}} \exists e': E_j. (c_j(d', e') \wedge Y_i^{p,q}(d, g_j(d', e'), e)) \\ &\quad \vee (X^{p,q}(d, d') \wedge step_i^{p,q}(d, d', e)) \\ step_i^{p,q}(d, d', e) &= \begin{cases} a_i = \tau : & X^{p,q}(g_i(d, e), d') \vee \bigvee_{\{j \in I_q | a_j = \tau\}} \exists e': E_j. (c_j(d', e') \wedge X^{p,q}(g_i(d, e), g_j(d', e'))) \\ a_i \neq \tau : & \bigvee_{\{j \in I_q | a_j = a_i\}} \exists e': E_j. (c_j(d', e') \wedge (a_i(d, e) = a_j(d', e')) \wedge X^{p,q}(g_i(d, e), g_j(d', e'))) \end{cases} \end{aligned}$$

**Strong Bisimulation**  $sbsim(M, S) = \nu E$ , where

$$E := \{X^{M,S}(d, d') = match^{M,S}(d, d') \wedge match^{S,M}(d', d), \\ X^{S,M}(d', d) = X^{M,S}(d, d')\}$$

with for all  $i \in I_p$  and  $(p, q) \in \{(M, S), (S, M)\}$ :

$$\begin{aligned} match^{p,q}(d, d') &= \bigwedge_{i \in I_p} \forall e: E_i. (c_i(d, e) \Rightarrow step_i^{p,q}(d, d', e)) \\ step_i^{p,q}(d, d', e) &= \bigvee_{j \in I_q} \exists e': E_j. (c_j(d', e') \wedge (a_i(d, e) = a_j(d', e')) \wedge X^{p,q}(g_i(d, e), g_j(d', e'))) \end{aligned}$$

**Weak Bisimulation**  $wbsim(M, S) = \nu E_2 \mu E_1$ , where

$$\begin{aligned} E_3 &:= \{X^{M,S}(d, d') = match^{M,S}(d, d') \wedge match^{S,M}(d', d), \\ &\quad X^{S,M}(d', d) = X^{M,S}(d, d')\} \\ E_2 &:= \{Y_{1,i}^{M,S}(d, d', e) = close_{1,i}^{M,S}(d, d', e) | i \in I_M, \\ &\quad Y_{2,i}^{M,S}(d, d') = close_{2,i}^{M,S}(d, d') | i \in I_M, \\ &\quad Y_{1,i}^{S,M}(d', d, e) = close_{1,i}^{S,M}(d', d, e) | i \in I_S, \\ &\quad Y_{2,i}^{S,M}(d', d) = close_{2,i}^{S,M}(d', d) | i \in I_S\} \end{aligned}$$

with for all  $i \in I_p$  and  $(p, q) \in \{(M, S), (S, M)\}$ :

$$\begin{aligned}
match^{p,q}(d, d') &= \bigwedge_{i \in I_p} \forall e: E_i. (c_i(d, e) \Rightarrow Y_{1,i}^{p,q}(d, d', e)) \\
close_{1,i}^{p,q}(d, d', e) &= \left( \bigvee_{\{j \in I_q \mid a_j = \tau\}} \exists e': E_j. (c_j(d', e') \wedge Y_{1,i}^{p,q}(d, g_j(d', e'), e)) \right) \vee step_i^{p,q}(d, d', e) \\
step_i^{p,q}(d, d', e) &= \begin{cases} a_i = \tau : close_{2,i}^{p,q}(g_i(d, e), d') \\ a_i \neq \tau : \bigvee_{j \in I_q} \exists e': E_j. (c_j(d', e') \wedge a_i(d, e) = a_j(d', e') \wedge close_{2,i}^{p,q}(g_i(d, e), g_j(d', e'))) \end{cases} \\
close_{2,i}^{p,q}(d, d') &= X^{p,q}(d, d') \vee \bigvee_{\{j \in I_q \mid a_j = \tau\}} (\exists e': E_j. c_j(d', e') \wedge Y_{2,i}^{p,q}(d, g_j(d', e')))
\end{aligned}$$

**Branching Simulation Equivalence**  $brbsim(M, S) = \nu E_2 \mu E_1$ , where

$$\begin{aligned}
E_2 &:= \{X^{M,S}(d, d') = match^{M,S}(d, d') \wedge match^{S,M}(d', d), \\
&\quad X^{M,S}(d, d') = X^{S,M}(d', d), \\
&\quad X^{S,M}(d', d) = X^{M,S}(d, d')\} \\
E_1 &:= \{Y_i^{M,S}(d, d', e) = close_i^{M,S}(d, d', e) \mid i \in I_M, \\
&\quad Y_i^{S,M}(d', d, e) = close_i^{S,M}(d', d, e) \mid i \in I_S\}
\end{aligned}$$

with  $match$ ,  $close$ , and  $step$  defined exactly the same as in branching bisimulation.

## 4 PBES rewriters

In this section we describe two PBES rewriters. We assume that a data rewriter `datar` is given that rewrites data terms.

### 4.1 Simplifying rewriter

We define a simplifying PBES rewriter `pbesr` recursively as follows

$$\begin{aligned}
\text{pbesr}(b) &= \text{datar}(b) \\
\text{pbesr}(\neg\varphi) &= \neg\text{pbesr}(\varphi) \\
\text{pbesr}(\varphi \wedge \psi) &= \begin{cases} \text{false} & \text{if } \text{pbesr}(\varphi) = \text{false} \vee \text{pbesr}(\psi) = \text{false} \\ \text{pbesr}(\psi) & \text{if } \text{pbesr}(\varphi) = \text{true} \\ \text{pbesr}(\varphi) & \text{if } \text{pbesr}(\psi) = \text{true} \\ \text{pbesr}(\varphi) & \text{if } \text{pbesr}(\varphi) = \text{pbesr}(\psi) \\ \text{pbesr}(\varphi) \wedge \text{pbesr}(\psi) & \text{otherwise} \end{cases} \\
\text{pbesr}(\varphi \vee \psi) &= \begin{cases} \text{true} & \text{if } \text{pbesr}(\varphi) = \text{true} \vee \text{pbesr}(\psi) = \text{true} \\ \text{pbesr}(\psi) & \text{if } \text{pbesr}(\varphi) = \text{false} \\ \text{pbesr}(\varphi) & \text{if } \text{pbesr}(\psi) = \text{false} \\ \text{pbesr}(\varphi) & \text{if } \text{pbesr}(\varphi) = \text{pbesr}(\psi) \\ \text{pbesr}(\varphi) \vee \text{pbesr}(\psi) & \text{otherwise} \end{cases} \\
\text{pbesr}(\varphi \rightarrow \psi) &= \begin{cases} \text{pbesr}(\psi) & \text{if } \text{pbesr}(\varphi) = \text{true} \\ \text{true} & \text{if } \text{pbesr}(\varphi) = \text{false} \\ \text{true} & \text{if } \text{pbesr}(\psi) = \text{true} \\ \text{pbesr}(\neg\varphi) & \text{if } \text{pbesr}(\psi) = \text{false} \\ \text{true} & \text{if } \text{pbesr}(\varphi) = \text{pbesr}(\psi) \\ \text{pbesr}(\varphi) \rightarrow \text{pbesr}(\psi) & \text{otherwise} \end{cases} \\
\text{pbesr}(\forall d:D.\varphi) &= \begin{cases} \text{true} & \text{if } \text{pbesr}(\varphi) = \text{true} \\ \text{false} & \text{if } \text{pbesr}(\varphi) = \text{false} \text{ and } D \text{ is non-empty} \\ \text{pbesr}(\varphi) & \text{if } d \text{ does not occur in } \varphi \\ \forall d:D.\text{pbesr}(\varphi) & \text{otherwise} \end{cases} \\
\text{pbesr}(\exists d:D.\varphi) &= \begin{cases} \text{true} & \text{if } \text{pbesr}(\varphi) = \text{true} \text{ and } D \text{ is non-empty} \\ \text{false} & \text{if } \text{pbesr}(\varphi) = \text{false} \\ \text{pbesr}(\varphi) & \text{if } d \text{ does not occur in } \varphi \\ \exists d:D.\text{pbesr}(\varphi) & \text{otherwise} \end{cases} \\
\text{pbesr}(X(e)) &= X(\text{datar}(e))
\end{aligned}$$

where  $b$  is a data term of data sort  $\mathbb{B}$ ,  $\text{true}$  and  $\text{false}$  are elements of data sort  $\mathbb{B}$ ,  $X$  is a predicate variable,  $e$  consists of zero or more data sorts and  $d, d_1, d_2$  are data variables of sort  $D$ .

**Simplify** The pbes expression rewrite system SIMPLIFY [Luc Engelen, 2007] consists of the following rules<sup>1</sup>:

$$\begin{aligned}
false \wedge x &\rightarrow false \\
x \wedge false &\rightarrow false \\
true \wedge x &\rightarrow x \\
x \wedge true &\rightarrow x \\
\neg true &\rightarrow false \\
\neg false &\rightarrow true \\
ITE(true, x, y) &\rightarrow x \\
ITE(false, x, y) &\rightarrow y \\
x &= x \rightarrow true \\
y &= x \rightarrow x = y, \text{ provided } y \succ x
\end{aligned}$$

## 4.2 Quantifier Elimination Rewriter

This section describes a rewriter on predicate formulae that eliminates quantifiers. It is based on the following property

$$\forall_{x:X}.\varphi \equiv \bigwedge_{y:X} \varphi[x := y] \quad \exists_{x:X}.\varphi \equiv \bigvee_{y:X} \varphi[x := y],$$

where the conjunction and disjunction on the right hand sides may be infinite. Because of this, the rewriter we describe here is not guaranteed to terminate. However, in many practical cases the rewriter can compute a finite expression even if the quantifier variables are of infinite sort. An example of this is the formula  $\forall_{n:\mathbb{N}}.(n > 2) \vee X(n)$  that can be rewritten into  $X(0) \wedge X(1) \wedge X(2)$ .

We assume that the sorts of quantifier variables can be enumerated. By this we mean the existence of a function *enum* that maps an arbitrary term  $d : D$  to a finite set of terms  $\{d_1, \dots, d_k\}$ , such that  $range(d) = \bigcup_{i=1..k} range(d_i)$ , where  $range(d)$  is the set of closed terms obtained from  $d$  by substituting values for the free variables of  $d$ . For example, if natural numbers are represented by  $S^n(0)$ , with  $S$  a function that expresses the successor of a number, then possible enumerations of the term  $n$  are  $\{0, S(n')\}$  and  $\{0, S(0), S(S(n''))\}$ . Let  $id$  be the identity function and let  $\sigma[d_1 := e_1, \dots, d_n := e_n]$  be the function  $\sigma'$  with  $\sigma'(x) = e_i$  if  $x = d_i$  and  $\sigma'(x) = \sigma(x)$  otherwise.

A parameter of the ELIMINATEQUANTIFIERS algorithm is a rewriter  $R$  on quantifier free predicate formulae, that is expected to have the following properties:

$$\begin{aligned}
R(\perp) &= \perp \\
(R(t) &= R(t')) \Rightarrow t \simeq t',
\end{aligned}$$

---

<sup>1</sup>Todo: reformulate this rewrite system.

where  $t \simeq t'$  indicates that  $t$  and  $t'$  are equivalent.

```

ELIMINATEQUANTIFIERS( $Q_{d_1:D_1, \dots, d_n:D_n} \cdot \varphi, R$ )
if  $\text{freevars}(R(\varphi)) \cap \{d_1, \dots, d_n\} = \emptyset$  then return  $R(\varphi)$ 
 $V := \emptyset$ 
for  $i \in \{1, \dots, n\}$  do  $E_i := \{d_i\}$ 
do
  choose  $e_k \in E_k$ , such that  $\text{dvar}(e_k) \neq \emptyset$ 
   $E_k := E_k \setminus \{e_k\}$ 
  for  $e \in \text{enum}(e_k)$  :
     $W := \emptyset$ 
    for  $\sigma \in \{id[d_1 := f_1, \dots, d_{k-1} := f_{k-1}, d_k := e, d_{k+1} := f_{k+1}, \dots, d_n := f_n]$ 
       $\wedge f_i \in E_i \quad (i = 1, \dots, n)\}$  :
         $W := W \cup \{R(\sigma(\varphi))\}$ 
    if  $\text{stop}_Q \in W$  then return  $\text{stop}_Q$ 
     $V := V \cup \{w \in W \mid \text{dvar}(w) \subset \text{dvar}(\varphi)\}$ 
    if  $\{w \in W \mid \text{dvar}(w) \subsetneq \text{dvar}(\varphi)\} \neq \emptyset$  then  $E_k := E_k \cup \{e\}$ 
  rof
while  $\bigvee_{i \in \{1, \dots, n\}} E_i \neq \emptyset$ 
return  $\bigoplus_{v \in V} v$ ,

```

where  $\text{stop}_Q = \perp$  and  $\bigoplus = \bigwedge$  in case  $Q = \forall$ , and where  $\text{stop}_Q = \top$  and  $\bigoplus = \bigvee$  in case  $Q = \exists$ .

## 5 PBES instantiation

In this section we describe an implementation of the instantiation algorithm *Inst*. Let  $\mathcal{E} = (\sigma_1 X_1(d_1 : D_1) = \varphi_1) \cdots (\sigma_n X_n(d_n : D_n) = \varphi_n)$  be a PBES, and  $X_{init}(e_{init})$  an initial state. The algorithm **PBES2BES** uses instantiation to compute a BES. It takes two extra parameters, an injective function  $\rho$  that renames proposition variables to predicate variables, and a rewriter  $R$  that eliminates quantifiers from predicate formulae. This rewriter is described in the next section.

```

PBES2BES( $\mathcal{E}$ ,  $X_{init}(e_{init})$ ,  $R$ ,  $\rho$ )
for  $i := 1 \cdots n$  do  $\mathcal{E}_i := \epsilon$ 
 $todo := \{R(X_{init}(e_{init}))\}$ 
 $done := \emptyset$ 
while  $todo \neq \emptyset$  do
  choose  $X_k(e) \in todo$ 
   $todo := todo \setminus \{X_k(e)\}$ 
   $done := done \cup \{X_k(e)\}$ 
   $X^e := \rho(X_k(e))$ 
   $\psi^e := R(\varphi_k[d_k := e])$ 
   $\mathcal{E}_k := \mathcal{E}_k(\sigma_k X^e = \rho(\psi^e))$ 
   $todo := todo \cup \{Y(f) \in \text{occ}(\psi^e) \mid Y(f) \notin done\}$ 
return  $\mathcal{E}_1 \cdots \mathcal{E}_n$ ,

```

where  $\rho$  is extended from predicate variables to quantifier free predicate formulae using

$$\rho(b) =_{def} b \quad \rho(\varphi \oplus \psi) =_{def} \rho(\varphi) \oplus \rho(\psi)$$

## 6 Constant Parameter Detection and Elimination

Let  $\mathcal{E} = (\sigma_1 X_1(d_{X_1} : D_{X_1}) = \varphi_{X_1}) \cdots (\sigma_n X_n(d_{X_n} : D_{X_n}) = \varphi_{X_n})$  be a PBES, and  $\kappa$  an initial state and let  $\text{eval}$  be an evaluator function on data expressions. We denote the  $i$ -th element of a vector  $x$  as  $x[i]$ . Then we define the algorithm `PBESCONSTELM` as follows:

```

PBESCONSTELM( $\mathcal{E}$ ,  $\kappa$ ,  $\text{eval}$ )
for  $X \in \text{occ}(\mathcal{E})$  do  $c_X := d_X$ 
for  $X(e) \in \text{iocc}(\kappa)$  do  $c_X := \text{update}_X(c_X, \text{eval}(e[d_X := c_X]))$ 
 $\text{todo} := \text{occ}(\kappa)$ 
while  $\text{todo} \neq \emptyset$  do
  choose  $X \in \text{todo}$ 
   $\text{todo} := \text{todo} \setminus \{X\}$ 
  for  $Y(e) \in \text{iocc}(\varphi_X)$  do
    if  $\text{eval}(\text{do\_update}(Y(e)[d_X := c_X])) \neq \text{false}$  then
       $c'_Y := \text{update}_X(c_Y, \text{eval}(e[d_X := c_X]))$ 
      if  $c'_Y \neq c_Y$  then
         $c_Y := c'_Y$ 
         $\text{todo} := \text{todo} \cup \{Y\}$ 
   $\text{redundant\_variables} = \{X \in \text{occ}(\mathcal{E}) \mid c_X = d_X\}$ 
   $\text{redundant\_parameters} = \{(X, i) \mid c_X[i] \in D_X[i]\}$ 
return  $\{\text{redundant\_variables}, \text{redundant\_parameters}\}$ 

```

where  $\text{update}$  is defined as follows:

$$\text{update}_X(c, c') =_{\text{def}} c'', \text{ with } c''[i] = \begin{cases} c[i] & \text{if } c[i] = c'[i] \\ c[i] & \text{if } c[i] \notin D_X \cup \{d_X[i]\} \\ c'[i] & \text{if } c[i] = d_X[i] \wedge c'[i] \notin D_X \\ \text{fresh\_var}(D_X[i]) & \text{otherwise} \end{cases}$$

and where  $\text{do\_update}$  is a boolean function that determines whether an update should be performed. A safe choice for this function is the constant function  $\text{true}$ , but a more sophisticated choice can be made (see [Simon Janssen, 2008]).

## 7 Gauß Elimination

A predicate formula  $\varphi$  is defined by the following grammar:

$$\varphi ::= b|X(e)|\neg\varphi|\varphi \wedge \varphi|\varphi \vee \varphi|\varphi \rightarrow \varphi|\forall d : D.\varphi|\exists d : D.\varphi|true|false$$

where  $b$  is a data term of sort  $\mathbb{B}$ ,  $X$  is a predicate variable,  $d$  is a data variable of sort  $D$ ,  $e$  is a data term,  $true$  represents *true*, and  $false$  represents *false*.

**Definition 6** (*Predicate Variable Substitution*) Let  $\varphi, \psi$  be predicate formulae and  $X$  a predicate variable. Then we define  $\psi[\varphi/X]$  as the result of applying the substitution  $X := \varphi$  to the formula  $\psi$ . To make this more precise: suppose  $X$  is declared as  $X(d : D)$ , then any occurrence  $X(\bar{d})$  in  $\psi$  is replaced by  $\varphi[d := \bar{d}]$ .

**Lemma 7** (*Substitution*) Let  $\mathcal{E}$  be an equation system for which  $X, Y \notin \text{bnd}(\mathcal{E})$ , then:

$$(\sigma X(d : D) = \varphi)\mathcal{E}(\sigma' Y(e : E) = \psi) \equiv (\sigma X(d : D) = \varphi)[\psi/Y]\mathcal{E}(\sigma' Y(e : E) = \psi)$$

**Definition 8** (*Approximation*) Let  $\varphi, \psi$  be predicate formulae and  $X$  a predicate variable. We inductively define  $\psi[\varphi/X]^k$  as follows:

$$\begin{aligned}\psi[\varphi/X]^0 &\stackrel{\text{def}}{=} \varphi \\ \psi[\varphi/X]^{k+1} &\stackrel{\text{def}}{=} \psi[\varphi/X]^k\end{aligned}$$

Thus,  $\psi[\varphi/X]^k$  represents the result of recursively substituting  $\varphi$  for  $X$  in  $\psi$ .

**Lemma 9** (*Approximants as Solutions*) Let  $\varphi$  be a predicate formula and  $k \in \mathbb{N}$  be a natural number. Then

$$\begin{aligned}(\mu X(d : D) = \varphi[\text{false}/X]^k) &\Rightarrow (\mu X(d : D) = \varphi) \\ (\nu X(d : D) = \varphi) &\Rightarrow (\nu X(d : D) = \varphi[\text{true}/X]^k)\end{aligned}$$

**Lemma 10** (*Stable Approximants as Solutions*) Let  $\varphi$  be a predicate formula and  $k \in \mathbb{N}$  be a natural number. Then

$$\begin{aligned}\text{if } \varphi[\text{false}/X]^k &\longleftarrow \varphi[\text{false}/X]^{k+1} \text{ then } (\mu X(d : D) = \varphi[\text{false}/X]^k) \equiv (\mu X(d : D) = \varphi) \\ \text{if } \varphi[\text{true}/X]^k &\longleftarrow \varphi[\text{true}/X]^{k+1} \text{ then } (\nu X(d : D) = \varphi[\text{true}/X]^k) \equiv (\nu X(d : D) = \varphi)\end{aligned}$$

### 7.1 Gauß Elimination Algorithm

Let  $\mathcal{E}$  be an equation system of the form

$$\mathcal{E} = (\sigma_1 X_1(d_1 : D_1) = \varphi_1) \cdots (\sigma_n X_n(d_n : D_n) = \varphi_n),$$

and let  $r$  be a rewrite function that maps a pbes expression  $\varphi$  to an equivalent expression  $\varphi'$ .

Then we define

```
GAUSS ELIMINATION( $\mathcal{E}, r$ )
 $\mathcal{E}' := \mathcal{E}$ 
 $i := n$ 
while not  $i = 0$ 
do
   $(\sigma_i X_i = \psi_i) := \text{SOLVEEQUATION}(\sigma_i X_i = \varphi_i)$ 
   $\varphi_i := \psi_i$ 
   $\mathcal{E}' := \mathcal{E}'(\sigma_i X_i = \varphi_i)$ 
  for  $k = 1$  to  $i - 1$  do  $\varphi_k := r(\varphi_k[\varphi_i/X_i])$  od
   $i := i - 1$ 
od
return  $\mathcal{E}'$ 
```

Here SOLVEEQUATION is an algorithm that solves a pbes equation, such that the resulting equation has no reference to the predicate variable in its right hand side. An example of such a solve equation algorithm is APPROXIMATE.

```

APPROXIMATE( $\sigma X = \varphi$ )
 $j := 0$ 
if  $\sigma = \nu$  then  $\psi_0 := true$  else  $\psi_0 := false$ 
repeat
     $\psi_{j+1} := \varphi[\psi_j/X]$ 
     $j := j + 1$ 
until ( $\psi_j = \psi_{j+1}$ )
return  $\sigma X = \psi_j$ 

```

Also pattern matching algorithms exist for this. The GAUSS ELIMINATION algorithm solves the equation system  $\mathcal{E}$  for the predicate variable  $X_1$ . To solve the system  $\mathcal{E}$  for all variables, the algorithm has to be applied repeatedly.

## 7.2 Solving a BES

If the equation system  $\mathcal{E}$  is a BES (i.e. the predicate variables have no parameters), then the following simple approximate function can be used to solve it:

```

APPROXIMATE-BES( $\sigma X = \varphi$ )
if  $\sigma = \nu$  then  $\psi_0 := true$  else  $\psi_0 := false$ 
return SIMPLIFY( $\sigma X = \varphi[\psi_0/X]$ )

```

## ATerm format

< DataExpr >	$c$
StateTrue	$true$
StateFalse	$false$
StateNot(< StateFrm >)	$\neg\varphi$
StateAnd(< StateFrm >, < StateFrm >)	$\varphi \wedge \varphi$
StateOr(< StateFrm >, < StateFrm >)	$\varphi \vee \varphi$
StateImp(< StateFrm >, < StateFrm >)	$\varphi \Rightarrow \varphi$
StateForall(< DataVarId > +, < StateFrm >)	$\forall x:D.\varphi$
StateExists(< DataVarId > +, < StateFrm >)	$\exists x:D.\varphi$
StateMust(< RegFrm >, < StateFrm >)	$\langle \alpha \rangle \varphi$
StateMay(< RegFrm >, < StateFrm >)	$[\alpha] \varphi$
StateYaled	$\nabla$
StateYaledTimed(< DataExpr >)	$\nabla(t)$
StateDelay	$\Delta$
StateDelayTimed(< DataExpr >)	$\Delta(t)$
StateVar(< String >, < DataExpr > *)	$X(d)$
StateNu(< String >, < DataVarIdInit > *, < StateFrm >)	$\nu X(x:D := d).\varphi$
StateMu(< String >, < DataVarIdInit > *, < StateFrm >)	$\mu X(x:D := d).\varphi$

## Naming conventions

$\text{left}(\varphi \otimes \psi)$	$= \varphi$
$\text{right}(\varphi \otimes \psi)$	$= \psi$
$\text{arg}(\neg\varphi)$	$= \varphi$
$\text{arg}(\forall d : D.\varphi) = \text{arg}(\exists d : D.\varphi)$	$= \varphi$
$\text{var}(\forall d : D.\varphi) = \text{var}(\exists d : D.\varphi)$	$= d : D$
$\text{arg}(\langle \alpha \rangle \varphi) = \text{arg}([\alpha] \varphi)$	$= \varphi$
$\text{act}(\langle \alpha \rangle \varphi) = \text{act}([\alpha] \varphi)$	$= \alpha$
$\text{time}(\nabla(t)) = \text{time}(\Delta(t))$	$= t$
$\text{var}(X(d : D))$	$= d : D$
$\text{arg}(\sigma X(d : D := e).\varphi)$	$= \varphi$
$\text{name}(\sigma X(d : D := e).\varphi)$	$= X$
$\text{var}(\sigma X(d : D := e).\varphi)$	$= d : D$
$\text{val}(\sigma X(d : D := e).\varphi)$	$= e$

where  $\sigma$  is either  $\mu$  or  $\nu$ , and  $\otimes$  is either  $\wedge$ ,  $\vee$ , or  $\Rightarrow$ .