

# PBES Implementation Notes

Wieger Wesselink

January 19, 2019

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  $B$ ,  $X$  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 Well typedness constraints

### 1.1.1 well typedness constraints for PBES equations

- the binding variable parameters have unique names
- the names of the quantifier variables in the equation are disjoint with the binding variable parameter names
- within the scope of a quantifier variable in the formula, no other quantifier variables with the same name may occur

### 1.1.2 well typedness constraints for PBESs

- the sorts occurring in the global variables of the equations are declared in the data specification
- the sorts occurring in the binding variable parameters are declared in the data specification
- the sorts occurring in the quantifier variables of the equations are declared in the data specification
- the binding variables of the equations have unique names (well formedness)
- the global variables occurring in the equations are declared in the global variable specification
- the global variables occurring in the equations with the same name are identical
- the declared global variables and the quantifier variables occurring in the equations have different names
- the predicate variable instantiations occurring in the equations match with their declarations
- the predicate variable instantiation occurring in the initial state matches with the declaration
- the data specification is well typed

## 1.2 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>
$m(\mu X(d:D := e).\varphi)$	$=_{def}$	$m(\varphi)$
$m(\nu X(d:D := e).\varphi)$	$=_{def}$	$m(\varphi)$

### 1.3 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.

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

## 2 Parity game generator

Let  $\mathcal{E} = (\sigma_1 X_1(d_1 : D_1) = \varphi_1) \dots (\sigma_n X_n(d_n : D_n) = \varphi_n)$  be a PBES with initial state  $X_{init}(e_{init})$ , and let  $R : PbesTerm \rightarrow PbesTerm$  be a rewriter. The PBES must be in normal form, i.e. it may not contain negations or implications. The following algorithm computes a BES. The generated equations are in a restricted format, such that the BES can be taken as input for a parity game solver.

```

GENERATEBES( $\mathcal{E}$ ,  $X_{init}(e_{init})$ ,  $R$ )
result :=  $\{(\nu Y_{\top} = Y_{\top}), (\mu Y_{\perp} = Y_{\perp})\}$ 
visited :=  $\{R(X_{init}(e_{init}))\}$ 
explored :=  $\{\top, \perp\}$ 
while visited  $\neq \emptyset$  do
  choose  $\psi \in$  visited  $\setminus$  explored
  visited := visited  $\setminus$   $\{\psi\}$ 
  explored := explored  $\cup$   $\{\psi\}$ 
  if  $\{\psi = X_k(e)\}$  then
     $\xi := R(\varphi_k[d_k := e])$ 
  else
     $\xi := \psi$ 
  if  $\{\xi = X_k(e)\}$  then
    result := result  $\cup$   $(\sigma_{\psi} Y_{\psi} = Y_{\xi})$ 
     $\sigma_{\xi} := \sigma_k$ 
    visited := visited  $\cup$   $\{\xi\}$ 
  else if  $\{\xi = \bigwedge_{j \in J} \phi_j\}$  then
    result := result  $\cup$   $(\sigma_{\psi} Y_{\psi} = \bigwedge_{j \in J} Y_{\phi_j})$ 
    for  $j \in J$  do
      if  $\{\xi = X_k(e)\}$  then  $\sigma_{\phi_j} := \sigma_k$  else  $\sigma_{\phi_j} := \sigma_{\psi}$ 
      visited := visited  $\cup$   $\{\phi_j\}_{j \in J}$ 
  else if  $\{\xi = \bigvee_{j \in J} \phi_j\}$  then
    result := result  $\cup$   $(\sigma_{\psi} Y_{\psi} = \bigvee_{j \in J} Y_{\phi_j})$ 
    for  $j \in J$  do
      if  $\{\xi = X_k(e)\}$  then  $\sigma_{\phi_j} := \sigma_k$  else  $\sigma_{\phi_j} := \sigma_{\psi}$ 
      visited := visited  $\cup$   $\{\phi_j\}_{j \in J}$ 
  else if  $\{\xi = \top\}$  then
    result := result  $\cup$   $(\sigma_{\psi} Y_{\psi} = Y_{\top})$ 
  else if  $\{\xi = \perp\}$  then
    result := result  $\cup$   $(\sigma_{\psi} Y_{\psi} = Y_{\perp})$ 
return result

```

In every step of the while loop the equation for  $Y_{\psi}$  is computed. If the right hand side of the equation for  $Y_{\psi}$  is a propositional variable instantiation, it is expanded into the right hand side of the corresponding PBES equation. Otherwise it is converted into a disjunction or conjunction by introducing new BES variables. The rewriter  $R$  is expected to eliminate all quantifiers, so the while loop does not contain cases for handling them. The order of the equations in the BES is significant. Therefore in the implementation instead of fixpoint symbols  $\sigma_{\psi}$  priority values are used. The BES variables  $Y_{\psi}$  are represented by integers.

An alternative for inserting the equations  $(\nu Y_{\top} = Y_{\top})$  and  $(\mu Y_{\perp} = Y_{\perp})$  at the beginning of the resulting BES is to replace  $\mu Y_{\psi} = \top$  and  $\nu Y_{\psi} = \top$  by  $\nu Y_{\psi} = Y_{\psi}$ , and to replace  $\mu Y_{\psi} = \perp$  and  $\nu Y_{\psi} = \perp$  by  $\mu Y_{\psi} = Y_{\psi}$ . This eliminates the need to introduce special equations for true and false.

This algorithm is implemented in the class `parity_game_generator`. The choice for  $\psi$  in the while loop is left to the user of the class.

### 3 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. Here, every  $d_{X_i}$  represents a vector of parameters. Furthermore, let  $\hat{X}(\hat{e})$  be an initial state and let *eval* be an evaluator function on data expressions. We denote the  $i$ -th element of a vector  $x$  as  $x[i]$ . We also use mappings: for a mapping  $c$ , the image of  $i$  is denoted  $c[i]$ . The empty mapping is denoted with  $\emptyset$  and the image of an element not present in a mapping is  $\perp$ . Note that  $\emptyset[i] = \perp$  for all  $i$ . Then we define the algorithm PBESCONSTELM as follows:

```

PBESCONSTELM( $\mathcal{E}$ ,  $\hat{X}(\hat{e})$ , eval)
for  $X \in \text{bnd}(\mathcal{E})$  do  $c_X := \emptyset$ 
 $c_{\hat{X}} := \text{update}(c_{\hat{X}}, \text{eval}(\hat{e}))$ 
 $todo := \{\hat{X}\}$ 
while  $todo \neq \emptyset$  do
  choose  $X \in todo$ 
   $todo := todo \setminus \{X\}$ 
  for  $Y(e) \in \text{iocc}(\varphi_X)$  do
    if  $\text{eval}(\text{needs\_update}(Y(e), \varphi_X)[d_X := c_X]) \neq \text{false}$  then
       $c'_Y := \text{update}(c_Y, \text{eval}(e[d_X := c_X]))$ 
      if  $c'_Y \neq c_Y$  then
         $c_Y := c'_Y$ 
         $todo := todo \cup \{Y\}$ 
   $\text{constant\_parameters} := \{(X, i) \mid c_X[i] \neq d_X[i]\}$ 
for  $i := 1 \cdots n$  do  $\varphi_{X_i} := \varphi_{X_i}[d_{X_i} := c_{X_i}]$ 
return  $\text{constant\_parameters}$ 

```

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

$$\text{update}(c, e) =_{\text{def}} c', \text{ with } c'[i] = \begin{cases} \perp & \text{if } c = \emptyset \text{ and } e = [] \\ e[i] & \text{if } c = \emptyset \text{ and } e[i] \text{ is constant} \\ c[i] & \text{if } e[i] = c[i] \\ d_X[i] & \text{otherwise} \end{cases}$$

and where *needs\_update* is a boolean function that determines whether an update should be performed. A safe choice for this function is the constant function *true*. In [Simon Janssen, 2008] the following alternative is proposed.

Let  $c_T$  and  $c_F$  be defined as

$$\begin{aligned}
c_T(c) &= c \\
c_F(c) &= \neg c \\
c_T(\neg\varphi) &= c_F(\varphi) \\
c_F(\neg\varphi) &= c_T(\varphi) \\
c_T(\mathbf{X}(e)) &= \text{false} \\
c_T(\mathbf{Q}d : D.\varphi) &= \mathbf{Q}d : D.c_T(\varphi) \\
c_T(\varphi \wedge \psi) &= c_T(\varphi) \wedge c_T(\psi) \\
c_T(\varphi \vee \psi) &= c_T(\varphi) \vee c_T(\psi) \\
c_T(\varphi \Rightarrow \psi) &= c_F(\varphi) \vee c_T(\psi) \\
c_F(\varphi \wedge \psi) &= c_F(\varphi) \vee c_F(\psi) \\
c_F(\varphi \vee \psi) &= c_F(\varphi) \wedge c_F(\psi) \\
c_F(\varphi \Rightarrow \psi) &= c_T(\varphi) \wedge c_F(\psi)
\end{aligned}$$

and let the multi sets  $cond_T$  and  $cond_F$  be defined as

$$\begin{aligned}
cond_{\Gamma}(X(e), c) &= \emptyset \\
cond_{\Gamma}(X(e), Y(f)) &= \emptyset \\
cond_{\Gamma}(X(e), \neg\varphi) &= cond_{\Gamma}(X(e), \varphi) \\
cond_{\Gamma}(X(e), \varphi \oplus \psi) &= \begin{cases} \{c_{\Gamma}(\varphi \oplus \psi)\} \cup cond_{\Gamma}(X(e), \varphi) & \text{if } X(e) \in \text{iocc}(\varphi) \\ \{c_{\Gamma}(\varphi \oplus \psi)\} \cup cond_{\Gamma}(X(e), \psi) & \text{if } X(e) \in \text{iocc}(\psi) \\ \emptyset & \text{otherwise} \end{cases} \\
cond_{\Gamma}(X(e), Qd : D.\varphi) &= \begin{cases} \{c_{\Gamma}(Qd : D.\varphi)\} \cup \{Qd : D.\theta \mid \theta \in cond_{\Gamma}(X(e), \varphi)\} & \text{if } X(e) \in \text{iocc}(\varphi) \\ \emptyset & \text{otherwise} \end{cases}
\end{aligned}$$

with  $\Gamma \in \{T, F\}$  and  $Q \in \{\forall, \exists\}$ . Then we define

$$needs\_update(X(e), \varphi) = \bigwedge_{c \in cond_T(X(e), \varphi) \cup cond_F(X(e), \varphi)} \neg c$$

**Remark 5** Note that a direct implementation of this function is extremely inefficient. It is expected that a more suitable representation can be found.

## 4 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

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

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

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

### 4.1 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( $\mathcal{E}, r, \text{SOLVE}$ )
 $\mathcal{E}' := \mathcal{E}$ 
 $i := n$ 
while not  $i = 0$ 
do
   $\varphi_i := \text{SOLVE}(\sigma_i X_i = \varphi_i)$ 
   $\mathcal{E}' := \mathcal{E}'(\sigma_i X_i = \varphi_i)$ 
  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( $\sigma X = \varphi$ , COMPARE)
 $j := 0$ 
if  $\sigma = \nu$  then  $\psi_0 := true$  else  $\psi_0 := false$ 
repeat
     $\psi_{j+1} := \varphi[\psi_j/X]$ 
     $j := j + 1$ 
until COMPARE( $\psi_j, \psi_{j+1}$ )
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.

## 4.2 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( $\sigma X = \varphi[\psi_0/X]$ )

```

## 5 Small progress measures

Let  $\mathcal{E} = (\sigma_1 X_1 = \varphi_1) \cdots (\sigma_n X_n = \varphi_n)$  be a BES in standard recursive form. Let  $d$  be the number of  $\mu$ -blocks appearing in  $\mathcal{E}$ . For each variable  $X_i$  we define a corresponding attribute  $\alpha_i \in \mathbb{N}^d$ , which is called the 'progress measure' of  $X_i$ . We define the algorithm SMALLPROGRESSMEASURES as follows:

```

SMALLPROGRESSMEASURES( $\mathcal{E}$ )
 $V := \{X_i \mid 1 \leq i \leq n\}$ 
 $V_{Even} := \{X_i \in V \mid \varphi_i \text{ is a disjunction}\}$ 
 $V_{Odd} := \{X_i \in V \mid \varphi_i \text{ is not a disjunction}\}$ 
for  $i := 1 \cdots n$  do  $\alpha_i := [0, \dots, 0]$ 
while liftable_variables( $V$ )  $\neq \emptyset$  do
  choose  $X_i \in \text{liftable\_variables}(V)$ 
   $\alpha_i := \begin{cases} \min\{\gamma \mid X_j \in \text{occ}(\varphi_i) \wedge \gamma = f(X_j, \text{rank}(X_i))\} & \text{if } X_i \in V_{Even} \\ \max\{\gamma \mid X_j \in \text{occ}(\varphi_i) \wedge \gamma = f(X_j, \text{rank}(X_i))\} & \text{if } X_i \in V_{Odd} \end{cases}$ 
return ( $W_{Even}, W_{Odd}$ )

```

where  $\min/\max$  is the minimum/maximum with respect to the lexicographical order on  $\mathbb{N}^d$ , and  $\beta \in \mathbb{N}^d$  is defined as

$$\beta_i = \begin{cases} 0 & \text{if } i \text{ is even} \\ |\{X_j \in V \mid \text{rank}(X_j) = i\}| & \text{if } i \text{ is odd} \end{cases}$$

and the function  $\text{inc} : \mathbb{N}^d \times \mathbb{Z} \rightarrow \mathbb{N}^d$  is defined inductively as

$$\begin{cases} \text{inc}(\alpha, -1) & = \top \\ \text{inc}(\alpha, i) & = \begin{cases} \text{inc}([\alpha_0, \dots, \alpha_{i-1}, 0, \alpha_{i+1}, \dots, \alpha_d], i-1) & \text{if } \alpha_i = \beta_i \\ [\alpha_0, \dots, \alpha_{i-1}, \alpha_i + 1, \alpha_{i+1}, \dots, \alpha_d] & \text{otherwise} \end{cases} \end{cases}$$

and the function  $f : \mathbb{N}^d \times \mathbb{N} \rightarrow \mathbb{N}^d$  is defined as:

$$f(X_j, m) = \begin{cases} [(\alpha_j)_1, \dots, (\alpha_j)_m, 0, \dots, 0] & \text{if } m \text{ is even} \\ \text{inc}([\alpha_j)_1, \dots, (\alpha_j)_m, 0, \dots, 0], m) & \text{if } m \text{ is odd} \end{cases}$$

and

$$\text{liftable\_variables}(V) = \{X_i \in V \mid \min\{\alpha \mid w \in \text{occ}(\varphi_i) \wedge \alpha = f(w)\} < \alpha_i\}.$$

and  $W_{Even}$  and  $W_{Odd}$  are defined as

$$\begin{cases} W_{Even} & = \{X_i \in V \mid \alpha_i < \top\} \\ W_{Odd} & = \{X_i \in V \mid \alpha_i = \top\} \end{cases}$$

and the function  $\text{rank}$  is defined inductively as follows:

$$\begin{cases} \text{rank}(X_1) & = \begin{cases} 0 & \text{if } \sigma_1 = \nu \\ 1 & \text{if } \sigma_1 = \mu \end{cases} \\ \text{rank}(X_{i+1}) & = \begin{cases} \text{rank}(X_i) & \text{if } \sigma_{i+1} = \sigma_i \\ \text{rank}(X_i) + 1 & \text{if } \sigma_{i+1} \neq \sigma_i \end{cases} \end{cases}$$

## 6 Stategraph

This section describes the implementation of the tool pbesstategraph.

### 6.1 Definitions

We denote the number of predicate variable instances occurring in a predicate formula  $\varphi$  by  $\text{npred}(\varphi)$ . We assume that predicate variable instances in  $\varphi$  are assigned a unique natural number between 1 and  $\text{npred}(\varphi)$ .

**Definition 11** *Let  $\varphi$  be a predicate formula and let  $i$  be between 1 and  $\text{npred}(\varphi)$ . The functions  $\text{pred}(\varphi, i)$ ,  $\text{data}(\varphi, i)$  and  $\text{PVI}(\varphi, i)$  are such that the predicate variable instance  $\text{PVI}(\varphi, i)$  is the  $i^{\text{th}}$  predicate variable instance in  $\varphi$ , syntactically present as  $\text{pred}(\varphi, i)(\text{data}(\varphi, i))$ . Let  $\psi$  be a predicate formula. We write  $\varphi[i \rightarrow \psi]$  to indicate that the predicate variable instance at position  $i$  is replaced syntactically by  $\psi$  in  $\varphi$ .*

**Definition 12** *Let  $\varphi$  be a predicate formula. We define the guard of predicate variable instantiation  $\text{PVI}(\varphi, i)$  for  $i \leq \text{npred}(\varphi)$  inductively as follows:*

$$\begin{aligned}
\text{guard}^i(c) &= \text{false} \\
\text{guard}^i(Y) &= \text{true} \\
\text{guard}^i(\forall d : D.\varphi) &= \text{guard}^i(\varphi) \\
\text{guard}^i(\exists d : D.\varphi) &= \text{guard}^i(\varphi) \\
\text{guard}^i(\varphi \wedge \psi) &= \begin{cases} s(\varphi) \wedge \text{guard}^{i-\text{npred}(\varphi)}(\psi) & \text{if } i > \text{npred}(\varphi) \\ s(\psi) \wedge \text{guard}^i(\varphi) & \text{if } i \leq \text{npred}(\varphi) \end{cases} \\
\text{guard}^i(\varphi \vee \psi) &= \begin{cases} n(\varphi) \wedge \text{guard}^i(\psi) & \text{if } i > \text{npred}(\varphi) \\ ns(\psi) \wedge \text{guard}^i(\varphi) & \text{if } i \leq \text{npred}(\varphi) \end{cases}
\end{aligned}$$

where

$$\begin{aligned}
s(\varphi) &= \begin{cases} \varphi & \text{if } \text{npred}(\varphi) = 0 \\ \text{true} & \text{otherwise} \end{cases} \\
ns(\varphi) &= \begin{cases} \neg\varphi & \text{if } \text{npred}(\varphi) = 0 \\ \text{true} & \text{otherwise} \end{cases}
\end{aligned}$$

We define the function  $\text{sig}$  for computing significant variables recursively as follows:

$$\begin{aligned}
\text{sig}(b) &= \text{FV}(b) \\
\text{sig}(\varphi \wedge \psi) &= \text{sig}(\varphi) \cup \text{sig}(\psi) \\
\text{sig}(\varphi \vee \psi) &= \text{sig}(\varphi) \cup \text{sig}(\psi) \\
\text{sig}(X(e)) &= \emptyset \\
\text{sig}(\exists d : D.\varphi) &= \text{sig}(\varphi) \setminus \{d\} \\
\text{sig}(\forall d : D.\varphi) &= \text{sig}(\varphi) \setminus \{d\} \\
\text{sig}(\varphi \Rightarrow \psi) &= \text{sig}(\varphi) \cup \text{sig}(\psi) \\
\text{sig}(\neg\varphi) &= \text{sig}(\varphi)
\end{aligned}$$

#### 6.1.1 The functions source, target and copy

Let  $X(d : D) = \varphi$  be a PBES equation. Let source be a function with the property that

$$\text{source}(X, i, j) = \begin{cases} e & \text{if } \text{guard}^i(\text{PVI}(\varphi_X, i)) \Rightarrow d[j] \approx e \\ \perp & \text{otherwise} \end{cases}$$

A possible heuristic for obtaining a source function is to look for positive occurrences of constraints of the form  $d[j] \approx e$  in the guards; these can be used to define the source function. Let  $\text{sigma}(X, i)$  be the substitution defined as

$$\text{sigma}(X, i)(v) = \begin{cases} e & \text{if } \text{source}(X, i, j) = e \text{ for some } j \\ v & \text{otherwise} \end{cases}$$

We define the function *target* as follows:

$$\text{target}(X, i, j) = \begin{cases} c & \text{if } \text{rewrite}(\text{sigma}(X, i)(\text{PVI}(\varphi, i)))[j] = c \\ \perp & \text{otherwise} \end{cases}$$

with  $c$  a constant. We define the function *copy* as follows:

$$\text{copy}(X, i, j) = \begin{cases} k & \text{if } \text{PVI}(\varphi, i)[k] = d[j] \\ \perp & \text{otherwise} \end{cases}$$

We define the function *used* as follows:

$$\text{used}(X, i, j) = d_X[j] \in FV(\text{guard}^i(\text{PVI}(\varphi_X, i)))$$

We define the function *changed* as follows:

$$\text{changed}(X, i, j) = \text{pred}(\varphi_X, i) = X \wedge d_X[j] \neq \text{data}(\varphi_X, i)[j]$$

Let  $\text{par}(X)$  be the set of parameters of the equation corresponding to  $X$ . Let  $\text{pos}(X, i)$  denote the  $i$ -th parameter of the equation corresponding to  $X$ .

## 6.2 Control flow parameters

Control flow parameters are computed in phases. First the function *LCFP* is computed, then the function *GCFP*, and finally they are related using  $\sim$ .

### 6.2.1 LCFP computation

There are two versions of the computation of LCFP.

```

COMPUTELocalControlFlowParametersDefault( $\mathcal{E}$ )
for  $X \in \text{bnd}(\mathcal{E})$  do
  for  $n = 1, \dots, |\text{par}(X)|$  do
     $LCFP(X, n) := \text{true}$ 
for  $X \in \text{bnd}(\mathcal{E})$  do
  for  $i = 1, \dots, \text{npred}(\varphi_X)$  do
    if  $\text{pred}(\varphi_X, i) = X$  then
      for  $n = 1, \dots, |\text{par}(X)|$  do
        if  $\text{source}(X, i, n) = \perp \wedge \text{target}(X, i, n) = \perp \wedge \text{copy}(X, i, n) \neq n$  then
           $LCFP(X, n) := \text{false}$ 
return  $LCFP$ 

```

```

COMPUTELocalControlFlowParametersAlternative( $\mathcal{E}$ )
for  $X \in \text{bnd}(\mathcal{E})$  do
  for  $n = 1, \dots, |\text{par}(X)|$  do
     $LCFP(X, n) := \text{true}$ 
for  $X \in \text{bnd}(\mathcal{E})$  do
  for  $i = 1, \dots, \text{npred}(\varphi_X)$  do
    if  $\text{pred}(\varphi_X, i) = X$  then
      for  $n = 1, \dots, |\text{par}(X)|$  do
        if  $(\text{source}(X, i, n) = \perp \wedge \text{target}(X, i, n) = \perp \wedge \text{copy}(X, i, n) \neq \perp)$ 
        or  $(\text{source}(X, i, n) \neq \perp \wedge \text{target}(X, i, n) \neq \perp \wedge \text{copy}(X, i, n) = \perp)$ 
        then
           $LCFP(X, n) := \text{false}$ 
return  $LCFP$ 

```

### 6.2.2 GCFP computation

```

COMPUTEGlobalControlFlowParameters( $\mathcal{E}, LCFP$ )
for  $X \in \text{bnd}(\mathcal{E})$  do
  for  $n = 1, \dots, |\text{par}(X)|$  do
     $GCFP(X, n) := LCFP(X, n)$ 
for  $X \in \text{bnd}(\mathcal{E})$  do
  for  $i = 1, \dots, \text{npred}(\varphi_X)$  do
    let  $Y = \text{pred}(\varphi_X, i)$ 
    if  $Y \neq X$  then
      for  $n = 1, \dots, |\text{par}(X)|$  do
        if  $\text{target}(X, i, n) = \perp \wedge \forall m : \text{copy}(X, i, m) \neq n$  then
           $GCFP(X, n) := \text{false}$ 
return  $GCFP$ 

```

### 6.2.3 Related control flow parameters

GCFP parameters can be related using the relation  $\sim$ . There are two versions of the computation of  $\sim$ .

```

COMPUTERELATEDGLOBALCONTROLFLOWPARAMETERSDEFAULT( $\mathcal{E}$ ,  $GCFP$ )
for  $X \in bnd(\mathcal{E})$  do
  for  $i = 1, \dots, npred(\varphi_X)$  do
    let  $Y = pred(\varphi_X, i)$ 
    for  $n = 1, \dots, |par(X)|$  do
      if  $copy(X, i, n) = m \neq \perp$  then
        if  $GCFP(X, n) \wedge GCFP(Y, m)$  then  $(X, n) \sim (Y, m)$ 

```

```

COMPUTERELATEDGLOBALCONTROLFLOWPARAMETERSALTERNATIVE( $\mathcal{E}$ ,  $GCFP$ )
for  $X \in bnd(\mathcal{E})$  do
  for  $i = 1, \dots, npred(\varphi_X)$  do
    let  $Y = pred(\varphi_X, i)$ 
    for  $n = 1, \dots, |par(X)|$  do
      if  $copy(X, i, n) = m \neq \perp$  then
        if  $GCFP(X, n) \wedge GCFP(Y, m) \wedge target(X, i, m) = \perp$  then  $(X, n) \sim (Y, m)$ 

```

### 6.2.4 Control flow graphs

The symmetric closure  $\sim_S$  of the relation  $\sim$  defines an undirected graph  $(V, \sim_S)$  on the set of vertices  $V = \{(X, i) \mid X \in bnd(\mathcal{E}) \wedge 1 \leq i \leq |d_X|\}$ . This graph is called the global control flow graph. The connected components in this graph are the local control flow graphs. A local control flow graph is called *invalid* if it contains two vertices  $(X, i)$  and  $(X, j)$  with  $i \neq j$ .

## 6.3 Global algorithm

### 6.3.1 Global control flow graph

The following algorithm computes the global control flow graph.

```

COMPUTEGLOBALCONTROLFLOWGRAPH( $\mathcal{E}$ ,  $X_{init}(e_{init})$ )
 $V := \emptyset$ 
 $E := \emptyset$ 
 $todo := \{(X_{init}, e) \mid \forall k \leq |c_X| : e[k] = e_{init}[\downarrow_X k]\}$ 
while  $todo \neq \emptyset$  do
  choose  $u \in todo$ 
   $todo := todo \setminus \{u\}$ 
   $V := V \cup \{u\}$ 
  for  $i = 1 \dots \text{npred}(\varphi)$  do
    if  $\text{ENABLEDEDGE}(u, i)$ 
       $v := \text{COMPUTEVERTEX}(u, i, \text{PVI}(\varphi_X, i))$ 
      if  $v \notin V$ 
         $V := V \cup \{v\}$ 
         $todo := todo \cup \{v\}$ 
       $E := E \cup \{(u, i, v)\}$ 
return  $(V, E)$ 

ENABLEDEDGE( $u, i$ )
let  $u = (X, e)$ 
for  $k = 1 \dots |c_X|$  do
  if  $\text{source}(X, i, \downarrow_X k) \neq \perp \wedge \text{source}(X, i, \downarrow_X k) \neq e_k$  then
    return false
return true

COMPUTEVERTEX( $u, i, Y(f)$ )
let  $u = (X, e)$ 
for  $l = 1 \dots |c_Y|$  do
   $q := \text{target}(X, i, \downarrow_Y l)$ 
  if  $q = \perp$  then
    choose  $k$  such that  $\text{copy}(X, i, \downarrow_X k) = \downarrow_Y l$ 
     $v_l := e_k$ 
  else
     $v_l := q$ 
return  $(Y, v)$ 

```

Note that  $\text{copy}(X, i, \downarrow_Y l) = \downarrow_X k$  implies that parameters  $(X, \downarrow_X k)$  and  $(Y, \downarrow_Y l)$  are related.

**Remark 13** In the code  $\downarrow_X k$  is represented by  $\text{cfp\_X}[k]$

**Remark 14** In the code  $\text{copy}(X, i, \downarrow_X k)$  is represented by  $Yf.\text{copy}(k)$ , where  $Yf = \text{PVI}(\varphi_X, i)$

### 6.3.2 Global control flow marking

The following algorithm computes the function *marking* that denotes which parameters are marked in a vertex of the control flow graph  $(V, E)$ .

```

COMPUTEMARKINGGLOBAL( $\mathcal{E}, X_{init}(e_{init}), V, E$ )
for  $u = X(e) \in V$  do  $marking(u) := sig(u) \cap par(X)$ 
 $todo := V$ 
while  $todo \neq \emptyset$  do
  choose  $v = X(e) \in todo$ 
   $todo := todo \setminus \{v\}$ 
  for  $(u, v) \in E$  do
    let  $X(f) = label(u, v)$ 
    for  $d_X[j] \in marking(v)$ 
       $M := (FV(f[j]) \setminus marking(u)) \cap par(X)$ 
      if  $M \neq \emptyset$ 
         $marking(u) := marking(u) \cup M$ 
         $todo := todo \cup \{u\}$ 

```

### 6.3.3 Global reset variables

Let  $PVI(\varphi_X, i) = Y(e)$  and let  $V$  be the global control flow graph. Then we define

```

RESETVARIABLEGLOBAL( $Y(e), i, V$ )
 $\varphi := true$ 
for  $u = Y(f) \in V$  do
   $c := true$ 
   $k := 1$ 
  for  $j = 1 \dots |par(Y)|$  do
     $r := \perp$ 
    if  $CFP(Y, j)$  then
      if  $target(X, i, j) = \perp$ 
         $c := c \wedge (e[j] = f[k])$ 
         $r := r \triangleleft f[k]$ 
      else if  $e[j] \in marking(u)$  then
         $r := r \triangleleft e[j]$ 
      else
         $r := r \triangleleft default\_value(e[j])$ 
     $k := k + 1$ 
   $\varphi := \varphi \wedge (c \Rightarrow Y(r))$ 
return  $\varphi$ 

```



## 6.4 Compute values

Let  $C$  be a component containing related CFPs.

```
COMPUTEVALUES( $\mathcal{E}$ ,  $X_{init}(e_{init})$ ,  $C$ )
result :=  $\emptyset$ 
for  $(X, j) \in C$  do
    if  $(X = X_{init})$  then result := result  $\cup$   $\{e_{init}[j]\}$ 
for  $(X, k) \in C$  do
    for  $i = 1 \cdots \text{npred}(\varphi_X)$  do
        if  $\text{source}(X, i, k) = v$  then result := result  $\cup$   $\{v\}$ 
for  $(Y, k) \in C$  do
    for  $i = 1 \cdots \text{npred}(\varphi_Y)$  do
        if  $\text{pred}(\varphi_Y, i) = Y \wedge \text{target}(Y, i, k) = v$  then result := result  $\cup$   $\{v\}$ 
return result
```

## 6.5 Local algorithm

### 6.5.1 Local control flow graph

Given a GCFG  $(W, \sim)$  and a component  $C \subseteq W$ , then we define  $U$  as  $\{(X, i, d = e) \mid (X, i, d) \in C \wedge e \in \text{COMPUTEVALUES}(C)\} \cup \{(X_{init}, ?, ? = ?) \mid \forall i : (X_{init}, i) \notin C\}$ . Note that this algorithm potentially extends the graph with 'undefined' nodes in a lazy fashion.

```

COMPUTELocalControlFlowGraph( $U, C$ )
 $V, E := U, \emptyset$ 
 $todo := U$ 
while  $todo \neq \emptyset$  do
  choose  $u = (X, k, d = e) \in todo$ 
   $todo := todo \setminus \{u\}$ 
  for  $i = 1 \dots \text{npred}(\varphi_X)$  do
    let  $Y = \text{pred}(\varphi_X, i)$ 
    if  $d = ?$  then
      if  $(Y, k') \in C$  for some  $k'$  then
        if  $\text{target}(X, i, k') = e'$  then
           $v := (Y, k', d_Y[k'] = e')$ 
           $\text{insert}(V, E, todo, u, i, v)$ 
        else
          if  $X \neq Y$  then
             $v := (Y, ?, ? = ?)$ 
             $\text{insert}(V, E, todo, u, i, v)$ 
      else
        if  $(Y, k') \in C$  for some  $k'$  then
          if  $(\text{source}(X, i, k) = e \wedge \text{target}(X, i, k') = e')$  then  $\text{insert}(V, E, todo, u, i, (Y, k', d_Y[k'] = e'))$ 
          else if  $(Y \neq X \wedge \text{source}(X, i, k) = \perp \wedge \text{target}(X, i, k') = e')$  then  $\text{insert}(V, E, todo, u, i, (Y, k', d_Y[k'] = e))$ 
          else if  $(Y \neq X \wedge \text{source}(X, i, k) = \perp \wedge \text{copy}(X, i, k) = k')$  then  $\text{insert}(V, E, todo, u, i, (Y, k', d_Y[k'] = e))$ 
        else
           $v := (Y, ?, ? = ?)$ 
           $\text{insert}(V, E, todo, u, i, v)$ 
    return  $(V, E)$ ,

```

where  $\text{insert}(V, E, todo, u, i, v)$  is shorthand for the statements

```

if  $v \notin todo \wedge v \notin V$  then  $todo := todo \cup \{v\}$ 
 $V := V \cup \{v\}$ 
 $E := E \cup \{(u, i, v)\}$ 

```

### 6.5.2 Local belongs relation

Let  $(V, \longrightarrow)$  be a local control flow graph, and rules be a predicate defined as

$$\text{rules}(V, X, i) = \exists_{(X, j, e) \in V} : (X, j, e) \xrightarrow{i}.$$

```

COMPUTEBELONGS(  $V, \longrightarrow$ , rules)
 $B_k := \emptyset$ 
for  $(X, j, v) \in V$  do
   $belongs := \{m \mid d_X[m] \text{ is a data parameter of } X\}$ 
  for  $i = 1 \dots \text{npred}(\varphi)$  do
    for  $m \in belongs$  do
      if  $(\text{used}(X, i, m) \vee \text{changed}(X, i, m)) \wedge \neg \text{rules}(V, X, i)$  then
         $belongs := belongs \setminus \{m\}$ 
       $B_k := B_k \cup \{(X, d_X[m]) \mid m \in belongs\}$ 
return  $B_k$ 

```

### 6.5.3 Local control flow marking

Let  $(V, E)$  be a local control flow graph, and  $B$  the corresponding belongs relation.

```

UPDATERULE1( $B, u, i, v$ )
let  $u = (X, n, d_X[n] = z)$ 
let  $v = (Y, m, d_Y[m] = w)$ 
let  $Y(e) = \text{PVI}(\varphi_X, i)$ 
 $M := \emptyset$ 
for  $d_Y[l] \in \text{marking}(v)$  do
   $M := M \cup (\text{FV}(\text{rewr}(e[l], [d_X[n] := z])) \cap \{d \mid (X, d) \in B\})$ 
return  $\text{marking}(u) \cup M$ 

UPDATERULE2( $B, u, i, v$ )
let  $u = (X, n, d_X[n] = z)$ 
let  $v = (Y, m, d_Y[m] = w)$ 
let  $Y(e) = \text{PVI}(\varphi_X, i)$ 
 $M := \emptyset$ 
for  $d_Y[l] \in \text{marking}(v)$  do
  if  $(Y, d_Y[l]) \notin B$  then
     $M := M \cup (\text{FV}(\text{rewr}(e[l], [d_X[n] := z])) \cap \{d \mid (X, d) \in B\})$ 
return  $\text{marking}(u) \cup M$ 

```

```

COMPUTEMARKINGLOCAL( $\mathcal{E}, (V_1, E_1, B_1), \dots, (V_J, E_J, B_J)$ )
for  $j = 1 \dots J$  do
  for  $u = (X, n, d_X[n] = z) \in V_j$  do
     $marking(u) := significant(u) \cap \{d \mid (X, d) \in B_j\}$ 
 $stable := false$ 
while  $\neg stable$  do
   $stableint := false$ 
  while  $\neg stableint$  do
     $stableint := true$ 
    for  $j = 1 \dots J$  do
       $todo := V_j$ 
      while  $todo \neq \emptyset$  do
        choose  $u = (X, n, d_X[n] = z) \in todo$ 
         $todo := todo \setminus \{u\}$ 
        if  $marking(u) = \{d \mid (X, d) \in B_j\}$  then continue
        for  $(u, i, v) \in E_j$  do
           $m := marking(u)$ 
           $marking(u) := UPDATEMARKINGRULE1(B_j, u, i, v)$ 
          if  $m \neq marking(u)$  then
             $todo := todo \cup \{v' \mid \exists i' : (v', i', u) \in E_j\}$ 
             $stableint := false$ 
       $stableext := false$ 
      while  $\neg stableext$  do
         $stableext := true$ 
        for  $j = 1 \dots J$  do
          for  $u = (X, n, d_X[n] = z) \in V_j$  do
            if  $marking(u) = \{d \mid (X, d) \in B_j\}$  then continue
            for  $(u, i, w) \in E_j$  do
              let  $Y(e) = PVI(\varphi_X, i)$ 
              for  $k = 1 \dots J$  do
                if  $k \neq j$  then
                  for  $v = (Y, m, d_Y[m] = w) \in V_k$  do
                    if  $\exists v' = (X, m', d_X[m'] = w) \in V_k$  such that  $(v', i, v) \in E_k$  then
                       $m := marking(u)$ 
                       $marking(u) := UPDATEMARKINGRULE2(B_j, u, i, v)$ 
                      if  $m \neq marking(u)$  then
                         $stableint := false$ 
                         $stableext := false$ 
           $stable := stableint \wedge stableext$ 

```

#### 6.5.4 Local reset variables

```

RESETVARIABLELOCAL( $i, \sigma X(d_X) = \varphi_X, V_1, \dots, V_J, B_1, \dots, B_J, \text{rules}$ )
let  $Y(e) = \text{PVI}(\varphi_X, i)$ 
 $e' := e$ 
for  $k = 1 \dots |e|$  do
  if  $\text{CFP}(Y, k)$  then continue
   $\text{relevant} := \text{true}$ 
   $\text{condition} := \{\}$ 
  for  $j = 1 \dots J$  do
    if  $\text{rules}(V_j, X, i)$  then
      let  $p, q$  be such that  $(Y, p, q) \in V_j$ 
      if  $\text{target}(X, i, p) \neq \perp$  then
        let  $q' = \text{target}(X, i, p)$ 
         $\text{relevant} := \text{relevant} \wedge ((Y, d_Y[k]) \in B_j \Rightarrow d_Y[k] \in \text{marking}(Y, p, q'))$ 
      else
         $\text{relevant} := \text{relevant} \wedge ((Y, d_Y[k]) \in B_j \Rightarrow \exists r : d_Y[k] \in \text{marking}(Y, p, r))$ 
         $\text{condition} += \{e[p] = r \mid (Y, d_Y[k]) \in B_j \wedge (Y, p, r) \in V_j \wedge d_Y[k] \notin \text{marking}(Y, p, r)\}$ 
    if  $\neg \text{relevant}$  then
       $e'[k] := \text{default\_value}(d_Y[k])$ 
    else if  $\text{condition} \neq \{\}$  then
       $e'[k] := \text{if}(\text{join\_or}(\text{condition}), \text{default\_value}(d_Y[k]), e[k])$ 
return  $Y(e')$ 

```

## 7 ReachableVariables

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 let  $X_{init}(e_{init})$  be the initial state. The algorithm REACHABLEVARIABLES computes the reachable predicate variables.

```
REACHABLEVARIABLES( $\mathcal{E}, X_{init}$ )
visited :=  $\{X_{init}\}$ ; explored :=  $\emptyset$ 
while visited  $\neq \emptyset$ 
  choose  $X_i \in$  visited
  visited := visited  $\setminus \{X_i\}$ 
  explored := explored  $\cup \{X_i\}$ 
  for each  $X_j(e) \in \text{iocc}(\varphi_{X_i})$ 
    if  $X_j \notin$  explored
      visited := visited  $\cup \{X_j\}$ 
return explored
```

## 8

### ATerm format

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

### Naming conventions

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

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