PBES instantiation and solving

Wieger Wesselink, Tim Willemse

January 19, 2019

This document describes instantiation and solving algorithms for PBESs that are used in the tools **pbesinst** and **pbessolve**.

1 Finite algorithm

In this section we describe an implementation of the finite instantiation algorithm PbesInstFinite that eliminates data parameters with finite sorts. It is implemented in the tool **pbesinst**. Let $\mathcal{E} = (\sigma_1 X_1(d_1 : D_1, e_1 : E_1) = \varphi_1) \cdots (\sigma_n X_n(d_n : D_n, e_n : E_n) = \varphi_n)$ be a PBES. We assume that all data sorts D_i are finite and all data sorts E_i are infinite. Let r be a data rewriter, and let ρ be an injective function that creates a unique predicate variable from a predicate variable name and a data value according to $\rho(X(d : D, e : E), d_0) \to Y(e : E)$, where D is finite and E is infinite and $d_0 \in D$. Note that D and D_i may be multi-dimensional sorts.

PbesInstFinite($\mathcal{E}$, r, ρ)
for $i := 1 \cdots n$ **do**
    $\mathcal{E}_i := \{\sigma_i \rho(X_i, d) = R(\varphi_k[d_k := d]) \mid d \in D_i\}$
return $\mathcal{E}_1 \cdots \mathcal{E}_n$,

with R a rewriter on pbes expressions that is defined as follows:

$$
\begin{aligned}
R(b) &= b \\
R(\neg\varphi) &= \neg R(\varphi) \\
R(\varphi \oplus \psi) &= R(\varphi) \oplus R(\psi) \\
R(X_i(d,e)) &= \begin{cases} \rho(X_i, r(d))(r(e)) & \text{if } FV(d) = \emptyset \\ \bigvee\limits_{d_i \in D_i} r(d = d_i) \wedge \rho(X_i, d_i)(r(e)) & \text{if } FV(d) \neq \emptyset \end{cases} \\
R(\forall_{d:D}.\varphi) &= \forall_{d:D}.R(\varphi) \\
R(\exists_{d:D}.\varphi) &= \exists_{d:D}.R(\varphi)
\end{aligned}
$$

where $\oplus \in \{\vee, \wedge, \Rightarrow\}$, b a data expression and φ and ψ pbes expressions and $FV(d)$ is the set of free variables appearing in d.

2 Lazy algorithm

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

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

where ρ is extended from predicate variables to quantifier free predicate formulae using

$$\begin{aligned} \rho(b) &= b \\ \rho(\varphi \oplus \psi) &= \rho(\varphi) \oplus \rho(\psi) \end{aligned}$$

3 Generic lazy algorithms

In this section two generic variants of lazy PBES instantiation are described that report all discovered BES equations using a callback function ReportEquation. These versions are later extended to compute structure graphs.

The first version PbesInstLazy1 maintains a collection *done*, that contains all BES variables for which an equation has been computed.

PbesInstLazy1($\mathcal{E}$, $X_{init}(e_{init})$, R)
$init := R(X_{init}(e_{init}))$
$todo := \{init\}$
$done := \emptyset$
while $todo \neq \emptyset$ **do**
  choose $X_k(e) \in todo$
  $todo := todo \setminus \{X_k(e)\}$
  $done := done \cup \{X_k(e)\}$
  $\psi^e := R(\varphi_k[d_k := e])$
  ReportEquation$(X_k(e), \psi^e)$
  $todo := todo \cup (\mathrm{occ}(\psi^e) \setminus done)$

The second version PbesInstLazy2 maintains a set *discovered* instead of *done*. This set contains BES variables that have been discovered, but for which the corresponding equation may not have been computed yet. The sets are related via $done = discovered \setminus todo$.

PbesInstLazy2($\mathcal{E}$, $X_{init}(e_{init})$, R)
$init := R(X_{init}(e_{init}))$
$todo := \{init\}$
$discovered := \{init\}$
while $todo \neq \emptyset$ **do**
  choose $X_k(e) \in todo$
  $todo := todo \setminus \{X_k(e)\}$
  $\psi^e := R(\varphi_k[d_k := e])$
  ReportEquation$(X_k(e), \psi^e)$
  $todo := todo \cup (\mathrm{occ}(\psi^e) \setminus discovered)$
  $discovered := discovered \ \cup \mathrm{occ}(\psi^e)$

It turned out that the second version has slightly better performance for some larger use cases, so the second version is implemented in the tool **pbessolve**.

To support breadth first and depth first search, the implementation stores the set *todo* as a double ended queue. New elements are always appended to *todo*. In the case of breadth first search always the first element is chosen, and in the case of depth first search the last element.

4 Structure graphs

A structure graph is a tuple (V, E, d, r) with V a set of nodes containing BES variables, E a set of edges, $r : V \to \mathbb{N}$ a partial function that assigns a rank to each node, and $d : V \to \{\blacktriangle, \blacktriangledown, \top, \bot\}$ a partial function that assigns a decoration to each node. A structure graph is formally defined using the following SOS rules:

$$\frac{X \in bnd(\mathcal{E})}{r(X) = rank_{\mathcal{E}}(X)}$$

$$\frac{}{d(true) = \top} \qquad \frac{}{d(false) = \bot}$$

$$\frac{}{d(f \wedge f') = \blacktriangle} \qquad \frac{}{d(f \vee f') = \blacktriangledown}$$

$$\frac{f\blacktriangle \quad f \to g}{(f \wedge f') \to g} \qquad \frac{f\blacktriangle \quad f \to g}{(f' \wedge f) \to g}$$

$$\frac{f\blacktriangledown \quad f \to g}{(f \vee f') \to g} \qquad \frac{f\blacktriangledown \quad f \to g}{(f' \vee f) \to g}$$

$$\frac{\neg f\blacktriangle}{f \wedge f' \to f} \qquad \frac{\neg f'\blacktriangle}{f \wedge f' \to f'}$$

$$\frac{\neg f\blacktriangledown}{f \vee f' \to f} \qquad \frac{\neg f'\blacktriangledown}{f \vee f' \to f'}$$

$$\frac{}{X \wedge f \to X} \qquad \frac{}{f \wedge X \to X}$$

$$\frac{}{X \vee f \to X} \qquad \frac{}{f \vee X \to X}$$

$$\frac{\sigma X = f \wedge f' \in \mathcal{E}}{d(X) = \blacktriangle} \qquad \frac{\sigma X = f \vee f' \in \mathcal{E}}{d(X) = \blacktriangledown}$$

$$\frac{\sigma X = Y \in \mathcal{E}}{X \to Y} \qquad \frac{\sigma X = \top \in \mathcal{E}}{X \to \top} \qquad \frac{\sigma X = \bot \in \mathcal{E}}{X \to \bot}$$

$$\frac{\sigma X = f \wedge f' \in \mathcal{E} \quad f \wedge f' \to g}{X \to g}$$

$$\frac{\sigma X = f \vee f' \in \mathcal{E} \quad f \vee f' \to g}{X \to g}$$

Note that in this definition separate nodes are created for the left hand side X and the right hand side f of each equation $\sigma X = f$. This is undesirable, hence in implementations usually the nodes X and f are merged into one node labeled with X.

4.1 Attractor sets

Let $A \subseteq V$ be a subset of vertices of a structure graph $G = (V, E, d, r)$. We define the following algorithms for computing an attractor set of A. The value $\alpha = 0$ corresponds with disjunction and $\alpha = 1$ with conjunction.

```
ATTR(A, α)
todo := ⋃_{u∈A} (pred(u) \ A)
while todo ≠ ∅ do
    choose u ∈ todo
    todo := todo \ {u}
    if d(u) = α ∨ succ(u) ⊆ A then
        A := A ∪ {u}
        todo := todo ∪ (pred(u) \ A)
return A
```

where

$$\begin{aligned} pred(v) &= \{u \in V \mid (u, v) \in E\} \\ succ(u) &= \{v \in V \mid (u, v) \in E\} \end{aligned}$$

4.2 Recursive procedure for solving structure graphs

Let $G = (V, E, d, r)$ be a structure graph. The following algorithm is used to compute a partitioning of V into (W_0, W_1) of vertices W_0 that represent equations evaluating to true and vertices W_1 that represent equations evaluating to false. A precondition of this algorithm is that it contains no nodes with decoration $\top$ or $\bot$. This algorithm is based on Zielonka's recursive algorithm.

SolveRecursive(V)
if $V = \emptyset$ **then return** $(\emptyset, \emptyset)$
$m := \min(\{r(v) \mid v \in V\})$
$\alpha := m \mod 2$
$U := \{v \in V \mid r(v) = m\}$
$A := \text{Attr}(U, \alpha)$
$W'_0, W'_1 := \text{SolveRecursive}(V \setminus A)$
if $W'_{1-\alpha} = \emptyset$ **then**
    $W_\alpha, W_{1-\alpha} := A \cup W'_\alpha, \emptyset$
else
    $B := \text{Attr}\left(W'_{1-\alpha}, 1-\alpha\right)$
    $W_0, W_1 := \text{SolveRecursive}(V \setminus B)$
    $W_{1-\alpha} := W_{1-\alpha} \cup B$
return W_0, W_1

where

$$succ(u, U) = \begin{cases} succ(u) \cap U & \text{if } succ(u) \cap U \neq \emptyset \\ succ(u) & \text{otherwise} \end{cases}$$

Tom van Dijk has introduced an optimization that may reduce the number of recursive calls. Note that this optimized version does not compute a complete strategy, so it cannot be used for counter example generation(!)

SolveRecursive(V)
if $V = \emptyset$ **then return** $(\emptyset, \emptyset)$
$m := \min(\{r(v) \mid v \in V\})$
$\alpha := m \mod 2$
$U := \{v \in V \mid r(v) = m\}$
$A := \text{Attr}(U, \alpha)$
$W'_0, W'_1 := \text{SolveRecursive}(V \setminus A)$
$B := \text{Attr}\left(W'_{1-\alpha}, 1-\alpha\right)$
if $W'_{1-\alpha} = B$ **then**
    $W_\alpha, W_{1-\alpha} := A \cup W'_\alpha, B$
else
    $W_0, W_1 := \text{SolveRecursive}(V \setminus B)$
    $W_{1-\alpha} := W_{1-\alpha} \cup B$
return W_0, W_1

The algorithm can be extended to sets V containing nodes with decoration $\top$ or $\bot$ as follows:

SolveRecursiveExtended(V)
$V_1 := \text{Attr}(\{v \in V \mid d(v) = \bot\}, 1)$
$V_0 := \text{Attr}(\{v \in V \mid d(v) = \top\}, 0)$
$(W_0, W_1) := \text{SolveRecursive}(V \setminus (V_0 \cup V_1))$
return $(W_0 \cup V_1, W_0 \cup V_1)$

Another possible optimization of the SolveRecursive algorithm is to insert the following shortcuts. This doesn't seem to have much effect in practice, so currently it isn't enabled in the code.

SolveRecursive(V)
if $V = \emptyset$ **then return** $(\emptyset, \emptyset)$
$m := \min(\{r(v) \mid v \in V\})$
$\alpha := m \bmod 2$
$U := \{v \in V \mid r(v) = m\}$
if $h = m \wedge \mathit{even}(m)$ **then return** $(\emptyset, V)$
if $h = m \wedge \mathit{odd}(m)$ **then return** $(V, \emptyset)$
$A := \text{Attr}\,(U, \alpha)$
$W'_0, W'_1 := \text{SolveRecursive}(V \setminus A)$
if $W'_{1-\alpha} = \emptyset$ **then**
    $W_\alpha, W_{1-\alpha} := A \cup W'_\alpha, \emptyset$
else
    $B := \text{Attr}\,\big(W'_{1-\alpha}, 1-\alpha\big)$
    $W_0, W_1 := \text{SolveRecursive}(V \setminus B)$
    $W_{1-\alpha} := W_{1-\alpha} \cup B$
return W_0, W_1

4.3 Computing a winning strategy

The SolveRecursive algorithm can as a side effect produce a mapping τ that corresponds to a winning strategy, by slightly adapting the algorithm and the attractor set computation:

$\mathit{precondition} : V \subseteq \mathit{dom}(r) \cup \mathit{dom}(d)$
SolveRecursive(V)
if $V = \emptyset$ **then return** $(\emptyset, \emptyset)$
$m := \min(\{r(v) \mid v \in V\})$
$\alpha := m \mod 2$
$U := \{v \in V \mid r(v) = m\}$
for $u \in U$ **if** $d(u) = \alpha \wedge \mathit{succ}(u) \neq \emptyset$ **then** $\tau[u] := v$ **with** $v \in \mathit{succ}(u)$
$A := \text{Attr}\,(U, \alpha)$
$W'_0, W'_1 := \text{SolveRecursive}(V \setminus A)$
$B := \text{Attr}\,\big(W'_{1-\alpha}, 1-\alpha\big)$
if $W'_{1-\alpha} = B$ **then**
    $W_\alpha, W_{1-\alpha} := A \cup W'_\alpha, B$
else
    $W_0, W_1 := \text{SolveRecursive}(V \setminus B)$
    $W_{1-\alpha} := W_{1-\alpha} \cup B$
return W_0, W_1

with

```
ATTR(A, α)
todo := ⋃_{u∈A} (pred(u) \ A)
while todo ≠ ∅ do
    choose u ∈ todo
    todo := todo \ {u}
    if d(u) = α ∨ succ(u) ⊆ A then
        if d(u) ≠ 1 − α then τ[u] := v with v ∈ A ∩ succ(u)
        A := A ∪ {u}
        todo := todo ∪ (pred(u) \ A)
return A
```

Note that the mapping τ is a global variable of ATTR, which is ugly. In the implementation the strategy is an attribute of the nodes.

5 Structure graph based PBES instantiation

In the tool **pbessolve** an extension called PBESINSTSTRUCTUREGRAPH of the PBESINSTLAZY2 algorithm is implemented that builds a structure graph from the reported equations. On top of it several optimizations to this algorithm are defined. For readability, we only present the full algorithms here, and highlight the changes with respect to previous versions. A graph G is represented as a tuple (V, E) with V the set of vertices and E the set of edges.

```
PBESINSTSTRUCTUREGRAPH(ℰ, X_init(e_init), R)
init := R(X_init(e_init))
todo := {init}
discovered := {init}
(V, E) := (∅, ∅)
while todo ≠ ∅ do
    choose X_k(e) ∈ todo
    todo := todo \ {X_k(e)}
    ψ^e := R(φ_k[d_k := e])
    (V, E) := (V, E) ∪ SG^0(X_k(e), ψ^e)
    todo := todo ∪ (occ(ψ^e) \ discovered)
    discovered := discovered ∪ occ(ψ^e)
return G
```

where SG^0 and SG^1 are defined as

ψ	$SG^0(\varphi, \psi)$
$true$	$(\{\varphi\}, \emptyset)$
$false$	$(\{\varphi\}, \emptyset)$
Y	$(\{\varphi, \psi\}, \{(\varphi, \psi)\})$
$\psi_1 \wedge \cdots \wedge \psi_n$	$(\{\varphi, \psi_1, \cdots, \psi_n\}, \{(\varphi, \psi_1), \cdots, (\varphi, \psi_n)\}) \cup \bigcup_{i=1}^{n} SG^1(\psi_i)$
$\psi_1 \vee \cdots \vee \psi_n$	$(\{\varphi, \psi_1, \cdots, \psi_n\}, \{(\varphi, \psi_1), \cdots, (\varphi, \psi_n)\}) \cup \bigcup_{i=1}^{n} SG^1(\psi_i)$

ψ	$SG^1(\varphi)$
$true$	$(\{\psi\}, \emptyset)$
$false$	$(\{\psi\}, \emptyset)$
Y	$(\{\psi\}, \emptyset)$
$\psi_1 \wedge \cdots \wedge \psi_n$	$(\{\psi, \psi_1, \cdots, \psi_n\}, \{(\psi, \psi_1), \cdots, (\psi, \psi_n)\}) \cup \bigcup_{i=1}^{n} SG^1(\psi_i)$
$\psi_1 \vee \cdots \vee \psi_n$	$(\{\psi, \psi_1, \cdots, \psi_n\}, \{(\psi, \psi_1), \cdots, (\psi, \psi_n)\}) \cup \bigcup_{i=1}^{n} SG^1(\psi_i),$

where we assume that in $\psi_1 \wedge \cdots \wedge \psi_n$ none of the ψ_i is a conjunction, and in $\psi_1 \vee \cdots \vee \psi_n$ none of the ψ_i is a disjunction. Note that both $SG^0(\varphi, \psi)$ and $SG^1(\varphi, \psi)$ are defined as a pair (V, E) of nodes and edges.

5.1 Optimisation 1

The lemma below indicates that one can simplify the BES equation that is being created without affecting the solution to the BES.

Lemma 1 *The solution to all variables in a BES $\mathcal{E}(\sigma X = f)\mathcal{E}'$ is equivalent to the solution to those variables in the BES $\mathcal{E}(\sigma X = f[X := b_\sigma])\mathcal{E}'$, where $b_\sigma = true$ if $\sigma = \nu$ and $b_\sigma = false$ if $\sigma = \mu$.*

Using this lemma, rather than creating a structure graph underlying the equation $\sigma X_e = \psi^e$, we can create a structure graph for $\sigma X_e = \psi^e[X^e := b_\sigma]$. This can be done by adding the following assignment below the assignment $\psi^e := R(\varphi_k[d_k := e])$:

$$\psi^e := R(\psi^e[X^e := b_\sigma])$$

This leads to the following adaptations in the code:

PbesinstStructureGraph$_1(\mathcal{E}, X_{init}(e_{init}), R)$
$init := R(X_{init}(e_{init}))$
$todo := \{init\}$
$discovered := \{init\}$
$(V, E) := (\emptyset, \emptyset)$
while $todo \neq \emptyset$ **do**
  choose $X_k(e) \in todo$
  $todo := todo \setminus \{X_k(e)\}$
  $\psi^e := R(\varphi_k[d_k := e])$
  $\psi^e :=$ **if** $\sigma_k = \mu$ **then** $R(\psi^e[X^e := false])$ **else** $R(\psi^e[X^e := true])$
  $(V, E) := (V, E) \cup SG^0(X^e, \psi^e)$
  $todo := todo \cup (\mathrm{occ}(\psi^e) \setminus discovered)$
  $discovered := discovered \cup \mathrm{occ}(\psi^e)$
return G,

5.2 Optimisation 2

Our second optimisation exploits the fact that some of the BES equations that are generated while exploring the PBES are already solved (possibly after using optimisation 1). We first introduce some additional notation. Let (V, E) be a (partial) structure graph underlying the PBES $\mathcal{E}$. By S_0 we denote the set of vertices that represent equations with solution $true$, whereas S_1 denotes the set of vertices representing equations with solution $false$. Let π be a partial function that maps vertices to the propositional variables they represent (and only those vertices that represent propositional variables). For a set of vertices $S \subseteq V$, we define the substitution ρ_i as follows for all $s \in S \cap \mathsf{dom}(\pi)$: $\rho_i(\pi(s)) = true$ if $i = 0$ and $\rho_i(\pi(s)) = false$ if $i = 1$. The union of two substitutions is again a substitution, provided that the domain of variables these substitutions range over are disjoint.

The lemma below indicates how one can utilise such information to simplify the BES equation that is being created, again without affecting the solution to the BES.

Lemma 2 *The solution to all variables in a BES $\mathcal{F} \equiv \mathcal{E}(\sigma X = f)\mathcal{E}'$ is equivalent to the solution to those variables in the BES $\mathcal{F}' \equiv \mathcal{E}(\sigma X = f(\rho_0(S_0) \cup \rho_1(S_1)))\mathcal{E}'$, where for all $S_0 \cup S_1 \subseteq \{v \in V \mid \forall \theta, \theta' : [\mathcal{F}]\theta(\pi(v) = [\mathcal{F}]\theta(\pi(v)\}$, where $[\mathcal{F}]\theta$ denotes the solution to $\mathcal{F}$ under environment θ.*

Using this lemma, rather than creating a structure graph underlying the equation $\sigma X_e = \psi^e$, we can create a structure graph for $\sigma X_e = \psi^e(\rho_0(S_0) \cup \rho_1(S_1))$, provided that S_0 and S_1 contain vertices that represent solved equations. This leads to the following adaptations in the code: we maintain two sets S_0 and S_1 for which we can (cheaply) establish that these correspond to solved equations, and we add an assignment below the assignment of optimisation 1.

PbesinstStructureGraph$_2(\mathcal{E}, X_{init}(e_{init}), R)$
$init := R(X_{init}(e_{init}))$
$todo := \{init\}$
$discovered := \{init\}$
$(V, E) := (\emptyset, \emptyset)$
$S_0 := \emptyset$
$S_1 := \emptyset$
while $todo \neq \emptyset$ **do**
  choose $X_k(e) \in todo$
  $todo := todo \setminus \{X_k(e)\}$
  $\psi^e := R(\varphi_k[d_k := e])$
  $\psi^e :=$ **if** $\sigma_k = \mu$ **then** $R(\psi^e[X^e := false])$ **else** $R(\psi^e[X^e := true])$
  $\psi^e := R(\psi^e(\rho_0(S_0) \cup \rho_1(S_1)))$
  $S_0 := S_0 \cup \{X^e \mid \psi^e \equiv true\}$
  $S_1 := S_1 \cup \{X^e \mid \psi^e \equiv false\}$
  $(V, E) := (V, E) \cup SG^0(X^e, \psi^e)$
  $todo := todo \cup (\mathrm{occ}(\psi^e) \setminus discovered)$
  $discovered := discovered \cup \mathrm{occ}(\psi^e)$
return G,

Note on computing winning strategies. Rewriting the propositional formula ψ^e using S_0 and S_1 may result in a loss of information, preventing us from constructing a winning strategy for both players. We can solve that by mimicking the attractor set computation in our rewriting; that is, we implement $R(\psi^e(\rho_0(S_0) \cup \rho_1(S_1)))$ using a rewriter R^+ which takes a formula (and implicitly takes sets S_0 and S_1 into account. This leads to the following bottom-up procedure in which $R^+(f)$ yields a tuple (b, f'), where b is either a Boolean value, or the value $\bot$, and f' is a propositional formula that is equivalent to f under the assumption that S_0 and S_1 are solved.

- Case $f \equiv true$ then $R^+(f) = (f, f)$;
- Case $f \equiv false$ then $R^+(f) = (f, f)$;
- Case $f \equiv X$; then $R^+(f) = (true, X)$ when $X \in S_0$, $(false, X)$ when $X \in S_1$ and $(\bot, X)$ otherwise;
- Case $f \equiv f_1 \wedge f_2$, assuming that $R^+(f_i) = (b_i, f_i')$. If $b_1 = b_2 = true$, then $R^+(f) = (true, f_1' \wedge f_2')$. If $b_1 = false$ and $b_2 \neq false$, then $R^+(f) = (false, f_1')$. If $b_2 = false$ and $b_1 \neq false$ then $R^+(f) = (false, f_2')$. If $b_1 = b_2 = false$ then $R^+(f) = (false, f_1')$ when $|f_1'| < |f_2'|$ and $(false, f_2')$ otherwise. If $b_1 = b_2 = \bot$ then $R^+(f) = (\bot, f_1' \wedge f_2')$;

- Case $f \equiv f_1 \vee f_2$, assuming that $R^+(f_i) = (b_i, f_i')$. If $b_1 = b_2 = false$, then $R^+(f) = (false, f_1' \vee f_2')$. If $b_1 = true$ and $b_2 \neq true$, then $R^+(f) = (true, f_1')$. If $b_2 = true$ and $b_1 \neq true$ then $R^+(f) = (true, f_2')$. If $b_1 = b_2 = true$ then $R^+(f) = (true, f_1')$ when $|f_1'| < |f_2'|$ and $(true, f_2')$ otherwise. If $b_1 = b_2 = \bot$ then $R^+(f) = (\bot, f_1' \vee f_2')$.

The assignment $\psi^e := R(\psi^e(\rho_0(S_0) \cup \rho_1(S_1)))$ can be replaced by an assignment $(b, \psi^e) := R^+(\psi^e)$, and the extension to S_0 and S_1 can then be replaced by the following code:

$$\textbf{if } b = true \textbf{ then } S_0 := S_0 \cup \{X^e\}$$
$$\textbf{if } b = false \textbf{ then } S_1 := S_1 \cup \{X^e\}$$

Note that R^+ can probably also be implemented at the level of PBES expressions.

This results in the following adapted version:

PbesinstStructureGraph$_{2.1}(\mathcal{E}, X_{init}(e_{init}), R)$
$init := R(X_{init}(e_{init}))$
$todo := \{init\}$
$discovered := \{init\}$
$(V, E) := (\emptyset, \emptyset)$
$S_0 := \emptyset$
$S_1 := \emptyset$
while $todo \neq \emptyset$ **do**
  choose $X_k(e) \in todo$
  $todo := todo \setminus \{X_k(e)\}$
  $\psi^e := R(\varphi_k[d_k := e])$
  $\psi^e :=$ **if** $\sigma_k = \mu$ **then** $R(\psi^e[X^e := false])$ **else** $R(\psi^e[X^e := true])$
  $(b, \psi^e) := R^+(\psi^e)$
  if $b = true$ **then** $S_0 := S_0 \cup \{X^e\}$
  if $b = false$ **then** $S_1 := S_1 \cup \{X^e\}$
  $(V, E) := (V, E) \cup SG^0(X^e, \psi^e)$
  $todo := todo \cup (\mathrm{occ}(\psi^e) \setminus discovered)$
  $discovered := discovered \cup \mathrm{occ}(\psi^e)$
return (V, E)

5.3 Optimisation 3

When the computation of $SG^0(\varphi, \psi, r)$ finishes and the subgraph represented by SG^0 is effectively solved (which is the case if it represents $true$ or $false$), we can use these results to solve other variables by propagating the information in S^0 and S^1 to the structure graph constructed so far. The modification is minor, re-using the attractor set computation (and setting of a winning strategy) that is also part of Zielonka's recursive algorithm. More specifically, we can ensure that S_0 and S_1 are closed under the appropriate attractor set computations. Furthermore, if the initial vertex belongs to either S_0 or S_1, we can terminate the search. This leads to the following modified algorithm:

PbesinstStructureGraph$_3(\mathcal{E}, X_{init}(e_{init}), R)$
$init := R(X_{init}(e_{init}))$
$todo := \{init\}$
$discovered := \{init\}$
$(V, E) := (\emptyset, \emptyset)$
$S_0 := \emptyset$
$S_1 := \emptyset$
while $todo \neq \emptyset \wedge X_{init}(e_{init}) \notin S_0 \cup S_1$ **do**
  choose $X_k(e) \in todo$
  $todo := todo \setminus \{X_k(e)\}$
  $\psi^e := R(\varphi_k[d_k := e])$
  $\psi^e :=$ **if** $\sigma_k = \mu$ **then** $R(\psi^e[X^e := false])$ **else** $R(\psi^e[X^e := true])$
  $(b, \psi^e) := R^+(\psi^e)$
  if $b = true$ **then** $S_0 :=$ Attr$(S_0 \cup \{X^e\}, 0)$
  if $b = false$ **then** $S_1 :=$ Attr$(S_1 \cup \{X^e\}, 1)$
  $(V, E) := (V, E) \cup SG^0(X^e, \psi^e)$
  $todo := todo \cup (\text{occ}(\psi^e) \setminus discovered)$
  $discovered := discovered \cup \text{occ}(\psi^e)$
if $todo \neq \emptyset \wedge X_{init}(e_{init}) \in S_0$ **then**
  $V := V \setminus$ Attr$(todo, 1)$
else if $todo \neq \emptyset \wedge X_{init}(e_{init}) \in S_1$ **then**
  $V := V \setminus$ Attr$(todo, 0)$
return (V, E)

Further needless instantiation can be avoided by utilising the attractor strategies that are set when extending S_0 and S_1 by, once in a while, computing which vertices are still reachable from $X_{init}(e_{init})$, ignoring edges emanating from a vertex that are not part of the strategy for that vertex (if the strategy is set). This reachability analysis may be combined with a cheap algorithm that detects whether the game can in fact already be solved (see e.g. optimisation 4).

Note on computing winning strategies. The winning strategy can be set by extending it using the attractor strategy that is computed while computing $Attr_0(S_0 \cup \{X^e\})$ and $Attr_1(S_1 \cup \{X^e\})$.

5.4 Optimisation 4

Several simple algorithms exist that can solve partial structure graphs. An example algorithm is, for instance, the algorithm that computes whether is an *odd-rank* dominated conjunctive loop or an *even-rank* dominated disjunctive loop. The `pbes2bool` optimisation, implemented at the level of the structure graph is as follows (it typically assumes that the set U contains all fully explored vertices, r is the rank function, $\mathsf{dom}(r)$ yields the set of vertices with a rank associated to them; the function $\mathsf{parity}(p) = \blacktriangle$ when p is odd, and $\blacktriangledown$ otherwise).

```
FindLoop(U, v, w, p, visited)
if d(w) ∈ {⊤, ⊥} then return false
if w ∈ dom(r) and r(w) ≠ p then return false
if w ∈ keys(visited) then return visited(w)
if w ∈ U then
    visited(w) := false
    if (w ∈ dom(d) ⇒ d(w) = parity(p)) then
        b := (w → v) ∨ ⋁_{u∈{u'|w→u'∧u'≠v}} FindLoop(U, v, u, p, visited)
    else
        return false
    visited(w) := b
    return b
return false
```

The routine FindLoop$(U, v, w, r(v), \text{visited})$ finds a $r(v)$-ranked (possibly tree-like) loop starting in v, within a set of vertices U. Note that in the above routine, parameter w represents the 'current' vertex, whereas v represents the vertex from which the search was initiated. Note that parameter p fulfils the role of the fixpoint sign σ in the original algorithm, parameter v fulfils the role of $X_k(e)$ and w fulfils the role of ϕ.

The procedure FindLoops can be called from within the main algorithm; it searches for loops in the structure graph currently constructed.

```
FindLoops(discovered, todo, S_0, S_1)
done := discovered \ todo
visited := []
b_0, b_1 := false, false
for u ∈ done ∩ dom(r) do
    if u ∉ keys(visited) then visited(u) := false
    b := FindLoop(done, u, u, r(u), visited)
    visited(u) := b
    if b then
        if r(u) mod 2 = 0 then S_0, b_0 := S_0 ∪ {u}, true
        else S_1, b_1 := S_1 ∪ {u}, true
if b_0 then S_0 := Attr(S_0, 0)
if b_1 then S_1 := Attr(S_1, 1)
return S_0, S_1
```

The above optimisation can be integrated in the algorithm as follows:

```
PbesinstStructureGraph4(ℰ, X_init(e_init), R)
init := R(X_init(e_init))
todo := {init}
discovered := {init}
(V, E) := (∅, ∅)
S_0 := ∅
S_1 := ∅
while todo ≠ ∅ ∧ X_init(e_init) ∉ S_0 ∪ S_1 do
    choose X_k(e) ∈ todo
    todo := todo \ {X_k(e)}
    ψ^e := R(φ_k[d_k := e])
    ψ^e := if σ_k = μ then R(ψ^e[X^e := false]) else R(ψ^e[X^e := true])
    (b, ψ^e) := R^+(ψ^e)
    if b = true then S_0 := Attr(S_0 ∪ {X^e}, 0)
    if b = false then S_1 := Attr(S_1 ∪ {X^e}, 1)
    (V, E) := (V, E) ∪ SG^0(X^e, ψ^e)
    todo := todo ∪ (occ(ψ^e) \ discovered)
    discovered := discovered ∪ occ(ψ^e)
    S_0, S_1 := FindLoops(discovered, todo, S_0, S_1)
if todo ≠ ∅ ∧ X_init(e_init) ∈ S_0 then
    V := V \ Attr(todo, 1)
else if todo ≠ ∅ ∧ X_init(e_init) ∈ S_1 then
    V := V \ Attr(todo, 0)
return (V, E)
```

Note that in the call to FindLoops we would like to pass the argument $done = discovered \setminus todo$. However this set is not available, and it is too expensive to compute. So we pass the larger set *discovered*, and we use $discovered \cap dom(r) = discovered \setminus todo$.

5.5 Optimisation 5

A generalisation of optimisation 4 utilises a so-called *fatal attractor*. Fatal attractors are based on the observation that those vertices with a dominant priority that are attracted into themselves are won by the player with the parity of this dominant priority. A simple algorithm exploiting this is the following. It uses a modified attractor computation.

```
FatalAttractors(V, S_0, S_1)
J := {j | ∃u ∈ V ∩ dom(r) : r(u) = j}
for j ∈ J
    α := j mod 2
    U_j := {u ∈ V | r(u) = j ∧ (α = 0 ⇒ d(u) ≠ false) ∧ (α = 1 ⇒ d(u) ≠ true)} \ S_{1−α}
    U := U_j ∪ S_α
    X := AttrMinRank(U, α, V, j)
    Y := V \ Attr(V \ X, 1 − α)
    while X ≠ Y do
        X := AttrMinRank(U ∩ Y, α, V, j)
        Y := Y \ Attr(Y \ X, 1 − α)
    S_α := S_α ∪ X
return Attr(S_0, 0), Attr(S_1, 1)
```

where AttrMinRank is a slightly modified version of the original attractor set computation Attr, where only predecessors in U with a rank of at least j are considered:

$\textsc{AttrMinRank}(A, \alpha, U, j)$
$todo := \bigcup_{u \in A}(\ pred^{\geq j}(u, U)\ \setminus A)$
while $todo \neq \emptyset$ **do**
  choose $u \in todo$
  $todo := todo \setminus \{u\}$
  if $d(u) = \alpha \vee succ(u) \subseteq A$
    $A := A \cup \{u\}$
    $todo := todo \cup (\ pred^{\geq j}(u, U)\ \setminus A)$
return A

where

$$pred^{\geq j}(u, U) = \{v \in U \mid (v, u) \in E \wedge (r(v) \geq j \vee (v \notin dom(r) \wedge d(v) \in \{\blacktriangle, \blacktriangledown\})\}$$

The above optimisation can be integrated in the algorithm as follows:

$\textsc{PbesinstStructureGraph}_5(\mathcal{E}, X_{init}(e_{init}), R)$
$init := R(X_{init}(e_{init}))$
$todo := \{init\}$
$discovered := \{init\}$
$(V, E) := (\emptyset, \emptyset)$
$S_0 := \emptyset$
$S_1 := \emptyset$
while $todo \neq \emptyset \wedge X_{init}(e_{init}) \notin S_0 \cup S_1$ **do**
  choose $X_k(e) \in todo$
  $todo := todo \setminus \{X_k(e)\}$
  $\psi^e := R(\varphi_k[d_k := e])$
  $\psi^e :=$ **if** $\sigma_k = \mu$ **then** $R(\psi^e[X^e := false])$ **else** $R(\psi^e[X^e := true])$
  $(b, \psi^e) := R^+(\psi^e)$
  if $b = true$ **then** $S_0 := \textsc{Attr}(S_0 \cup \{X^e\}, 0)$
  if $b = false$ **then** $S_1 := \textsc{Attr}(S_1 \cup \{X^e\}, 1)$
  $(V, E) := (V, E) \cup SG^0(X^e, \psi^e)$
  $todo := todo \cup (\mathrm{occ}(\psi^e) \setminus discovered)$
  $discovered := discovered\ \cup \mathrm{occ}(\psi^e)$
  $S_0, S_1 := \textsc{FatalAttractors}(V, S_0, S_1)$
if $todo \neq \emptyset \wedge X_{init}(e_{init}) \in S_0$ **then**
  $V := V \setminus \textsc{Attr}(todo, 1)$
else if $todo \neq \emptyset \wedge X_{init}(e_{init}) \in S_1$ **then**
  $V := V \setminus \textsc{Attr}(todo, 0)$
return (V, E)

5.6 Optimisation 6

Optimisation 6 is a slightly differen fatal attractor computation that is very close to the original one by Michael Huth, Jim Huan-Pu Kuo, and Nir Piterman.

```
        AttrMinRankOriginal(A, α, U, j)
        {compute the α-min attractor into the set A, restricted to vertices in U}
        todo := ⋃_{u∈A}(pred^{≥j}(u, U))
        X := {u ∈ todo ∩ A | d(u) = α ∨ succ(u) ⊆ A}
        while todo ≠ ∅ do
            choose u ∈ todo
            todo := todo \ {u}
            if d(u) = α ∨ succ(u) ⊆ A ∪ X
                X := X ∪ {u}
                todo := todo ∪ (pred^{≥j}(u, U) \ X)
        return X

    FatalAttractorsOriginal(V, S_0, S_1)
    J := {j | ∃u ∈ V : r(u) = j}
    for j ∈ J
        α := j  mod 2
        U_j := {u ∈ V | r(u) = j ∧ (α = 0 ⇒ d(u) ≠ false) ∧ (α = 1 ⇒ d(u) ≠ true)} \ S_{1−α}
        X := ∅
        while U_j ≠ ∅ ∧ U_j ≠ X
            X := U_j
            Y := AttrMinRankOriginal(X ∪ S_α, α, V, j)
            if U_j ⊆ Y
                S_α := S_α ∪ Y
                break
            else
                U_j := U_j ∩ Y
    return Attr(S_0, 0), Attr(S_1, 1)
```

5.7 Optimisation 7

In optimisation 7 the sets S_0 and S_1 are extended by solving a partial game.

$\textsc{PbesinstStructureGraph}_7(\mathcal{E}, X_{init}(e_{init}), R)$
$init := R(X_{init}(e_{init}))$
$todo := \{init\}$
$discovered := \{init\}$
$(V, E) := (\emptyset, \emptyset)$
$S_0 := \emptyset$
$S_1 := \emptyset$
while $todo \neq \emptyset \wedge X_{init}(e_{init}) \notin S_0 \cup S_1$ **do**
  choose $X_k(e) \in todo$
  $todo := todo \setminus \{X_k(e)\}$
  $\psi^e := R(\varphi_k[d_k := e])$
  $\psi^e :=$ **if** $\sigma_k = \mu$ **then** $R(\psi^e[X^e := false])$ **else** $R(\psi^e[X^e := true])$
  $(b, \psi^e) := R^+(\psi^e)$
  if $b = true$ **then** $S_0 := \textsc{Attr}(S_0 \cup \{X^e\}, 0)$
  if $b = false$ **then** $S_1 := \textsc{Attr}(S_1 \cup \{X^e\}, 1)$
  $(V, E) := (V, E) \cup SG^0(X^e, \psi^e)$
  $todo := todo \cup (\mathrm{occ}(\psi^e) \setminus discovered)$
  $discovered := discovered \cup \mathrm{occ}(\psi^e)$
  $S_0, S_1 := \textsc{PartialSolve}(V, todo, S_0, S_1)$
if $todo \neq \emptyset \wedge X_{init}(e_{init}) \in S_0$ **then**
  $V := V \setminus \textsc{Attr}(todo, 1)$
else if $todo \neq \emptyset \wedge X_{init}(e_{init}) \in S_1$ **then**
  $V := V \setminus \textsc{Attr}(todo, 0)$
return (V, E)

where PartialSolve is defined as

$\textsc{PartialSolve}(V, todo, S_0, S_1)$
$\{$*use the solver to compute an over and underapproaximation*$\}$
$S_0, S_1 := \textsc{Attr}(S_0, 0), \textsc{Attr}(S_1, 1)$
$(W_0, W_1) := \textsc{SolveRecursive}(V \setminus (S_1 \cup \textsc{Attr}(S_0 \cup todo, 0)))$
$S_1 := \textsc{Attr}(S_1 \cup W_1, 1)$
$(W_0, W_1) := \textsc{SolveRecursive}(V \setminus (S_0 \cup \textsc{Attr}(S_1 \cup todo, 1)))$
$S_0 := \textsc{Attr}(S_0 \cup W_0, 0)$