PBES rewriters

Wieger Wesselink, Tim Willemse, Thomas Neele

April 26, 2024

This document describes several rewriters on PBES expressions. We assume that a data rewriter r is given that rewrites data expressions.

1 Simplifying rewriter

We define a simplifying PBES rewriter R recursively as follows. We assume that D is a non-empty data type, and we denote the free variables appearing in φ as $\mathsf{free}(\varphi)$. We assume that a data rewriter r is given that rewrites data terms.

$$
\begin{array}{lcl}
b & \to & r(b) \\
\neg\neg\varphi & \to & \varphi \\
\varphi \wedge true & \to & \varphi \\
true \wedge \varphi & \to & \varphi \\
\varphi \wedge false & \to & false \\
false \wedge \varphi & \to & false \\
\varphi \wedge \varphi & \to & \varphi \\
\varphi \vee true & \to & true \\
true \vee \varphi & \to & true \\
\varphi \vee false & \to & \varphi \\
false \vee \varphi & \to & \varphi \\
\varphi \vee \varphi & \to & \varphi \\
\varphi \Rightarrow \psi & \to & \neg\varphi \vee \psi \\
\forall_{d:D}.\varphi & \to & \varphi \text{ if } d \notin \mathsf{free}(\varphi) \\
\forall_{d:D}.\neg\varphi & \to & \neg\exists_{d:D}.\varphi \\
\forall_{d:D}.\varphi \wedge \psi & \to & \forall_{d:D}.\varphi \wedge \forall_{d:D}.\psi \\
\forall_{d:D}.\varphi \vee \psi & \to & (\forall_{d:D}.\varphi) \vee \psi \text{ if } d \notin \mathsf{free}(\psi) \\
\forall_{d:D}.\varphi \vee \psi & \to & \varphi \vee (\forall_{d:D}.\psi) \text{ if } d \notin \mathsf{free}(\varphi) \\
\exists_{d:D}.\varphi & \to & \varphi \text{ if } d \notin \mathsf{free}(\varphi) \\
\exists_{d:D}.\neg\varphi & \to & \neg\forall_{d:D}.\varphi \\
\exists_{d:D}.\varphi \vee \psi & \to & \exists_{d:D}.\varphi \vee \exists_{d:D}.\psi \\
\exists_{d:D}.\varphi \wedge \psi & \to & (\exists_{d:D}.\varphi) \wedge \psi \text{ if } d \notin \mathsf{free}(\psi) \\
\exists_{d:D}.\varphi \wedge \psi & \to & \varphi \wedge (\exists_{d:D}.\psi) \text{ if } d \notin \mathsf{free}(\varphi) \\
X(e) & \to & X(r(e))
\end{array}
$$

where φ and ψ are arbitrary pbes expressions, b is a data term of data sort $\mathbb{B}$, $true$ and $false$ are elements of data sort $\mathbb{B}$, X is a predicate variable, e consists of zero or more data sorts and d is a data variable of sort D.

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

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

[1] Todo: reformulate this rewrite system.

2 PFNF Rewriter

Definition 1 *A predicate formula is said to be in* Predicate Formula Normal Form *(PFNF) if it has the following form:*

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

where $X^j \in \chi$ (χ is a countable of sorted predicate variables), $\mathsf{Q}_i \in \{\forall,\exists\}$, 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:

$$\begin{array}{lll}
p(c) & =_{def} & c\\
p(X(d)) & =_{def} & X(d)\\
p(\forall x{:}D.\varphi) & =_{def} & \forall x{:}D.p(\varphi)\\
p(\exists x{:}D.\varphi) & =_{def} & \exists x{:}D.p(\varphi)
\end{array}$$

$$\begin{array}{lll}
p(\varphi\wedge\psi) & =_{def} & \mathsf{Q}^{\varphi}_1\cdots\mathsf{Q}^{\varphi}_{n^\varphi}\mathsf{Q}^{\psi}_1\cdots\mathsf{Q}^{\psi}_{n^\psi}.\ \left(h^\varphi\wedge h^\psi\right)\\
 & & \wedge\bigwedge_{i\in I^\varphi\cup I^\psi}\left(g_i \implies \bigvee_{j\in J_i} X^j(e^j)\right)
\end{array}$$

$$\begin{array}{lll}
p(\varphi\vee\psi) & =_{def} & \mathsf{Q}^{\varphi}_1\cdots\mathsf{Q}^{\varphi}_{n^\varphi}\mathsf{Q}^{\psi}_1\cdots\mathsf{Q}^{\psi}_{n^\psi}.\left(h^\varphi\vee h^\psi\right)\\
 & & \wedge\bigwedge_{i\in I^\varphi}\left(\left(\neg h^\psi\wedge g_i\right) \implies \bigvee_{j\in J_i} X^j(e^j)\right)\\
 & & \wedge\bigwedge_{i\in I^\psi}\left(\left(\neg h^\varphi\wedge g_i\right) \implies \bigvee_{j\in J_i} X^j(e^j)\right)\\
 & & \wedge\bigwedge_{i\in I^\varphi,k\in I^\psi}\left((g_i\wedge g_k) \implies \bigvee_{j\in J_i,m\in J_k} X^j(e^j)\vee X^m(e^m)\right)
\end{array}$$

where

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

under the assumption that I^φ and I^ψ are disjoint, and $v^\varphi_i \neq v^\psi_j$ for all i,j.

3 OnePointRule Quantifier Elimination Rewriter

The function Eq computes a set of equalities and inequalities for an expression φ, such that the following holds:

$$Eq(\varphi) = (\{b_1 = e_1, \cdots, b_n = e_n\}, W) \quad \Rightarrow \quad \varphi \equiv \psi \wedge \bigwedge_{i=1}^{n} (b_i = e_i)$$

$$Eq(\varphi) = (V, \{b_1 \neq e_1, \cdots, b_n \neq e_n\}) \quad \Rightarrow \quad \varphi \equiv \psi \vee \bigvee_{i=1}^{n} (b_i \neq e_i)$$

for some expression ψ.

The function Eq is inductively defined as follows:

$$\begin{array}{lcl}
Eq(true) & = & (\{\emptyset, \top\}) \\
Eq(false) & = & (\{\top, \emptyset\}) \\
Eq(b) & = & (\{b = true\}, \{b \neq false\}) \\
Eq(d = e) & = & \begin{cases} (\{d = e\}, \emptyset) & \text{if } d \notin FV(e) \\ (\emptyset, \emptyset) & \text{otherwise} \end{cases} \\
Eq(e = d) & = & Eq(d = e) \\
Eq(d \neq e) & = & \begin{cases} (\emptyset, \{d \neq e\}) & \text{if } d \notin FV(e) \\ (\emptyset, \emptyset) & \text{otherwise} \end{cases} \\
Eq(e \neq d) & = & Eq(d \neq e) \\
Eq(\neg\varphi) & = & swap(Eq(\varphi)) \\
Eq(\varphi \wedge \psi) & = & join_and((Eq(\varphi), Eq(\psi))) \\
Eq(\varphi \vee \psi) & = & join_or((Eq(\varphi), Eq(\psi))) \\
Eq(\varphi \Rightarrow \psi) & = & join_or((swap(Eq(\varphi)), Eq(\psi))) \\
Eq(if(\varphi, \psi, \chi)) & = & ((V_\varphi \cup V_\psi) \cap (W_\varphi \cup V_\chi), (V_\varphi \cup W_\psi) \cap (W_\varphi \cup W_\chi)), \\
 & & \quad \text{where} \quad (V_\varphi, W_\varphi) := Eq(\varphi) \\
 & & \qquad\quad (V_\psi, W_\psi) := Eq(\psi) \\
 & & \qquad\quad (V_\chi, W_\chi) := Eq(\chi) \\
Eq(\forall x.\varphi) & = & delete(x, Eq(\varphi)) \\
Eq(\exists x.\varphi) & = & delete(x, Eq(\varphi)) \\
Eq(\varphi) & = & (\emptyset, \emptyset) \text{ otherwise}
\end{array}$$

where b is a boolean data variable, d is an arbitrary data variable, e is a boolean data expression, $\top$ is the set of all equalities and inequalities (so it is the idempotent element for $\cap$) and

$$\begin{array}{lcl}
swap((V, W)) & = & (W, V) \\
join_and((V_1, W_1), (V_2, W_2)) & = & (V_1 \cup V_2, W_1 \cap W_2) \\
join_or((V_1, W_1), (V_2, W_2)) & = & (V_1 \cap V_2, W_1 \cup W_2) \\
delete(x, (V, W)) & = & (V_1, W_1) \text{ where} \\
 & & \begin{cases} V_1 = \{d = e \in V \mid d \neq x \wedge x \notin FV(e)\} \\ W_1 = \{d \neq e \in W \mid d \neq x \wedge x \notin FV(e)\} \end{cases}
\end{array}$$

We define the OnePointRule rewriter R inductively as follows:

$$
\begin{array}{lcl}
R(\neg\varphi) & = & \neg R(\varphi) \\
R(\varphi \wedge \psi) & = & R(\varphi) \wedge R(\psi) \\
R(\varphi \vee \psi) & = & R(\varphi) \vee R(\psi) \\
R(\varphi \Rightarrow \psi) & = & R(\varphi) \Rightarrow R(\psi) \\
R(\forall x.\varphi) & = & \begin{cases} R(\varphi)[x := e] & \text{if } x \neq e \in W, \text{ where } (V, W) = Eq(\varphi) \\ \forall x.R(\varphi) & \text{otherwise} \end{cases} \\
R(\exists x.\varphi) & = & \begin{cases} R(\varphi)[x := e] & \text{if } x = e \in V, \text{ where } (V, W) = Eq(\varphi) \\ \exists x.R(\varphi) & \text{otherwise} \end{cases} \\
R(\varphi) & = & \varphi \text{ otherwise}
\end{array}
$$

4 Quantifier Inside Rewriter

This rewriter was originally specified by Jan Friso Groote. We define a rewriter that pushes universal and existential quantifiers into an expression. A typical example is

$$\exists x.\forall y.(f(y,y) \wedge (f(x,y) \vee f(x,x)))$$

which is rewritten to

$$(\forall y.f(y,y)) \wedge ((\exists x.\forall y.f(x,y)) \vee \exists x.f(x,x)).$$

If quantifiers are pushed inside formulas as much as possible, quantifier elimination leads to smaller expressions and the one-point rewriter is more often applicable. There might be cases where the application of this push-quantifiers-inside-rewriter can also have averse effects.

Below the definition of the rewrite rules are given. It is assumed that it is cheap to obtain the free variable in each term, which are denoted by $vars(\phi)$ for a term ϕ. If not, care needs to be taken as in the formulation below calculating the variables of each term recursively on a by need basis can be very expensive.

Let ϕ be a pbes expression. We define $R(\phi)$ using the functions $R_\forall(V,\phi)$ and $R_\exists(V,\phi)$ where V is a set of typed variables. The definition of R employs the structure of a formula:

$$\begin{array}{lcl}
R(\neg\phi) & = & \neg R(\phi) \\
R(\phi \wedge \psi) & = & R(\phi) \wedge R(\psi) \\
R(\phi \vee \psi) & = & R(\phi) \vee R(\psi) \\
R(\phi \Rightarrow \psi) & = & R(\phi) \Rightarrow R(\psi) \\
R(\forall W.\phi) & = & R_\forall(W, R(\phi)) \\
R(\exists W.\phi) & = & R_\exists(W, R(\phi)) \\
R(\phi) & = & \phi \text{ otherwise.}
\end{array}$$

Here W is a set of typed variables.

The function $R_\forall$ is defined as

$$\begin{array}{lcl}
R_\forall(V,\neg\phi) & = & \neg R_\exists(V,\phi) \\
R_\forall(V,\phi \wedge \psi) & = & R_\forall(V,\phi) \wedge R_\forall(V,\psi) \\
R_\forall(V,\bigvee_i \phi_i) & = & \begin{cases}
\forall V \cap vars(\bigvee_i \phi_i).\bigvee_i \phi_i & \text{if } \Psi = \mathit{false} \\
\forall V \cap vars(\Phi) \cap vars(\Psi). & \text{otherwise} \\
\quad \big(R_\forall(V \cap vars(\Phi) \setminus vars(\Psi), \Phi) \vee R_\forall(V \cap vars(\Psi) \setminus vars(\Phi), \Psi)\big) &
\end{cases} \\
R_\forall(V,\phi \Rightarrow \psi) & = & R_\forall(V,\neg\phi \vee \psi) \\
R_\forall(V,\forall W.\phi) & = & R_\forall(V \cup W,\phi) \\
R_\forall(V,\phi) & = & \forall V \cap vars(\phi).\phi \text{ otherwise.}
\end{array}$$

where

$$\begin{cases}
Z = vars(\phi_j) \cap V \text{ for some } j \text{ such that } |Z| \text{ is minimal} \\
\Phi = \bigvee \{\phi_i \mid (vars(\phi_i) \cap V) \subseteq Z\} \\
\Psi = \bigvee \{\phi_i \mid (vars(\phi_i) \cap V) \not\subseteq Z\}
\end{cases}$$

The function $R_\exists$ is defined as

$$\begin{array}{lcl}
R_\exists(V,\neg\phi) & = & \neg R_\forall(V,\phi) \\
R_\exists(V,\bigwedge_i \phi_i) & = & \begin{cases}
\exists V \cap vars(\bigwedge_i \phi_i).\bigwedge_i \phi_i & \text{if } \Psi = \mathit{true} \\
\exists V \cap vars(\Phi) \cap vars(\Psi). & \text{otherwise} \\
\quad \big(R_\exists(V \cap vars(\Phi) \setminus vars(\Psi), \Phi) \wedge R_\exists(V \cap vars(\Psi) \setminus vars(\Phi), \Psi)\big) &
\end{cases} \\
R_\exists(V,\phi \vee \psi) & = & R_\exists(V,\phi) \vee R_\exists(V,\psi) \\
R_\exists(V,\phi \Rightarrow \psi) & = & R_\exists(V,\neg\phi \vee \psi) \\
R_\exists(V,\exists W.\phi) & = & R_\exists(V \cup W,\phi) \\
R_\exists(V,\phi) & = & \exists V \cap vars(\phi).\phi \text{ otherwise.}
\end{array}$$

where

$$\begin{cases} Z = vars(\phi_j) \cap V \text{ for some } j \text{ such that } |Z| \text{ is minimal,} \\ \Phi = \bigwedge \{\phi_i \mid (vars(\phi_i) \cap V) \subseteq Z\} \\ \Psi = \bigwedge \{\phi_i \mid (vars(\phi_i) \cap V) \nsubseteq Z\} \end{cases}$$

5 Quantifier Expansion Rewriter

5.1 Conjunctions / disjunctions

The function CD computes a sequence of conjunctions and disjunctions for an expression φ, such that the following holds:

$$\begin{aligned} CD\left(\varphi\right) &= \left(\left[\varphi_1, \cdots, \varphi_n\right], W\right) \Rightarrow \varphi = \bigwedge_{i=1}^{n} \varphi_i \\ CD\left(\varphi\right) &= \left(V, \left[\varphi_1, \cdots, \varphi_n\right]\right) \Rightarrow \varphi = \bigvee_{i=1}^{n} \varphi_i. \end{aligned}$$

The function S is inductively defined as follows:

$$\begin{array}{lcl} CD\left(\neg\varphi\right) & = & negate\left(CD\left(\varphi\right)\right) \\ CD\left(\varphi \wedge \psi\right) & = & \left(conjunctions\left(CD\left(\varphi\right)\right) +\!\!+ conjunctions\left(CD\left(\psi\right)\right), \emptyset\right) \\ CD\left(\varphi \vee \psi\right) & = & \left(\emptyset, disjunctions\left(CD\left(\varphi\right)\right) +\!\!+ disjunctions\left(CD\left(\psi\right)\right)\right) \\ CD\left(\varphi \Rightarrow \psi\right) & = & \left(\emptyset, disjunctions\left(negate(CD\left(\varphi\right))\right) +\!\!+ disjunctions\left(CD\left(\psi\right)\right)\right) \\ CD\left(\varphi\right) & = & \left(\varphi, \varphi\right) \text{ otherwise} \end{array}$$

with

$$\begin{array}{lcl} negate\left(\left(\left[\varphi_1, \cdots, \varphi_m\right], \left[\psi_1, \cdots, \psi_n\right]\right)\right) & = & \left(\left[\neg\psi_1, \cdots, \neg\psi_n\right], \left[\neg\varphi_1, \cdots, \neg\varphi_m\right]\right) \\ conjunctions\left(\left(V, W\right)\right) & = & \begin{cases} V & \text{if } V \neq \emptyset \\ \bigvee\limits_{w \in W} w & \text{otherwise} \end{cases} \\ disjunctions\left(\left(V, W\right)\right) & = & \begin{cases} W & \text{if } W \neq \emptyset \\ \bigwedge\limits_{v \in V} v & \text{otherwise} \end{cases} \end{array}$$

N.B. This has not been implemented yet.