PBES Abstraction

Wieger Wesselink, Tim Willemse

January 19, 2019

1 Absinthe

The Absinthe algorithm takes as input a PBES p, a substitution on function symbols $\sigma_F : \mathcal{F} \to \mathcal{F}$, and a substitution on sorts $\sigma_S : \mathcal{S} \to \mathcal{S}$, where $\mathcal{F}$ is the set of function symbols and $\mathcal{S}$ is the set of sorts. For each $s \in \mathit{domain}(\sigma_S)$ there is a corresponding *abstraction function* h_s.

1.1 Definitions

We define $\mathit{FunctionSymbols}(p)$ as the set of function symbols that occur in the PBES p. For a sort s we define $\mathit{ContainerConstructors}(s)$ as the set of constructor functions of the sorts $\mathit{List}(s)$, $\mathit{Set}(s)$ and $\mathit{Bag}(s)$. This includes for example the functions $[] : \mathit{List}(s)$ and $\triangleright : s \times \mathit{List}(s) \to \mathit{List}(s)$.

1.2 Algorithm

The algorithm first extends and modifies the substitution σ_F. Then a transformation to p is applied.

1.2.1 Step 1

The algorithm first extends the substitution σ_F. Let

$$A = \left(\mathit{FunctionSymbols}(p) \cup \bigcup_{s \in \mathit{domain}(\sigma_S)} \mathit{ContainerConstructors}(s) \right) \setminus \mathit{domain}(\sigma_F).$$

For each function symbol $f_1 \in A$ the substitution σ_F is extended with $\sigma_F(f_1) := f_2$, where f_2 is obtained from f_1 using

$$\begin{array}{lcl} f_1 : s_b & \mapsto & f_2 : s_b^{\sigma_S} \\ f_1 : C(s) & \mapsto & f_2 : C(s)^{\sigma_S} \\ f_1 : s_1 \times \cdots \times s_n \to s & \mapsto & f_2 : s_1^{\sigma_S} \times \cdots s_n^{\sigma_S} \to \mathit{Set}(s^{\sigma_S}), \end{array}$$

where s_b is a basic sort, $s_1 \times \cdots \times s_n \to s$ is a function sort, and $C(s)$ is a container sort, and where f_2 is a fresh identifier.

Remark 1 *The sort of f_1 may not contain any element of $\mathit{domain}(\sigma_S)$ as a subsort.*

Remark 2 *If f_1 is a function update, then the domain of the updated function may not contain any element of $\mathit{domain}(\sigma_S)$ as a subsort.*

For each of the function symbols $f \in FunctionSymbols(p) \setminus domain(\sigma_F)$ a corresponding equation is introduced:

$$\begin{aligned} f_2 &= \begin{cases} h_{s_b}(f_1) & \text{if } s_b \in domain(\sigma_S) \\ f^{\sigma_S} & \text{otherwise} \end{cases} \\ f_2 &= f_1^{\sigma_S} \\ f_2(x) &= \begin{cases} \{h_s(f_1(x))\} & \text{if } TargetSort(s) \in domain(\sigma_S) \\ \{f_1(x)\} & \text{otherwise,} \end{cases} \end{aligned}$$

where $x : s_1^{\sigma_S} \times \cdots \times s_n^{\sigma_S} \to s^{\sigma_S}$, and where $TargetSort$ is recursively defined as

$$\begin{aligned} TargetSort(s_b) &= s_b \\ TargetSort(C(s)) &= C(s) \\ TargetSort(s_1 \times \cdots \times s_n \to s) &= TargetSort(s). \end{aligned}$$

1.2.2 Step 2

After this the substitution σ_F is transformed. Each $(f_1, f_2) \in \sigma_F$ is replaced by (f_1, f_3), where f_3 is obtained from f_2 as follows:

$$\begin{aligned} f_2 : s_b &\mapsto f_3 : Set(s_b) \\ f_2 : C(s) &\mapsto f_3 : Set(C(s)) \\ f_2 : s_1 \times \cdots \times s_n \to s &\mapsto f_3 : Set(s_1) \times \cdots \times Set(s_n) \to s, \end{aligned}$$

where $f3$ is a fresh name.

For each pair (f_2, f_3) a corresponding equation is generated:

$$\begin{aligned} f_3 &= \{f_2\} \\ f_3 &= \{f_2\} \\ f_3(X) &= \{y : s \mid \exists_{x:s_1 \times \cdots \times s_n \to s} \,.\, x \in X \wedge y \in f_2(x)\}, \end{aligned}$$

where $X : Set(s_1) \times \cdots \times Set(s_n) \to s$.

1.2.3 Step 3

The PBES p is transformed using the transformations T and U, that is defined recursively as:

$$
\begin{array}{rcl}
T((\sigma_1 X_1(d_1 & : & D_1) = \varphi_1) \ldots (\sigma_n X_n(d_n : D_n) = \varphi_n)) = (\sigma_1 \widehat{X_1}(d_1 : D_1^{\sigma_S}) = T(\varphi_1)) \ldots (\sigma_n \widehat{X_n}(d_n : D_n^{\sigma_S}) = T(\varphi_n)) \\
T(\neg\varphi) & = & \neg T(\varphi) \\
T(\varphi \oplus \psi) & = & T(\varphi) \oplus T(\psi) \\
T(X_i(e)) & = & \begin{cases} \bigvee\limits_{d \in T(e)} \widehat{X_i}(d) & \text{if it is an under-approximation} \\ \bigwedge\limits_{d \in T(e)} \widehat{X_i}(d) & \text{if it is an over-approximation} \end{cases} \\
T(\forall_{d:D}.\varphi) & = & \forall_{d:D^{\sigma_S}}.T(\varphi) \\
T(\exists_{d:D}.\varphi) & = & \exists_{d:D^{\sigma_S}}.T(\varphi) \\
T(b) & = & \begin{cases} false \in U(b) & \text{if it is an under-approximation} \\ true \in U(b) & \text{if it is an over-approximation,} \end{cases} \\
U(v) & = & \{v^{\sigma_S}\} \\
U(f) & = & \sigma_F(f) \\
U(c) & = & \{h_s(c)\} \text{ if } c : s \text{ is a ground term and } s \in domain(\sigma_S) \\
U(f(x)) & = & U(f)(U(x)) \text{ N.B. The general case } U(y(x)) \text{ is not supported!} \\
U(\lambda_{d:D}.x) & = & \{v : s^{\sigma_S} \mid v = U(x)\}, \text{ where } x : s \\
U(\forall_{d:D}.x) & = & ? \\
U(\exists_{d:D}.x) & = & ? \\
U(x \text{ whr } y & : & = z) = ?
\end{array}
$$

where $\oplus \in \{\wedge, \vee, \Rightarrow\}$. Note that T operates on PBES expressions, and U operates on data expressions.

2 PBES abstraction

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 V_i be a subset of the parameters in d_{X_i} for $i = 1 \cdots n$. Let $e \in \{true, false\}$ be a data expression. Then we define the algorithm $\mathsf{abstract}$ as follows:

$$
\begin{array}{lcl}
\mathsf{abstract}(d, V, e) & = & \begin{cases} d & \text{if } \mathsf{freevar}(d) \cap V = \emptyset \\ e & \text{otherwise} \end{cases} \\
\mathsf{abstract}(\varphi \oplus \psi, V, e) & = & \mathsf{abstract}(\varphi, V, e) \oplus \mathsf{abstract}(\varphi, V, e) \\
\mathsf{abstract}(\mathsf{Q}_{d_1:D_1,\cdots,d_m:D_m}.\varphi, V, e) & = & \mathsf{Q}_{d_1:D_1,\cdots,d_m:D_m}.\mathsf{abstract}(\varphi, V \setminus \{d_1 : D_1, \cdots, d_m : D_m\}, e) \\
\mathsf{abstract}(\sigma X(d_1 : D_1, \cdots, d_m : D_m) = \varphi, V, e) & = & \sigma X(d_1 : D_1, \cdots, d_m : D_m) = \mathsf{abstract}(\varphi, V, e) \\
\mathsf{abstract}(\mathcal{E}, [V_1, \cdots, V_n], e) & = & \begin{array}{l} (\sigma_1 X_1(d_{X_1} : D_{X_1}) = \mathsf{abstract}(\varphi_{X_1}, V_1, e)) \\ \cdots \\ (\sigma_n X_n(d_{X_n} : D_{X_n}) = \mathsf{abstract}(\varphi_{X_n}, V_n, e) \end{array}
\end{array}
$$

with d a data expression, $\oplus \in \{\wedge, \vee, \Rightarrow\}$, and $\mathsf{Q} \in \{\forall, \exists\}$ and $V \subset \{d_1 : D_1, \cdots, d_m : D_m\}$.

2.1 Motivation

The motivation for this algorithm is that if the solution of $\mathsf{abstract}(\mathcal{E}, [V_1, \cdots, V_n], false)$ is $true$, this implies that the solution of $\mathcal{E}$ is $true$ as well, and if the solution of $\mathsf{abstract}(\mathcal{E}, [V_1, \cdots, V_n], true)$ is $false$, this implies that the solution of $\mathcal{E}$ is $false$.