Information and Computation 249 (2016) 205–236

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Synthesis of positive logic programs for checking a class of definitions with infinite quantification

Francisco J. Galán *, José M. Cañete-Valdeón

Dept. of Languages and Computer Systems, Faculty of Computer Science, Av. Reina Mercedes s/n, 41012, Seville, Spain

ARTICLE INFO

Article history:
Received 21 June 2010
Received in revised form 24 October 2013
Available online 16 June 2016

Keywords:
Program assertion
Logic program
Program synthesis
Unfold/fold transformation

ABSTRACT

We describe a method based on unfold/fold transformations that synthesizes positive logic programs $P(r)$ with the purpose of checking mechanically definitions of the form $D(r) = \forall X(r(X) \Leftrightarrow QY R(X,Y))$ where r is the relation defined by the formula $QY R(X,Y)$, X is a set of variables to be instantiated at runtime by ground terms, QY is a set of quantified variables on infinite domains (Q is the quantifier) and $R(X,Y)$ a quantifier-free formula in the language of a first-order logic theory. This work constitutes a first step towards the construction of a new type of assertion checkers with the ability of handling restricted forms of infinite quantification.

1. Introduction

An assertion is a logic formula representing a condition the program-being-tested must satisfy in each of its executions. Current technology is not able to check assertions containing some kind of infinite quantification [36,40,5,29,33,30,61,50]. However, infinite quantification has shown to be a useful resource for expressing program states in a declarative way [20,17,26,18]. For instance, the following assertion formalizes the subset relation between a program variable L and a program variable S where $member(X,Y)$ is true if a natural number X is included in a sequence of natural numbers Y and false otherwise:

$$\forall E(member(E,L) \Rightarrow member(E,S))$$

Despite its simplicity, the sub-expression $\forall E$ formalizes a quantification over the infinite set of natural numbers. This kind of quantification is not recognized by current assertion checkers.

To palliate this lack of expressivity in current assertion languages, we propose the use of a class of assertion definitions of the form $D(r) = \forall X(r(X) \Leftrightarrow QY R(X,Y))$ where r is the relation defined by the assertion $QY R(X,Y)$, X is a set of variables to be instantiated at runtime by ground terms, QY is a set of quantified variables on infinite domains (Q is the quantifier) and $R(X,Y)$ a quantifier-free formula in the language of a typed first-order logic theory. For instance,

$$D(subset) = \forall L, S(subset(L,S) \Leftrightarrow \forall E(member(E,L) \Rightarrow member(E,S)))$$

is an assertion definition written in the language of the following typed first-order logic theory:

* Corresponding author.
E-mail address: galanm@us.es (F.J. Galán).

http://dx.doi.org/10.1016/j.ic.2016.06.014
0890-5401/© 2016 Elsevier Inc. All rights reserved.

Theory $\mathcal{T}_0$

Types

Nat	generated by	$zero : \rightarrow Nat$	$succ : Nat \rightarrow Nat$
Seq	generated by	$empty : \rightarrow Seq$	$seq : Nat \times Seq \rightarrow Seq$

Predicates

Signature: $id : Nat \times Nat$

Axioms:

1. $id(zero, zero) \Leftrightarrow true$
2. $\forall(id(succ(X), zero) \Leftrightarrow false)$
3. $\forall(id(zero, succ(Y)) \Leftrightarrow false)$
4. $\forall(id(succ(X), succ(Y)) \Leftrightarrow id(X, Y))$

Signature: $member : Nat \times Seq$

Axioms:

5. $\forall(member(E, empty) \Leftrightarrow false)$
6. $\forall(member(E, seq(X, Y)) \Leftrightarrow id(E, X) \vee member(E, Y))$

The *rationale* of our proposal is the following. Given an assertion definition $D(r) = \forall X(r(X) \Leftrightarrow Q\, Y R(X, Y))$, we propose to synthesize a positive logic program $P(r)$ for checking (runtime) assertions of the form $r(X)\theta$, being θ a ground substitution for X. The high-level design of $P(r)$ will depend on the quantifier Q in $D(r)$. If $Q = \forall$ then $P(r)$ will be a program which searches for refutations and if $Q = \exists$ then $P(r)$ will be a program which searches for proofs. In concrete terms, $P(r)$ is implemented by a clause $\forall(r(X) \Leftarrow r_1(X, Y))$ which defines r in terms of a new relation symbol r_1. This new relation is defined by a positive logic program $P(r_1)$ which is synthesized from an auxiliary specification $S(r_1) = \forall X, Y(r_1(X, Y) \Leftrightarrow \neg R(X, Y))$ if $Q = \forall$ or from an auxiliary specification $S(r_1) = \forall X, Y(r_1(X, Y) \Leftrightarrow R(X, Y))$ if $Q = \exists$. For instance, $P(subset_1)$ is synthesized from the following auxiliary specification:

$$S(subset_1) = \forall E, L, S(subset_1(E, L, S) \Leftrightarrow \neg(member(E, L) \Rightarrow member(E, S)))$$

For synthesizing positive logic programs, we follow the so-called transformational approach [19,20]. Most of the program synthesis methods proposed in the literature [3,14,16,19,20,28,43,47,49] are of theoretical nature or require human intervention. By contrast, our proposal is similar in spirit to the ones described in [13,54] where programs are derived from a restricted class of specifications in a completely automatic manner.

Our synthesis method must satisfy two main requirements: (1) the synthesis process must be terminating, that is, $P(r_1)$ must be synthesized in a finite amount of transformation steps and (2) the synthesized programs (i.e. $P(r_1)$) must preserve total correctness wrt the class of goals $\Leftarrow r_1(X, Y)\theta$, that is, $\exists Y r_1(X, Y)\theta$ is true if and only if $P(r_1) \cup \{\Leftarrow r_1(X, Y)\theta\}$ has an SLD-refutation [37].

In our example, $P(assert_1)$ has been synthesized after five transformation steps resulting the following program:

Synthesized program $P(subset_1)$

$\forall(subset_1(E, seq(X, Y), S) \Leftarrow subset_2(E, S) \wedge subset_3(E, X))$
$\forall(subset_1(E, seq(X, Y), S) \Leftarrow subset_1(E, Y, S) \wedge subset_4(E, X))$
$\forall(subset_1(E, empty, S) \Leftarrow subset_5(E, S))$
$\forall(subset_2(E, seq(X, Y)) \Leftarrow subset_2(E, Y) \wedge subset_4(E, X))$
$\forall(subset_2(E, empty) \Leftarrow subset_7)$
$subset_3(zero, zero) \Leftarrow subset_9$
$\forall(subset_3(succ(X), succ(Y)) \Leftarrow subset_3(X, Y))$
$\forall(subset_4(zero, succ(Y)) \Leftarrow subset_9)$
$\forall(subset_4(succ(X), succ(Y)) \Leftarrow subset_4(X, Y))$
$\forall(subset_4(succ(X), zero) \Leftarrow subset_9)$
$\forall(subset_5(E, seq(X, Y)) \Leftarrow subset_5(E, Y) \wedge subset_4(E, X))$
$subset_7 \Leftarrow$
$subset_9 \Leftarrow$

Finally, $P(subset)$ results from the union of $P(subset_1)$ and the clause which defines *subset* in terms of the new relation symbol $subset_1$: $\forall(subset(L, S) \Leftarrow subset_1(E, L, S))$.

As we will see in detail in Sect. 7, for checking a (runtime) assertion $r(X)\theta$, we propose to compute $P(r) \cup \{G\}$, being G the goal $\Leftarrow r(X)\theta$ and $P(r)$ the synthesized program for checking $D(r)$. Again, depending on the quantifier Q in $D(r)$, we can distinguish two cases: (a) for $Q = \forall$, we have that $r(X)\theta$ is false if the empty answer is computed for $P(r) \cup \{G\}$ (*refutation*) and $r(X)\theta$ is true if no answer is computed for $P(r) \cup \{G\}$ (*impossibility of refutation*) and (b) for $Q = \exists$, we have that $r(X)\theta$ is true if the empty answer is computed for $P(r) \cup \{G\}$ (*proof*) and $r(X)\theta$ is false if no answer is computed for $P(r) \cup \{G\}$ (*impossibility of proof*).