On computing invariants for predicate abstraction by SAT-solving

David Monniaux

On computing invariants for predicate abstraction by SAT-solving

David Monniaux
CNRS/VERIMAG

January 29, 2009

Abstract

We propose a method for computing invariants in disjunctive predicate abstraction domains using satisfiability modulo theory (SMT) testing.

1 Introduction

Traditionally, in abstract interpretation, most analyzes compute inductive invariants by iteration: one gets safe overapproximations of reachable states after 1, 2, 3, ..., iterations of the loop. If the domain in which the approximations are computed has finite height, then these iterations terminate on an inductive invariant. If it has infinite height, then possibly one should use so-called widening operators [Cousot and Cousot, 1992]. Widenings introduce extra overapproximation, and long iteration sequences may result in long analysis times. As a result, there has been considerable interest recently in finding alternate methods for computing inductive invariants.

Predicate abstraction [Graf and Saïdi, 1997] is a kind of abstract interpretation where, at the simplest, program states are abstracted to the vector of their Boolean values over a finite set of predicates. In this article, we consider *disjunctive* abstractions: one abstracts a set of program states by a formula in disjunctive normal form, where the literals are taken from the finite set of predicates. If one limits the size of the disjunction, this abstract domain is finite and could find inductive invariants in it by the iteration method. Gulwani et al. [2009] have however shown how to reduce the problem of computing such an invariant to propositional satisfiability problems amenable to efficient solving by modern SAT implementations. In this article, we show how to make this reduction more simply.

2 Templates for predicate abstraction

We look for invariants of the form $I \triangleq \bigvee_{i=1}^{k} C_i$ where the C_i are conjunctions whose conjuncts are taken from a finite set $P = \{\pi_1, \ldots, \pi_{|P|}\}$ of predicates over a theory T, referring to program state variables $q_1, \ldots$ (which can be Boolean or theory-specific). Such an invariant can be characterized by a $k \times |P|$ matrix

of Booleans $a_{i,j}$, meaning that predicate π_j is present in disjunct number i:

$$I \triangleq \bigvee_{i=1}^{k} \bigwedge_{j=1}^{|P|} (a_{i,j} \Rightarrow \pi_j) \tag{1}$$

In other words, the concretization of Booleans $(a_{i,j})_{1\leq i\leq k, 1\leq j\leq |P|}$ is

$$\gamma((a_{i,j})_{1\leq i\leq k, 1\leq j\leq |P|}) = \bigcup_{i=1}^{k} \bigcap_{j=1}^{|P|} \gamma(\pi_j) \tag{2}$$

Let C be a precondition predicate (over variables $q_1, \ldots, q_{n_q}$), let T be a "next state" transition predicate (over variables $q_1, \ldots, q_{n_q}$ for previous state, variables $q'_1, \ldots, q'_{n_q}$ for next state). The correctness condition for I being an invariant is:

$$G \triangleq \forall q_1 \ldots \forall q_{n_q} \forall q'_1 \ldots \forall q'_{n_q} (C \Rightarrow I) \wedge (I \wedge T \Rightarrow I[q'_1/q_1, \ldots, q'_{n_q}/q_{n_q}]) \tag{3}$$

Note that the pointwise ordering on the $(a_{i,j})_{1\leq i\leq k, 1\leq j\leq |P|}$ implies the inverse of the concretization ordering (the ordering induced by γ and $\subseteq$). In order to obtain strong invariants, one should look for acceptable values of $(a_{i,j})_{1\leq i\leq k, 1\leq j\leq |P|}$ with a large amount of true variables.

3 Finding solutions

Formula G, as defined in Eq. 3, is of the following form:

$$G \triangleq \forall q_1 \ldots \forall q_{n_q} F \tag{4}$$

where F is a quantifier-free formula in a certain theory T, and the only free variable of G are propositional variables $\{p_1 \ldots, p_{n_p}\}$. We want solutions for $(p_1 \ldots, p_{n_p})$, preferably with many "true" values.

In this section, for a variable v, we shall note $\tilde{v}$ some "current solution" value for v. We shall show how to reduce computing solutions for G to propositional SAT-solving and to SAT-solving modulo the theory T.

3.1 Eager constraint generation

Since G is a quantified formula where the only free variables are propositional variables, performing quantifier elimination should yield a purely propositional formula. The problem is then reduced to finding a satisfying assignment to a propositional problem (SAT), with a preference for "true" values in the assignment, a task performed by SAT-solvers.

A first strategy is to perform quantifier elimination on G by finding successive solutions to $\neg G$ and projecting them out, as proposed by Monniaux [2008]. Because we eliminate all non-propositional variables, the algorithm can be simplified into:

$O := \mathsf{true}$
$H := \neg F$

while H is satisfiable **do**
  Pick a model $(\tilde{p}_1, \ldots, \tilde{p}_{n_p}, \tilde{q}_1, \ldots, \tilde{q}_{n_q}) \models H$
  $S := \{1, \ldots, n_p\}$
  for all $i \in \{1, \ldots, n_p\}$ **do**
    if $\neg H[\tilde{p}_j/p_j, j \in S \setminus \{i\}, \tilde{q}_1/q_1, \ldots, \tilde{q}_{n_q}/q_{n_q}]$ is not satisfiable **then**
      $S := S \setminus \{i\}$
    end if
  end for
  $\{\left(\bigwedge_{i \in S | \tilde{p}_i = \mathsf{true}} p_i\right) \wedge \left(\bigwedge_{i \in S | \tilde{p}_i = \mathsf{false}} \neg p_i\right) \Rightarrow H[\tilde{q}_1/q_1, \ldots, \tilde{q}_{n_q}/q_{n_q}]\}$
  $\{\left(\bigwedge_{i \in S | \tilde{p}_i = \mathsf{true}} p_i\right) \wedge \left(\bigwedge_{i \in S | \tilde{p}_i = \mathsf{false}} \neg p_i\right) \Rightarrow \exists b_1 \ldots \exists b_{n_b} \exists q_1 \ldots \exists q_{n_q}\ H\}$
  $\{(\forall b_1 \ldots \forall b_{n_b} \forall q_1 \ldots \forall q_{n_q}\ F) \Rightarrow \left(\bigvee_{i \in S | \tilde{p}_i = \mathsf{true}} \neg p_i\right) \vee \left(\bigvee_{i \in S | \tilde{p}_i = \mathsf{false}} p_i\right)\}$
  $H := H \wedge \left(\left(\bigvee_{i \in S | \tilde{p}_i = \mathsf{true}} \neg p_i\right) \vee \left(\bigvee_{i \in S | \tilde{p}_i = \mathsf{false}} p_i\right)\right)$
  $O := O \wedge \left(\left(\bigvee_{i \in S | \tilde{p}_i = \mathsf{true}} \neg p_i\right) \vee \left(\bigvee_{i \in S | \tilde{p}_i = \mathsf{false}} p_i\right)\right)$
end while

Conditions are accumulated into O by adding constraints to the SMT-solver. The final result is that O is a CNF propositional formula (canonical SAT input) equivalent to G.

3.2 Lazy constraint generation

It seems like a waste to generate the full CNF whereas, perhaps, a few constraints suffice to constrain the search space sufficiently to find a solution. We therefore suggest an algorithm that would generate these clauses on demand.[1]

$O := \mathsf{true}$
while O is satisfiable **do**
  Pick a model $(\tilde{p}_1, \ldots, \tilde{p}_{n_p})$ of O (preferably with a large number of "true" variables).
  if $\neg F[\tilde{p}_1/p_1, \ldots, \tilde{p}_{n_p}/p_{n_p}]$ is unsatisfiable **then**
    return $(\tilde{p}_1, \ldots, \tilde{p}_{n_p})$, a model of G
  else
    Pick a model $(\tilde{q}_1, \ldots, \tilde{q}_{n_q})$ of $\neg F[\tilde{p}_1/p_1, \ldots, \tilde{p}_{n_p}/p_{n_p}]$
    $S := \{1, \ldots, n_p\}$
    for all $i \in \{1, \ldots, n_p\}$ **do**
      if $F[\tilde{p}_j/p_j, j \in S \setminus \{i\}, \tilde{q}_1/q_1, \ldots, \tilde{q}_{n_q}/q_{n_q}]$ is unsatisfiable **then**
        $S := S \setminus \{i\}$
      end if
    end for
    $\{(\forall b_1 \ldots \forall b_{n_b} \forall q_1 \ldots \forall q_{n_q}\ F) \Rightarrow \left(\bigvee_{i \in S | \tilde{p}_i = \mathsf{true}} \neg p_i\right) \vee \left(\bigvee_{i \in S | \tilde{p}_i = \mathsf{false}} p_i\right)\}$
    $O := O \wedge \left(\left(\bigvee_{i \in S | \tilde{p}_i = \mathsf{true}} \neg p_i\right) \vee \left(\bigvee_{i \in S | \tilde{p}_i = \mathsf{false}} p_i\right)\right)$
  end if
end while

[1] This is the same idea as for SMT solving: instead of generating the full theory for the atomic predicates appearing in our formulas (that is, all the unsatisfiable conjunctions of these predicates), we generate only those actually needed, lazily.

4 Related works and conclusion

Monniaux [2009] reduced finding postconditions, inductive invariants, strongest postconditions and strongest inductive invariants in certain numerical template domains to quantifier elimination.

References

Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *J. of Logic and Computation*, pages 511–547, August 1992.

Susanne Graf and Hassan Saïdi. Construction of abstract state graphs with PVS. In Orna Grumberg, editor, *Computer-Aided Verification (CAV)*, number 1254 in Lecture Notes in Computer Science, pages 72–83. Springer Verlag, 1997.

Sumit Gulwani, Saurabh Srivastava, and Ramarathnam Venkatesan. Constraint-based invariant inference over predicate abstraction. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 5403 of *Lecture Notes in Computer Science*, pages 120–135. Springer Verlag, 2009. ISBN 978-3-540-93899-6. doi: 10.1007/978-3-540-93900-9_13.

David Monniaux. A quantifier elimination algorithm for linear real arithmetic. In *LPAR (Logic for Programming Artificial Intelligence and Reasoning)*, volume 5330 of *Lecture Notes in Computer Science*, pages 243–257. Springer Verlag, 2008. doi: 10.1007/978-3-540-89439-1_18.

David Monniaux. Automatic modular abstractions for linear constraints. In *POPL (Principles of programming languages)*. ACM, 2009.