Computer Languages, Systems & Structures 47 (2017) 77–103

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Inference of ranking functions for proving temporal properties by abstract interpretation ☆

Caterina Urban [a,b,*], Antoine Miné [a,c]

[a] École Normale Supérieure, Paris, France
[b] ETH Zürich, Zurich, Switzerland
[c] LIP6 – UPMC, Paris, France

ARTICLE INFO

Article history:
Received 20 May 2015
Received in revised form
17 September 2015
Accepted 15 October 2015
Available online 24 October 2015

Keywords:
Static analysis
Abstract interpretation
Liveness
Temporal properties
Ranking functions
Termination

ABSTRACT

We present new static analysis methods for proving *liveness properties* of programs. In particular, with reference to the hierarchy of temporal properties proposed by Manna and Pnueli, we focus on guarantee (i.e., "something good occurs *at least once*") and recurrence (i.e., "something good occurs *infinitely often*") temporal properties.

We generalize the abstract interpretation framework for termination presented by Cousot and Cousot. Specifically, static analyses of guarantee and recurrence temporal properties are systematically derived by abstraction of the program operational trace semantics.

These methods automatically infer *sufficient preconditions* for the temporal properties by reusing existing numerical abstract domains based on piecewise-defined ranking functions. We augment these abstract domains with new abstract operators, including a *dual widening*.

To illustrate the potential of the proposed methods, we have implemented a research prototype static analyzer, for programs written in a C-like syntax, that yielded interesting preliminary results.

1. Introduction

Software verification addresses the problem of checking that programs satisfy certain properties. Lamport, in the late 1970s, suggested a classification of program properties into the classes of *safety* and *liveness* properties [1]. The class of safety properties is informally characterized as the class of properties stating that "something *bad* never happens", that is, a program never reaches an unacceptable state. The class of liveness properties is informally characterized as the class of properties stating that "something *good* eventually happens", that is, a program *eventually* reaches a desirable state.

Manna and Pnueli, in the late 1980s, suggested a more fine grained classification of program properties into a hierarchy [2], which distinguishes four basic classes making different claims about the frequency or occurrence of "something good" mentioned in the informal characterizations proposed by Lamport:

☆ Dedicated to the memory of Radhia Cousot.

* Corresponding author.
E-mail addresses: caterina.urban@inf.ethz.ch (C. Urban), antoine.mine@lip6.fr (A. Miné).

http://dx.doi.org/10.1016/j.cl.2015.10.001
1477-8424/

```
while [1]( 0≤x ) do [2]x := x+1 od
while [3]( true ) do
  if [4]( x≤10 ) then [5]x := x+1 else [6]x := −x fi
od[7]
```

Fig. 1. Program SIMPLE.

- *safety* properties: "something good *always* happens", i.e., the program never reaches an unacceptable state (e.g., partial correctness, mutual exclusion);
- *guarantee* properties: "something good happens *at least once*", i.e., the program *eventually* reaches a desirable state (e.g., total correctness, termination);
- *recurrence* properties: "something good happens *infinitely often*", i.e., the program reaches a desirable state *infinitely often* (e.g., starvation freedom);
- *persistence* properties: "something good *eventually always* happens", i.e., the program eventually reaches and stays in a desirable state (e.g., stabilization).

This paper concerns the verification of programs by static analysis. We set our work in the framework of Abstract Interpretation [3], a general theory of semantic approximation that provides a basis for various successful industrial-scale tools (e.g., Astrée [4]). Abstract Interpretation has to a large extent been concerned with safety properties and has only recently been extended to program termination [5], which is just a particular guarantee property.

In this paper, we generalize the framework proposed by Cousot and Cousot for termination [5] and we propose an abstract interpretation framework for proving *guarantee* and *recurrence* temporal properties of programs. Moreover, we present new static analysis methods for inferring *sufficient preconditions* for these temporal properties. Let us consider the program SIMPLE in Fig. 1, where the program variables are interpreted in the set of mathematical integers. The first loop is an infinite loop for the values of the variable x greater than or equal to zero: at each iteration the value of x is increased by one. The second loop is an infinite loop for any value of the variable x: at each iteration, the value of x is increased by one or negated when it becomes greater than ten. Given the guarantee property "$x=3$ at least once", where $x=3$ is the desirable state, our approach is able to automatically infer that the property is true if the initial value of x is smaller than or equal to three. Given the recurrence property "$x=3$ infinitely often", our approach is able to automatically infer that the property is true if the initial value of x is strictly negative (i.e., if the first loop is not entered).

Our approach follows the traditional method for proving liveness properties by means of a well-founded argument (i.e., a function from the states of a program to a well-ordered set whose value decreases during program execution). More precisely, we build a well-founded argument for guarantee and recurrence properties in an incremental way: we start from the desirable program states, where the function has value zero (and is undefined elsewhere); then, we add states to the domain of the function, retracing the program backwards and counting the maximum number of performed program steps as value of the function. Additionally, for recurrence properties, this process is iteratively repeated in order to construct an argument that is also invariant with respect to program execution steps so that even after reaching a desirable state we know that the execution will reach a desirable state again. We formalize these intuitions into *sound* and *complete* guarantee and recurrence semantics that are systematically derived by abstract interpretation of the program operational trace semantics.

In order to achieve effective static analyses, we further abstract these semantics. Specifically, we leverage existing numerical abstract domains based on piecewise-defined ranking functions [6–8] by introducing new abstract operators, including a *dual widening*. The piecewise-defined ranking functions are attached to the program control points and represent an *upper bound* on the number of program execution steps before the program reaches a desirable state. They are automatically inferred through backward analysis and yield *sufficient preconditions* for the guarantee and recurrence temporal properties. We prove the soundness of the analysis, meaning that all program executions respecting these preconditions indeed satisfy the temporal properties, while a program execution that does not respect these preconditions might or might not satisfy the temporal properties.

To illustrate the potential of our approach, let us consider again the program SIMPLE in Fig. 1. Given the guarantee property "$x=3$ at least once", the piecewise-defined ranking function inferred at program control point 1 is

$$\lambda x.\begin{cases} -3x+10 & x<0 \\ -2x+6 & 0\le x \wedge x\le 3 \\ \text{undefined} & \text{otherwise} \end{cases}$$

which bounds the wait (from the program control point 1) for the desirable state $x=3$ by $-3x+10$ program execution steps when $x<0$, and by $-2x+6$ execution steps when $0\le x\wedge x\le 3$. The analysis is inconclusive when $3<x$. In this case, when $3<x$, the guarantee property is never satisfied. Thus, the precondition $x\le 3$ induced by the domain of the ranking function is the weakest precondition for "$x=3$ at least once". Given the recurrence property "$x=3$ infinitely often", the piecewise-defined ranking function at program point 1 bounds the wait for the *next* occurrence of the desirable state $x=3$ by $-3x+10$