Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 93 (2005) 281–288
Information Processing Letters
www.elsevier.com/locate/ipl

Efficient weakest preconditions

K. Rustan M. Leino

Microsoft Research, Redmond, WA, USA

Received 17 November 2003; received in revised form 3 November 2004

Communicated by F.B. Schneider

In memory of Edsger W. Dijkstra

Abstract

Desired computer-program properties can be described by logical formulas called verification conditions. Different mathematically-equivalent forms of these verification conditions can have a great impact on the performance of an automatic theorem prover that tries to discharge them. This paper presents a simple weakest-precondition understanding of the ESC/Java technique for generating verification conditions. This new understanding of the technique spotlights the program property that makes the technique work.

Keywords: Program correctness; Formal semantics; Automatic theorem proving

0. Introduction

Various computer-program checking tools and verification tools generate *verification conditions*, logical formulas whose validity reflect the correctness of the program under analysis. Each verification condition is then passed to a mechanical theorem prover or some suite of decision procedures. The Extended Static Checkers for Modula-3 and for Java are examples of program checkers built around this architecture [5,8,10].

There are many mathematically equivalent ways to formulate a verification condition, and which formulation one uses can have a dramatic impact on the performance of the program-checking system. The ESC/Modula-3 and ESC/Java projects have explored techniques for formulating verification conditions that substantially improve the way they are handled by the underlying automatic theorem prover. The variation of the technique used in ESC/Java is described by Flanagan and Saxe [9]. Their paper compares the ESC/Java technique with the well-known verification-condition technique of *weakest preconditions* [6]. In this paper, I show that the ESC/Java

E-mail address: leino@microsoft.com (K.R.M. Leino).

0020-0190/$ – see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.ipl.2004.10.015

technique *is* in fact the technique of weakest preconditions with the additional use of a certain weakest-precondition property that holds only for a restricted class of programs.

1. Weakest preconditions

Let's start by reviewing weakest preconditions and the problem with their traditional application. We consider a simple language like the following, which is representative of the intermediate language used in ESC/Java [11]:

$$\begin{aligned} S, T ::= {} & Id := Expr \\ | \ & \textbf{assert } Expr \\ | \ & \textbf{assume } Expr \\ | \ & S \,;\, T \\ | \ & S \,[]\, T \end{aligned}$$

The assignment statement $x := E$ sets program variable x to the value of E. The assert and assume statements are no-ops if the given expression evaluates to *true*. If the expression evaluates to *false*, the assert statement is an irrevocable error (the execution *goes wrong*) and the assume statement is a partial command that does not start (the execution *blocks*) [12]. Every execution in our simple language either blocks, goes wrong, or terminates. The statement $S \,;\, T$ is the sequential composition of S and T, where T is executed only if S terminates, and $S \,[]\, T$ is the arbitrary choice between S and T. The statements of the simple language are rich enough to encode loops declared with loop invariants and procedure calls declared with procedure specifications (cf. [11,1]). For this paper, it suffices to know that a common program statement like

if B **then** S **else** T **end**

is encoded as the choice statement

(**assume** $B \,;\, S$) [] (**assume** $\neg B \,;\, T$)

in the simple language.

The *weakest conservative precondition* of a statement S with respect to a predicate Q on the post-state of S, denoted $wp(S, Q)$, is a predicate on the pre-state of S, characterizing all pre-states from which every non-blocking execution of S does not go wrong and terminates in a state satisfying Q. Similarly, the *weakest liberal precondition* of S with respect to Q, denoted $wlp(S, Q)$, characterizes the pre-states from which every non-blocking execution of S either goes wrong or terminates in a state satisfying Q. The connection between *wp* and *wlp* is described by the following equation, which holds for every statement S [6]:

$$(\forall Q \bullet wp(S, Q) \ \equiv \ wp(S, true) \wedge wlp(S, Q)). \tag{0}$$

The semantics of the statements in the simple language are defined by the following weakest preconditions, for any predicate Q [6,12]:

Stmt	$wp(Stmt, Q)$	$wlp(Stmt, Q)$
$x := E$	$Q[x := E]$	$Q[x := E]$
assert E	$E \wedge Q$	$E \Rightarrow Q$
assume E	$E \Rightarrow Q$	$E \Rightarrow Q$
$S \,;\, T$	$wp(S, wp(T, Q))$	$wlp(S, wlp(T, Q))$
$S \,[]\, T$	$wp(S, Q) \wedge wp(T, Q)$	$wlp(S, Q) \wedge wlp(T, Q)$

(1)

where $Q[x := E]$ says about E what Q says about x, that is:

$$Q[x := E] \ = \ \textbf{let } x = E \textbf{ in } Q \textbf{ end}$$