A Bounded Verification Tool for Java

Stefan Koppier

January 31, 2019

Introduction

Contents

Chapter 1

Phases of the Tool

1.1 Lexing and Parsing

1.2 Analysis

1.2.1 Syntax Transformation

1.2.2 Control Flow Analysis

We perform the Control Flow Analysis (CFA) in a way similar to that described by Nielson et al. [1].

We define the CFA as:

$$
\begin{aligned}
init([x = e]^l) &= l \\
init([S_1];\ [S_2]) &= init(S_1) \\
init([\textbf{assert}\ e\ g]^l) &= l \\
init([\textbf{assume}\ e\ g]^l) &= l \\
init([\textbf{break}\ x]^l) &= l \\
init([\textbf{continue}\ x]^l) &= l \\
init(\textbf{if}\ ([e]^l)\ \{S\}) &= l \\
init(\textbf{if}\ ([e]^l)\ \{S_1\}\ \textbf{else}\ \{S_2\}) &= l \\
init(\textbf{while}\ ([e]^l)\ \{S\}) &= l \\
init(\textbf{for}\ (;\ [e]^l;\ \ldots)\ \{S\}) &= l \\
init(\textbf{for}\ ([i]^{l'};\ [e]^l;\ \ldots)\ \{S\}) &= l'
\end{aligned}
$$

$$
\begin{aligned}
final([x = e]^l) &= \{l\} \\
final([S_1];\ [S_2]) &= final(S_2) \\
final([\textbf{assert}\ e\ g]^l) &= \{l\} \\
final([\textbf{assume}\ e\ g]^l) &= \{l\} \\
final([\textbf{break}\ x]^l) &= \{l\} \\
final([\textbf{continue}\ x]^l) &= \{l\} \\
final(\textbf{if}\ ([e]^l)\ \{S\}) &= final(S) \\
final(\textbf{if}\ ([e]^l)\ \{S_1\}\ \textbf{else}\ \{S_2\}) &= final(S_1) \cup final(S_2) \\
final(\textbf{while}\ ([e]^l)\ \{S\}) &= \{l\} \cup breaks_0(S) \\
final(\textbf{for}\ (\ldots;\ [e]^l;\ \ldots)\ \{S\}) &= \{l\} \cup breaks_0(S) \\
final(x:\ \textbf{while}\ ([e]^l)\ \{S\}) &= \{l\} \cup breaks_x(S) \\
final(x:\ \textbf{for}\ (\ldots;\ [e]^l;\ \ldots)\ \{S\}) &= \{l\} \cup breaks_x(S)
\end{aligned}
$$

$$
\begin{aligned}
flow([x = e]^l) &= \emptyset \\
flow([S];\ [\textbf{break}\ x]^l) &= flow(S) \\
flow([S];\ [\textbf{continue}\ x]^l) &= flow(S) \\
flow([S_1];\ [S_2]) &= flow(S_1) \cup flow(S_2) \cup \{(l, init(S_2) \mid \in final(S_1))\} \\
flow([\textbf{assert}\ e\ g]^l) &= \emptyset \\
flow([\textbf{assume}\ e\ g]^l) &= \emptyset \\
flow([\textbf{break}\ x]^l) &= \emptyset \\
flow([\textbf{continue}\ x]^l) &= \emptyset \\
flow(\textbf{if}\ ([e]^l)\ \{S\}) &= flow(S) \cup (l, init(S)) \\
flow(\textbf{if}\ ([e]^l)\ \{S_1\}\ \textbf{else}\ \{S_2\}) &= flow(S_1) \cup flow(S_2) \cup (l, init(S_1)) \cup (l, init(S_2)) \\
flow(\textbf{while}\ ([e]^l)\ \{S\}) &= todo \\
flow(\textbf{for}\ (;\ [e]^l;\ \ldots)\ \{S\}) &= todo \\
flow(\textbf{for}\ ([i]^{l'};\ [e]^l;\ \ldots)\ \{S\}) &= todo
\end{aligned}
$$

1.2.3 Reachability Analysis

Chapter 2

CProver

2.1 Properties

array bounds	test
pointer	test
division by zero	test
arithmetic over- and underflow	test
shift greater than bit-width	test
floating-point for +/-Inf	test
floating-point for NaN	test
user assertions	test

Chapter 3

Translation of Java to C

3.1 Primitive data types

Table 3.1: Equivalence of primitive Java data types.

Type	Description	C equivalent
boolean	true or false	_Bool
char	16-bit Unicode value	
byte	8-bit signed integral value	__int8
short	16-bit signed integral value	__int16
int	32-bit signed integral value	__int32
long	64-bit integral value	__int64
float	IEEE 754 64-bit floating point value	float
double	IEEE 754 32-bit floating point value	double

Bibliography

[1] Flemming Nielson, Hanne R Nielson, and Chris Hankin. *Principles of program analysis*. Springer, 2015.