Program Analysis and Verification
Course 0368-4479
Final Project

Noam Rinetzky

Due 15/August/2021

Design, prove correct, implement, and document the following static analyses. The analyses should be conservative and terminating. You should also write five interesting test programs for each analysis. The test program should demonstrate the strengths and weaknesses of each analysis.

Programming language. Consider a language of programs which manipulate natural numbers (including zero) with the following primitive commands:

C	::=	`skip` | i := j | i := K | i := ? | i := j + 1 | i := j -1
	|	`assume` E | `assert` ORC
E	::=	i = j | i != j | i = K | i != K | `TRUE` | `FALSE`
ORC	::=	(ANDC) | (ANDC) ORC
ANDC	::=	b | b ANDC

In the above syntax, K is s constant and ? is an unknown arbitrary value. An `assert` command aborts the program if the assertion does not hold. It takes as an argument a sequence of parenthesized expressions and should be interpreted as a disjunction: The assertion holds if at least one of the parenthesized expressions hold. A parenthesized expression should be interpreted as a conjunction of atomic predicates b whose syntax is defined soon: For the parenthesized expression to hold all of its atomic predicates should be true.

Assume that the program is written in a CFG syntax,

$$P = \overline{i}\ \overline{\ell\ C\ \ell},$$

where $\overline{i}$ is the sequence of variable names (strings comprised of lowercase letters) and $\ell\ C\ \ell$ is an edge in the control flow graph. For example, the following program increments i and j until their value is equal to that of n:

```
n i j

L0  n := ?         L1
L1  i := 0         L2
L2  j := 0         L3
L3  assume(i = n)  L6
L3  assume(i != n) L4
L4  i := j + 1     L5
L5  j:= i          L3
L6  skip           L7
```

Simplifying assumptions You may assume the following:

1. The only one node with no incoming edges is the one from which the execution starts.
2. Every node in the control flow graph has at most two out-going edges.
3. If a node has more than one outgoing edges than these edges are annotated with `assume` commands.
4. Programs are syntactically legal, i.e., there is no need to handle syntax errors.

1 Parity Analysis

Design an abstract domain and transformers which can prove that a program does not violate assertions pertaining to the parity of each variables. Specifically, consider atomic predicates of the following form:

b ::= `EVEN` i | `ODD` i

For example, assume we replace the `skip` command in the example program with the following command:

`assert` (`ODD` i `ODD` j) (`EVEN` i `EVEN` j)

The analysis should be able to prove that the program does not violate the assertion, i.e., that i has the same parity as j when the program reaches L6.

2 Summation Analysis

Design an abstract domain and transformers which can prove that a program does not violate assertions pertaining to the the equality of the summation of two sets of variables. Specifically, consider atomic predicates of the following form:

$$\text{b} \quad ::= \quad \texttt{SUM}\ \bar{i} = \texttt{SUM}\ \bar{j}$$

For example, if we replace the `skip` command in the example program with the following command:

$$\texttt{assert}\ (\texttt{SUM}\ i\ j\ = \ \texttt{SUM}\ j\ n)$$

then the analysis should be able to prove that the program does not violate this assertion.

3 Combined Analysis

Combine the Parity and Summation analyses in three different ways and find examples that show the cost/precision differences between the analyses.