Exam for WSSA'03 Course on *Static Analysis*

David A. Schmidt

Computing and Information Sciences Department
Kansas State University
Manhattan, KS 66506 USA.

Answer 3 of the following 4 questions.

Note: In the questions that follow, I reference slides from the lectures, where I use numberings from the PS-files posted on my web page. Those numbers might not match exactly the ones in your notes, so please consult my web site if you are uncertain of a reference.

Question 1. Introduction to abstraction and static analysis

From Lecture 1, Slide 17 ("A Starting Point: Trace-Based Operational Semantics") shows the trace-based operational semantics for a while-loop program.

Here is a syntax for the language in which the program was written:

```
  P : ProgramPoint
  C : Command
  E : Expression
  F : BasicExpression
  x : Variable
  N : Numeral

C ::=  P: x := E   |  P: while E do C   |  C1 ; C2

E ::=  F1 + F2   |  F1 < F2
F ::=  N   |  x
N ::=  0   |  1  | 2  | ...

P ::=  p0   |  p1   |  p2  | ...
x ::=  a   |  b   |  c   | ...
```

Write an algorithm that translates a program written in the above language into a set of state transition rules that compute the program's "concrete" execution semantics on integers.

Next, write an algorithm that translates a program written in the above language into a set of state transition rules that compute the program's abstract semantics on the abstract data values, $\{even, odd\}$. (See Slide 18: "We can abstractly interpret, say, for polarity").

Finally, write a proof that, for every program, the translated rules that compute the abstract values, $\{even, odd\}$, of variables are sound with respect to the rules that compute the concrete semantics. (Hint: Read Slides 20 and 21: "The underlying abstract intepretation semantics.")

Question 2. Foundations of Abstract Interpretation

From Lecture 2, read Slides 14 and 15, "Closed binary relations." Use the definition of γ on Slide 14 to prove the Proposition stated on Slide 15.

Question 3. Mechanics of Static Analysis

Slide 16 of Lecture 3 defines *forwards-necessarily reaching definitions analysis.*

Rewrite the definitions of $inReach(p_i)$ and $f_i^{\#}$ on that slide so that the analysis compute forwards-*possibly* reaching definitions analysis.

Use your definition to rewrite the abstract transfer functions for the program on Slide 9 and recompute the flow-analysis table for the program, which should look similar to the one on Slide 11.

Question 4. Static Analysis: Applications and Logics

In Lecture 4, Slide 17 ("Constructing an abstract logic") shows how to generate a distributive complete lattice. Prove Items 1 and 2 on Slide 18.

Next, prove that the distributive lattice generated from $\{neg, zero, pos\}$, where

$n \models neg$ for all $n < 0$
$n \models pos$ for all $n > 0$
$0 \models zero$

quotiented by the induced function, γ (that is, $A \equiv_\gamma B$ iff $\gamma(A) = \gamma(B)$), is exactly the **Signs** lattice, displayed at the bottom of Slide 18.