CS 232 Static Program Analysis, Winter 2020

Instructor: Jens Palsberg

Final Exam, Mar 16, 2020

ID

Name

This exam consists of 21 questions, divided into categories as follows.

Question category		**Maximum**
One-line-answer questions	5 questions, each max 2 points	10 points
Short-answer questions	10 questions, each max 3 points	30 points
Draw graphs	3 questions, each max 10 points	30 points
Write an input program	2 questions, each max 10 points	20 points
Write constraints	1 question, max 10 points	10 points
Total		**100 points**

Background Information

Here is a reminder about the type system known as: "Partial Types for Lambda-Calculus". This type system uses the following types (called *partial types*)

$t \ ::= \ t \to t \mid \top$

and the following definition of subtyping:

$t \leq \top$, for all t

$s \to t \leq s' \to t'$ if and only if $s' \leq s \ \wedge \ t \leq t'$

and the following rule for using subtyping in a type derivation:

if $A \vdash e : s$ and $e \leq t$, then $A \vdash e : t$.

One-Line-Answer Questions

For each of the following five questions, a good answer can be given in a single line.

Question 1
In the setting of type inference with simple types for lambda-calculus, what is the time complexity of deciding whether a λ-term has simple type?

Question 2
Consider the set S of λ-terms that each has a simple type. Consider also the set P of λ-terms that each has a partial type. What can we say about the relationship between S and P?

Question 3
For a function $f : D \to D$, where D is a finite set with partial order and a bottom element, a sufficient condition for that f has a fixed point is that f is _____.

Question 4
Consider the domain $D = \{notNull, unknown\}$ and the relation $\leq$ which is defined as follows: $\forall d \in D : d \leq d$ and $notNull \leq unknown$. Is $(D, \leq)$ a partial order with a bottom element?

Question 5
Consider the set PT of programs that type check with Type inference with Partial Types. Consider also the set SA of programs that 0CFA-based safety analysis successfully checks. What can we say about the relationship between PT and SA?

Short-Answer Questions

For each of the following ten questions, a good answer can be given in no more than three sentences.

Question 6

For the program $\lambda f.\lambda x.f(f\ x)$, some of the type constraints that express typability in simply-typed lambda-calculus are: $[\![\lambda f.\lambda x.f(f\ x)]\!] = [\![f]\!] \to [\![\lambda x.f(f\ x)]\!]$ and $[\![\lambda x.f(f\ x)]\!] = [\![x]\!] \to [\![f(f\ x)]\!]$. Which constraints are missing to complete the picture?

Question 7

For the program $\lambda x.(x\ x)$, we consider type inference with partial types for lambda-calculus. The types variables are

$$\{[\![\lambda x.(x\ x)]\!], x, [\![x_2]\!], [\![x_2]\!], [\![x\ x]\!]\}$$

where we think of $(x\ x)$ as $(x_1\ x_2)$. What are the type constraints that express typability of the above program with partial types for lambda-calculus?

Question 8

The top type in a type system is a close cousin of the _____ in a 0CFA-based safety analysis.

Question 9

Let C and A be lattices. In the case of C, the order is written as $\leq_C$, while in the case of A, the order is written as $\leq_A$. A Galois connection is a relationship between two monotone functions $\alpha : C \to A$ and $\gamma : A \to C$ that satisfy _____, for all $x \in C$ and $y \in A$.

Question 10
Let C and A be finite lattices. In the case of C, the order is written as $\leq_C$, while in the case of A, the order is written as $\leq_A$. Let $f : C \to C$ be a monotone function. Suppose we have a Galois connection (α, γ), where $\alpha : C \to A$ and $\gamma : A \to C$. We can define an abstract version of f, which we will call g and which will have functionality $g : A \to A$, as follows.

Question 11 What are the relative strengths and weaknesses of context-insensitive analysis and context-sensitive analysis?

Question 12 What are the relative strengths and weaknesses of type inference with simple types and type inference with partial types?

Question 13 How can we leverage 0CFA to build a call graph?

Question 14 What are the advantages and disadvantages of doing static program analysis by first mapping the program to a system of constraints and then solving the constraints?

Question 15 A tool developer is considering to change a key component in a tool for Java: the idea is to replace the entire 0CFA component with an implementation of Heintze and McAllester's fast flow analysis. What is your advice and why?

Draw Graphs

Question 16 Here is a Java program:

```
class MainClass {
  public static void main(String[] a){
      System.out.println(new A().m(10));
  }
}

class A {
  int m(int i) {
    return new C().p(i+1) + 1;
  }
}
```

```
class B extends A {
  int m(int i) {
    return new C().p(2) * i;
  }
}

class C {
  int p(int j) {
    return j+2;
  }
}
```

a) For the above Java program, draw the call graph that is based on Class Hierarchy Analysis.

b) For the above Java program, draw the call graph that is based on 0CFA.

Question 17 Here is a program in the interrupt calculus.

```
// main:
imr = imr or 1111b
loop { imr = imr or 1111b }

handler 1 {
   imr = imr and 1011b
   imr = imr or 1000b
   iret
}
```

```
handler 2 {
   imr = imr and 1101b
   imr = imr or 1101b
   iret
}
handler 3 {
   imr = imr and 1110b
   imr = imr or 1000b
   iret
}
```

a) On a separate piece of paper, draw the static-analysis graph, as covered in class, for this program. Submit the separate piece of paper together with the rest. Write a brief note here that the grader should indeed look for that separate piece of paper.

b) Does a static analysis that uses the graph in (a) determine that the stack is bounded for the above program? If so, what is the maximum stack size according to the static analysis and why, and if not, why?

Question 18 Consider the λ-term: $G = (\lambda^1 x.\ x\ (x\ x))\ (\lambda^2 y.y)$

a) Show the graph for G that is used by the Heintze/McAllester quadratic-time flow analysis algorithm.

b) What is the label set for G produced by the algorithm?

Write an Input Program

Question 19

a) Sketch a Java program that can be used to demonstrate the difference in analysis ability between 0CFA and 1CFA (where we can think of 1CFA as a context-sensitive version of 0CFA). Specifically, the Java program should have a call site $e.m()$ such that if we run 0CFA, we find that the call has multiple targets, while if we run a context-sensitive version of 0CFA, we find that the call has a single target.

b) Explain why your Java program illustrates the difference between 0CFA and 1CFA.

Question 20 A tool developer wants to implement 0CFA but got the constraints wrong. Here is an explanation of what went wrong. Consider each pair of a call site $e_1.m(e_2)$ and a method declared in a class `C` and with argument type `s` and with return type `t`, such as:

```
class C {
  ...
  t void m(s x) { return e; }
  ...
}
```

For such a pair of a call site and a method, the tool developer used these constraints:

$$C \in [\![e_1]\!] \Rightarrow \begin{cases} [\![e_2]\!] \subseteq [\![x]\!] \wedge \\ [\![e_1.m(e_2)]\!] \subseteq [\![e]\!] \end{cases}$$

a) Sketch a Java program that can be used as input to demonstrate that something is wrong.

b) Explain why your Java program demonstrates that something is wrong.

Write Constraints

Question 21

Consider the sequences of assignment statements and the expressions generated by this grammar:

$$\begin{array}{lrcl} \text{(Statement)} & s & ::= & \mathit{skip} \mid v = e;\ s \\ \text{(Expression)} & e & ::= & c \mid v \mid e \times e \end{array}$$

where *skip* is the empty statement, $v = e$; is an assignment; c ranges over integer constants, v ranges over local variables, and $\times$ denotes integer multiplication. We want a constraint-based static analysis that for a given expression determines whether e evaluates to an even number or an odd number.

a) Write down an approach to generating constraints that are sound and reasonably accurate.

b) Explain why your constraints are sound.

c) Explain why your constraints are reasonably accurate.

CS 232 Static Program Analysis, Winter 2020

Instructor: Jens Palsberg

Final Exam, Mar 16, 2020

Your Statistics

Score: 98/100 98%

Class Statistics

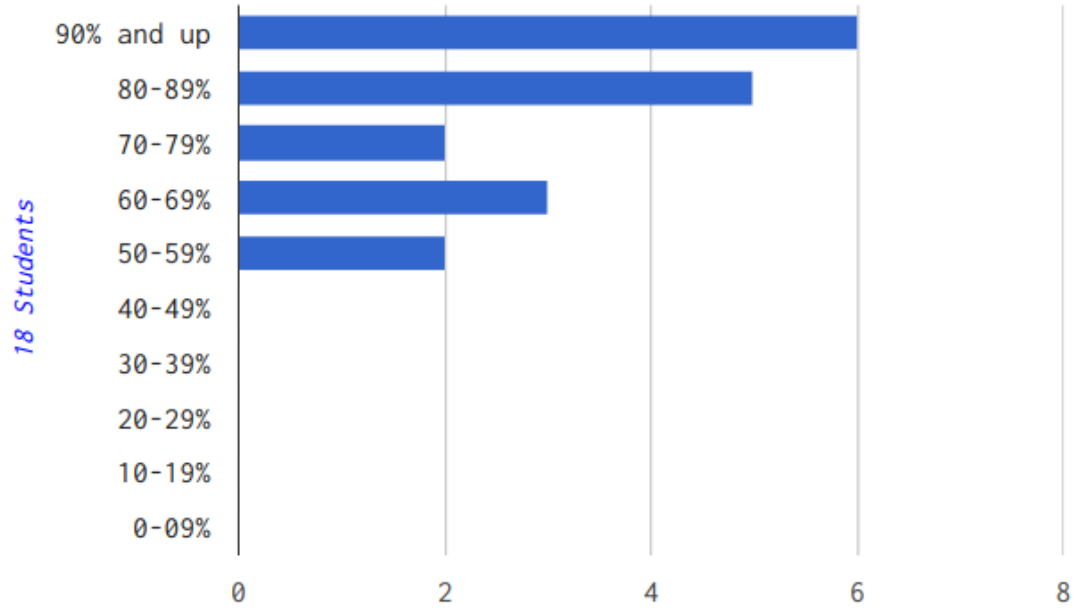
High: 99.00000 99%

Low: 55.00000 55%

Median: 83.5 84%

Mean: 80.5 80%

Final exam



1. $O(n)$
2. $S \subseteq P$
3. monotonic and continuous
4. Yes
5. $PT \subseteq SA$
6. $[\![f]\!] = [\![x]\!] \longrightarrow [\![f\ x]\!]$
   $[\![f]\!] = [\![f\ x]\!] \longrightarrow [\![f\ (f\ x)]\!]$
7. $x \leq [\![x_1]\!]$ $\qquad$ $x \longrightarrow [\![x\ x]\!] \leq [\![\lambda x.\ x\ x]\!]$
   $x \leq [\![x_2]\!]$ $\qquad$ $[\![x_1]\!] \leq [\![x_2]\!] \longrightarrow [\![x\ x]\!]$
8. The set of all possible values in the program. Or from the paper: "the largest possible set of flow information."
9. $\alpha(x) \leq_A y \iff x \leq_C \gamma(y)$
10. $g(y) = \alpha(f(\gamma(y)))$, or $g = \alpha \circ f \circ \gamma$.
11. Context-insensitive analyses are generally faster, but "good enough" for use cases such as reachability. (For instance, the context-insensitive 0-CFA runs in polynomial time.) Context-sensitive analyses are slower, but provide more precise answers. (1-CFA, which is context-sensitive, runs in exponential time.)

12. Type inference with simple types can run very quickly (linear time for existence check), but does not work with as many programs. Partial types provide subtyping which allows it to work with more PLs / programs that require support for it, but make analysis slower ($O(n^3)$).

13. Assume that we know the set of values that each expression can take (through 0-CFA). Then derive the set of procedures that a given procedure can call in turn using the flow information.

For Java:

given $e.m(\ldots)$ in method $C::p$, $[\![e]\!]$ = set of classes e can be

add edge $(C::p, D::m)$ for each $D \in [\![e]\!]$

For λ-calculus: $[\![e]\!]$ = label set of e

given $\lambda^{l} x. \ldots e_1 e_2 \ldots$

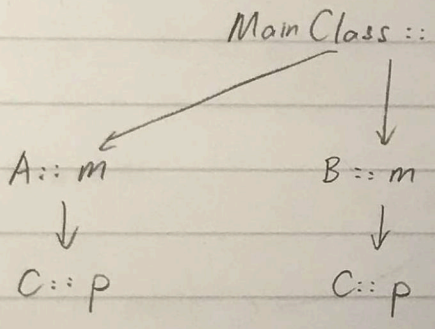
add edge (l, p) for each $p \in [\![e_1]\!]$

14. Advantage: Neatly separates the constraint generation stage from the solving stage. Solving the constraints could be studied for its own sake. Existing algorithms for constraint solving could be leveraged w/o knowledge of internal details.

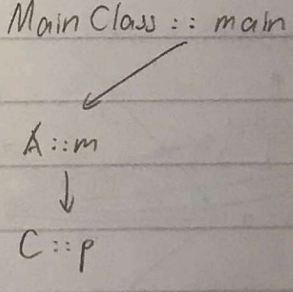
Disadvantage: Certain static analyses cannot be easily reduced to constraints, such as stack space analysis.

15. Instead of wholesale replacement, run both algorithms in parallel. Heintze and McAllester analysis may not terminate for some valid Java programs, but if it does it tends to be faster than the traditional approach. Best to try both methods and pick the one that finishes faster.

16. a)



b)



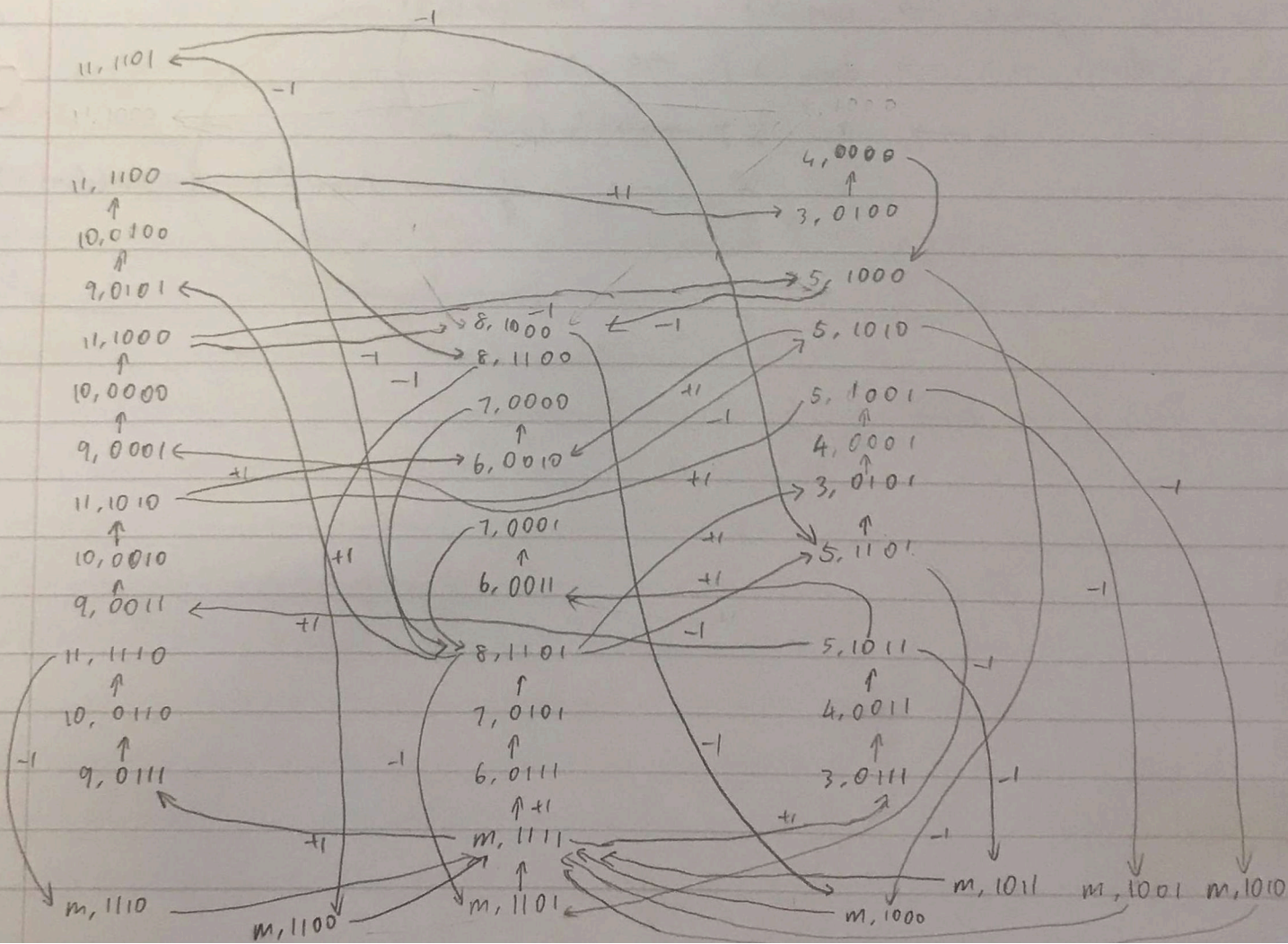
17.a)

```
handler 1 {
3    imr = imr and 1011b
4    imr = imr or 1000b
5    iret
}
```

```
handler 2 {
6    imr = imr and 1101b
7    imr = imr or 1101b
8    iret
}
```

```
handler 3 {
9    imr = imr and 1110b
10   imr = imr or 1000b
11   iret
}
```

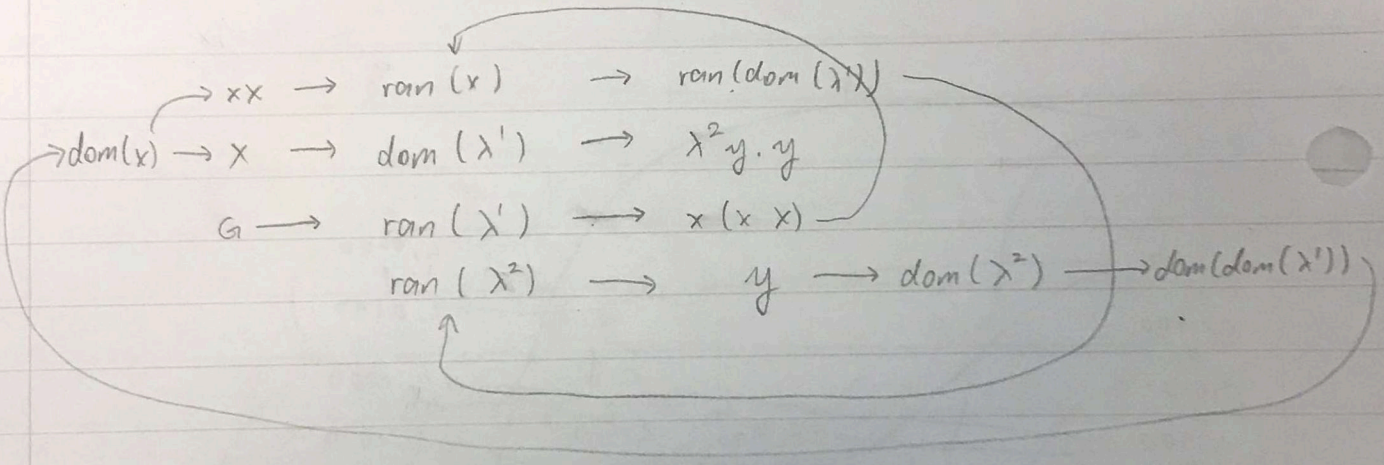
each state: x, imr = imr value occurred at line x.



17. b) The static analysis shows that the stack is bounded. The maximum stack size is [5].

$$m, 1111 \xrightarrow{+1} 3, 0111 \xrightarrow{*} 5, 1011 \xrightarrow{+1} 9, 0011 \xrightarrow{*} 11, 1010$$
$$\xrightarrow{+1} 6, 0010 \xrightarrow{*} 8, 1101 \xrightarrow{+1} 3, 0101 \xrightarrow{*} 5, 1001$$
$$\xrightarrow{+1} 9, 0001 \xrightarrow{*} 11, 1000 \xrightarrow{-1} 3, 1000 \xrightarrow{-1} 8, 1000$$
$$\xrightarrow{-1} 11, 1000 \xrightarrow{-1} 5, 1000 \xrightarrow{-1} m, 1000 \longrightarrow m, 1111$$

18. a)



b) $L(G) = \{2\}$

```
1  class Main {
2      public static void main (String [] args) {
3          A a, b, c;
4          c = new A();
5          b = c.id (new B());
6          a = c.id (new A());
7          a.p();
8      }
9  }
```

```
10 class A {
11     public A id (A arg) {
12         return arg;
13     }
14     public void p () {
15         System.out.println ("a");
16     }
17 }
18 class B extends A {
19     public void p () {
20         System.out.println ("b");
21     }
22 }
```

Under 0-CFA, there is only a single copy of A::id, so there is only a single $[\![arg]\!]$ set variable. This variable must be $\{A, B\}$, as we have passed an object of each type to A::id. It follows that $[\![c.id(new\ A())]\!] \supseteq [\![arg]\!] = \{A, B\}$, and a.p() can have two targets, A::p and B::p.

With 1-CFA, there are two copies of A::id: the first one (id_1) resulting from the callsite at line 5, and A::id_2 resulting from line 6. In particular, we have $[\![arg_2]\!]$ can only be $\{A\}$, and $[\![a]\!] = [\![c.id_2(new\ A())]\!] = [\![arg_2]\!] = \{A\}$. So a.p() can only have A::p as target.

(7)

20. Consider

```
1  class M {
2      public static void main(String[]args){
3          A a, b;
4          a = new A();
5          b = a.id(a);
6      }
7  }
```

```
8  class A {
9      public A id(A arg){
10         return arg;
11     }
12 }
```

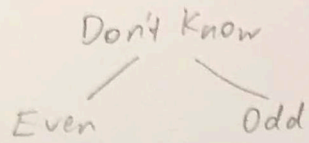
Using the given constraints, we have the following as least solution:

$$[\![a.id(a)]\!] = \emptyset.$$

This is because the consequente $[\![e_1.m(e_2)]\!] \subseteq [\![e]\!]$ is flipped, leading to $[\![e_1.m(e_2)]\!] = \emptyset$ being a valid solution for any call.

In the context of our program, this means that $[\![b]\!] = \emptyset$ is output by the tool, which is clearly incorrect. It should be that $[\![b]\!] = [\![a.id(a)]\!] = \{A\}$.

21. a) Let $D = \{Even, Odd, Don't know\}$:



be the abstract domain. We use a flow-sensitive analysis in the style of liveness analysis. Let $in[s, v] \in D$ be the state of local variable v before the execution of statement s. Let $out[s, v] \in D$ be the state of v after executing s. Let $res[s, e]^{e \in D}$ be the state of e that is a part of statement s. Let $succ(s)$ be the successor of s. Let g be the first statement.

For statement $v = e$ as s:

$$out[s, v] = res[s, e] \quad (1)$$

$$out[s, v'] = in[s, v'] \quad \forall\, v' \neq v \quad (2)$$

$$in[succ(s), v'] = out[s, v''] \quad \forall\, v'' \quad (3)$$

For g:

$$in[g, v] = Don't\ Know \quad \forall\, v. \quad (4)$$

Now define res:

for c as e: $res[s, e] = Odd$ if c is odd, or Even otherwise. (5)

for v as e: $res[s, e] = in[s, v]$ (6)

for $e_1 \otimes e_2$ as e: $res[s, e] = res[s, e_1] \otimes res[s, e_2]$ (7)

$\otimes$	OP_1: DK	E	O
OP_2: DK	DK	DK	DK
E	DK	E	E
O	DK	E	O

$\otimes : D \times D \longrightarrow D$ is defined by the table on the left

b) The constraints are sound, as they closely approximate program execution. At the very beginning, all variables are presumed unknown. This is rule (4). Thereafter, since there are no jump constructs in the language, each assignment statement is followed by the subsequent statement, such that the output of the previous is immediately visible in the next. This is (3). When an assignment occurs, the state of the RHS is computed based on the laws of

(continued)

(a)

arithmetics (5), (7), with any variable references resolved (6). The assignment is done in (1), while all unchanged variables inherit their current values (2).

c) These constraints are quite accurate, since for a "strict" program at any program point we know exactly the state of each variable: if it is even or odd. We assume that the vast majority of input programs will be "strict", as "initializing-before-use" is encouraged or required for most programming languages.

A "strict" program is one such that every variable is initialized to some non-DontKnow value before it is used. We say a variable v is used when (6) looks at in[s, v] for some s.

leave