Programming Languages and Compilers
Qualifying Examination

Tuesday, September 17, 2018

Answer 4 of 6 questions.

GENERAL INSTRUCTIONS

1. Answer each question in a separate book.
2. Indicate on the cover of *each* book the area of the exam, your code number, and the question answered in that book. On *one* of your books list the numbers of *all* the questions answered. *Do not write your name on any answer book.*
3. Return all answer books in the folder provided. Additional answer books are available if needed.

POLICY ON MISPRINTS AND AMBIGUITIES

The Exam Committee tries to proofread the exam as carefully as possible. Nevertheless, the exam sometimes contains misprints and ambiguities. If you are convinced that a problem has been stated incorrectly, mention this to the proctor. If necessary, the proctor will contact a representative of the area to resolve problems during the *first hour* of the exam. In any case, you should indicate your interpretation of the problem in your written answer. Your interpretation should be such that the problem is nontrivial.

1 Logique de Hoare

In Hoare logic, we use the notation

$$\models \{Q\}\ p\ \{R\}$$

to denote that if the program p starts executing from a state in set Q, then, if it terminates, its final state is in set R. Assume that programs in our setting are sequences of assignment instructions, conditionals, and while loops. Note that we do not have procedures or memory allocation.

For the rest of this question, imagine that you have an almighty machine M that, given a Hoare triple $\{Q\}\ p\ \{R\}$, determines whether it is valid or not—i.e., whether $\models \{Q\}\ p\ \{R\}$.

1.1 Encoding Assertions

Suppose that you would like to extend the programming language with assertion statements of the form $\mathsf{assert}(C)$, where C is some Boolean condition. As is standard, if the program reaches an assertion statement and C is false, then the program terminates returning an assertion violation.

Describe how would you use the machine M to check if a program can ever experience an assertion violation. *Hint: you are expected to transform the program into one without assertions to be able to use M.*

1.2 Encoding Equivalence

Suppose that you have two programs p_1, p_2, both of which are of type $\mathbb{Z} \to \mathbb{Z}$. Describe how you could use the machine M to prove the p_1 and p_2 are equivalent, meaning that for every input they produce the same output.

1.3 Encoding Associativity

Suppose that you have a program p that is a binary function of type $\mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$. Describe how you could use the machine M to prove the p encodes an associative function.

1.4 Encoding Upperbounds

Suppose that every assignment instruction takes 1ms to execute, and all other instructions take 0ms. Given a program p with a single input variable $n \in \mathbb{Z}^{\geq 0}$, we would like to prove that the program executes in time at most n^2 ms. Describe how you could use the machine M to prove that fact.

2 Evaluation Strategies

Modern programming languages mostly use three types of evaluation strategies for variable passing: 1) call-by-value, 2) call-by-reference, and 3) call-by-need (or lazy evaluation).

2.1 Part 1:

Describe in words how these three strategies work.

2.2 Part 2:

Provide a program that terminates when evaluated using call-by-value, but not when evaluated using call-by-reference. Explain why one program terminates and the other one does not.

2.3 Part 3:

Provide a program that terminates when evaluated using call-by-name, but not when evaluated using call-by-value. Explain why one program terminates and the other one does not.

2.4 Part 4:

Provide a strategy for implementing call-by-need evaluation. Describe in necessary detail what data structures you would use and their efficiency implication. Also describe the limitations of your strategy.

3 Language Semantics

This question concerns the semantics of a simple object-oriented language. Consider the following language of expressions:

$$e ::= x \mid \texttt{new}\ \tau \mid \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \mid \texttt{while}\ e_1\ \texttt{do}\ e_2 \mid x\ \texttt{:=}\ e \mid e_1.m(e_2)$$

Type names τ are drawn from a fixed, finite set of class identifiers $\mathbb{T}$. Let the class hierarchy be represented by a relation $isub \in \mathbb{T} \times \mathbb{T}$ where $isub(\tau_1, \tau_2)$ indicates that τ_1 is an immediate subclass of τ_2. There is a distinguished class called `Bool` $\in \mathbb{T}$ representing Boolean values. This class has two subclasses, `True` and `False`, respectively representing truth and falsehood.

Variable names x are drawn from a fixed, finite set of global variable identifiers $\mathbb{X}$. Let the types of these variables be given by a function $varType : \mathbb{X} \to \mathbb{T}$ where $varType(x) = \tau$ indicates that the variable named x has type τ.

Method names m are drawn from a fixed, finite set of method identifiers $\mathbb{M}$. Notice that every method takes exactly one argument in addition to the object on which the method is being called. Let the set of predefined methods be described by the relation $argType \in \mathbb{T} \times \mathbb{M} \times \mathbb{T} \times \mathbb{T}$ where $argType(\tau_1, m, \tau_2, \tau_3)$ if class τ_1 has or inherits a method m that takes a value of type τ_2 as its one argument, and yields a result of type τ_3.

3.1 Well-Formed Class Hierarchy

What are the loosest restrictions that *isub* must obey in order for it to represent a well-formed, single-inheritance, single-rooted class hierarchy?

What are the loosest restrictions that *argType* must obey in order for it to represent well-formed inheritance of methods with no overloading?

3.2 Static Typing Judgments

Typing rules for this language should allow judgments of the form "$\vdash e : \tau$", meaning that the expression e has type τ. For example, the following would be reasonable typing judgments for variables and assignments:

$$\frac{varType(x) = \tau}{\vdash x : \tau} \qquad \frac{varType(x) = \tau \quad \vdash e : \tau}{\vdash x\ \texttt{:=}\ e : \tau}$$

Observe that we do not use a typing environment Γ. The set of identifiers is global and fixed, as are their types, so no typing environment is needed.

Write typing judgment rules for the `if` and method-invocation constructs of our language. Your answers should follow the style of the examples above.

3.3 Operational Semantics

Assume that the set of concrete values computed at run time is named $\mathbb{V}$ and consists of elements of the form "`Obj(`τ`)`". That is, the expression "`new` τ" always yields the concrete value `Obj(`τ`)`.

An operational semantics for this language should allow judgments of the form "$(\sigma_1, e) \Rightarrow (\sigma_2, v)$", meaning that evaluating e in state σ_1 yields the value v and leaves the system in state σ_2. For example, the following would be a reasonable operational-semantics judgment for the assignment construct of our language:

$$\frac{(\sigma_1, e) \Rightarrow (\sigma_2, v)}{(\sigma_1, x \texttt{ := } e) \Rightarrow (\sigma_2[x \mapsto v], v)}$$

Here the "$\mapsto$" notation represents function updating: "$f[a \mapsto b]$" is a new function that maps a to b but that maps any other a' distinct from a to $f(a')$.

Write the evaluation rules for an operational semantics of the `if` and `while` constructs of our language. Your answers should follow the style of the examples above.

3.4 Static vs. Dynamic

Suppose you have a static judgment that $\vdash e : \tau_1$ and a dynamic operational-semantics judgment that $(\sigma_1, e) \Rightarrow (\sigma_2, \texttt{Obj(}\tau_2\texttt{)})$. **What relationship is there between the two judgments and under what conditions?**

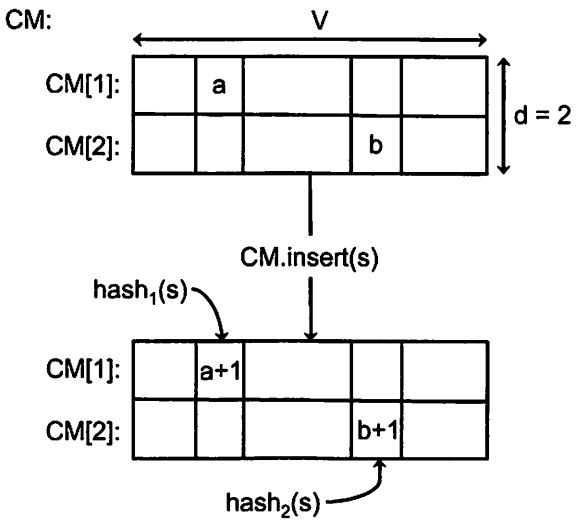

Figure 1: Diagram showing the effect of performing the operation CM.insert(s).

4 Representing Multi-Sets

In this question, array indices start with 1, not 0.

A *multi-set* (sometimes called a "bag") is a collection of elements, possibly with repetitions. For instance, $\{a, a, a, b, b, c\}$ is a multi-set that contains three occurrences of "a," two occurrences of "b," and a single occurrence of "c." We say that multi-set M_1 *over-approximates* M_2—denoted by $M_1 \geq M_2$—if, for each element $s \in U$, the number of occurrences of s in M_1 is $\geq$ the number of occurrences of s in M_2. For instance, $\{a, a, a, a, b, b, c, d\} \geq \{a, a, a, b, b, c\}$.

This question concerns some ideas that one might use in an abstract-interpretation system to represent multi-sets. In particular, the concrete domain *Conc* is the set of multi-sets over U, ordered by $\geq$. The least element is the empty multi-set: $\{\}$.

(So that we do not have to abstract counts, we will implicitly assume that there is a bound on the number of occurrences of each item in each multi-set.)

A *CountMin structure* is like a counting version of a Bloom filter. (You do not need to know what a Bloom filter is to answer this question.) A CountMin structure consists of a two-dimensional array of size $d \times V$, each cell of which can hold a count. For each of the d rows, there is a different hash function; each hash function $\text{hash}_j : U \rightarrow \text{index}$, $1 \leq j \leq d$, hashes an element $s \in U$ to an index in the range $[1 \ldots V]$. (See the upper part in Fig. 1.)

We can use a CountMin structure to represent a multi-set. For instance, the empty multi-set is represented by the CountMin structure in which all counts in the 2D-array are 0. Suppose that CountMin structure CM represents multi-set M, and we wish to insert an element $s \in U$ into M. We denote this operation as CM.insert(s). Fig. 1 indicates how CM.insert(s) is performed. The idea is that multi-set insertion can be *safely* simulated in CM by feeding s to each hash function to obtain d indices: we take each index value $i_j = \text{hash}_j(s)$ and increment the count in cell CM$[j][i_j]$ by 1.

To query how many occurrences of $s \in U$ there are in the multi-set represented by CM, we perform

$$\text{CM.query}(s) =_{\text{df}} \min\{\text{CM}[j][\text{hash}_j(s)] \mid 1 \leq j \leq d\}.$$

A multi-set can also be thought of as a function from the universe of elements U to $\mathbb{N}$. For instance, $\{a, a, a, b, b, c\}$ corresponds to the function $[a \mapsto 3, b \mapsto 2, c \mapsto 1]$, where all other elements of U map to 0. Thus, a concretization function γ : CountMin $\rightarrow$ MultiSet, specifying what multi-set a given CountMin structure represents, can be defined as follows:

$$\gamma(\text{CM}) =_{\text{df}} \lambda s.(\text{CM.query}(s)). \tag{1}$$

4.1

Let *intersection* of multi-sets be defined as follows:

$$M_1 \cap M_2 =_{\text{df}} \lambda s.\min(M_1(s), M_2(s)),$$

and *union* of multi-sets be defined as follows:

$$M_1 \cup M_2 =_{\text{df}} \lambda s.\max(M_1(s), M_2(s)).$$

Turning to CountMin structures, we want the *meet* operation ($\sqcap$) to over-approximate multi-set intersection,

$$\gamma(\text{CM}_1 \sqcap \text{CM}_2) \geq \gamma(\text{CM}_1) \cap \gamma(\text{CM}_2), \tag{2}$$

and the *join* operation ($\sqcup$) to over-approximate multi-set union,

$$\gamma(\text{CM}_1 \sqcup \text{CM}_2) \geq \gamma(\text{CM}_1) \cup \gamma(\text{CM}_2). \tag{3}$$

(i) Give definitions of meet and join of CountMin structures, and (ii) show that the desired properties hold (assuming that γ is defined as in Eqn. (1)).

4.2

For this part and Part (c), suppose that a CountMin structure has just two rows (i.e., $d = 2$). Suppose that our universe U is just $\{a, b, c, d\}$, and that due to the quirks of the hash function used, we have

$$\begin{array}{llllll} \text{hash}_1(a) & = & 3; & \text{hash}_2(a) & = & 3 \\ \text{hash}_1(b) & = & 4; & \text{hash}_2(b) & = & 4 \\ \text{hash}_1(c) & = & 3; & \text{hash}_2(c) & = & 4 \\ \text{hash}_1(d) & = & 4; & \text{hash}_2(d) & = & 3 \end{array}$$

Suppose that CM is the following CountMin structure:

$$\begin{array}{lll} \text{CM}[1] & = & 0, 0, 1, 1, 0, 0, \ldots \\ \text{CM}[2] & = & 0, 0, 1, 1, 0, 0, \ldots \end{array}$$

What is the value of $\gamma(\text{CM})$ according to Eqn. (1)?

4.3

The theory of abstract interpretation often makes use of Galois connections to connect sets of concrete values in a concrete domain *Conc* with abstract values in an abstract domain *Abs*.

Definition 4.1 *A* Galois connection *is defined by a pair of functions that relate elements in domains Conc and Abs as follows:*

$$\begin{aligned} \alpha &: Abs \to Conc \\ \gamma &: Conc \to Abs \end{aligned}$$

such that

$$\text{for all } c \in Conc, (\gamma \circ \alpha)(c) \sqsupseteq_{Conc} c \tag{4}$$

$$\text{for all } a \in Abs, (\alpha \circ \gamma)(a) \sqsubseteq_{Abs} a \tag{5}$$

In particular, we will use the $\geq$ relation on multi-sets for $\sqsupseteq_{Conc}$. For the $\sqsubseteq_{Abs}$ relation on CountMin structures, we define $a_1 \sqsubseteq_{Abs} a_2$ iff $a_1 \sqcup a_2 = a_2$

Suppose that we define $\alpha(S)$, where S is a multi-set, by the following program:

```
CM = new CountMin      // array initialized to all-zeros
for each occurrence in S of an s ∈ U do
    CM.insert(s)
od
return CM
```

For CM as defined in Part (b), fill in the following table:

$$\alpha(\{a,b\}) = \begin{bmatrix} \text{CM[1]} = \\ \text{CM[2]} = \end{bmatrix}$$

$$\alpha(\{a,c\}) = \begin{bmatrix} \text{CM[1]} = \\ \text{CM[2]} = \end{bmatrix}$$

$$\alpha(\{b,d\}) = \begin{bmatrix} \text{CM[1]} = \\ \text{CM[2]} = \end{bmatrix}$$

$$\alpha(\{c,d\}) = \begin{bmatrix} \text{CM[1]} = \\ \text{CM[2]} = \end{bmatrix}$$

$$\alpha(\{a,b,c\}) = \begin{bmatrix} \text{CM[1]} = \\ \text{CM[2]} = \end{bmatrix}$$

$$\alpha(\{a,b,c,d\}) = \begin{bmatrix} \text{CM[1]} = \\ \text{CM[2]} = \end{bmatrix}$$

(Note: The question continues on the next page.)

4.4

In light of what you found in Parts (b) and (c), show that γ and α, as defined above, do **not** define a Galois connection between the space *Conc* of multi-sets and the space *Abs* of CountMin structures. In particular, (i) say whether it is the property from Eqn. (4) or Eqn. (5) that **fails to hold**, and (ii) show that the other property **does hold**.

5 Simulation Relation

Let $[k]$ be the set $\{1, 2, \cdots, k\}$. A *labeled graph* is a 4-tuple (V, I, E, L), where V is a finite set of vertices, $I \subseteq V$ is a set of initial states, $E \subseteq V \times V$ is the set of edges, and $L : V \to [k]$ labels each vertex with an element of the set $[k]$.

Given two labeled graphs $G_1 = (V_1, I_1, E_1, L_1)$ and $G_2 = (V_2, I_2, E_2, L_2)$ a relation $S \subseteq V_1 \times V_2$ is called a *simulation relation* iff it satisfies the following conditions:

1. For all vertices $v \in I_1$ there exists a vertex $v' \in I_2$ such that $(v, v') \in S$.
2. For all $(v_1, v_2) \in S$ and all $w_1 \in V_1$, if there exists $(v_1, w_1) \in E_1$, then there exists $w_2 \in V_2$ such that $(v_2, w_2) \in E_2$ and $(w_1, w_2) \in S$.

Let $Sim(G_1, G_2)$ be all the simulation relations between the labeled graphs G_1 and G_2.

5.1

Show that the set $Sim(G_1, G_2)$ is non empty.

5.2

Let S_1 and S_2 be two simulation relations in the set $Sim(G_1, G_2)$. Prove that $S_1 \cap S_2$ is also a simulation relation in the set $Sim(G_1, G_2)$.

5.3

Show that there is a smallest simulation S^* relation in the set $Sim(G_1, G_2)$ (i.e. for all $S \in Sim(G_1, G_2)$, $S^* \subseteq S$).

5.4

Show an algorithm for computing S^* that runs in time polynomial in n, where $n = |V_1| + |V_2|$.

6 Security

6.1

Languages like C that do not guarantee array-bounds checking and that allow pointer arithmetic can lead to programs that are vulnerable to certain kinds of malicious attacks. Consider the program shown in Figure 2. How could a malicious user cause a buffer overrun?

6.2

Explain how a malicious user can exploit buffer overrun vulnerability in a program.

6.3

We will sketch a static analysis technique to detect buffer overruns. Each buffer *buf* (variable of type `char *`) is associated with two range-valued variables $R_{len}(buf)$ and $R_{alloc}(buf)$ (one for the length and other for the allocated space). A program variable i (of type `int`) is associated with a single range-valued variable $R(i)$ (representing the possible values of i). Intuitively, if $R_{len}(buf) = (n, m)$, then the minimum and maximum length of the buffer *buf* are n and m respectively. Similarly, $R_{buf}(buf) = (n, m)$ indicates that the minimum and maximum allocated space for the buffer *buf* are n and m respectively. Each program statement generates a subset constraint on range-valued variables. For example, consider the following statement:

```
strcpy(a,b)
```

Since b is copied into a, the following constraint is generated:

$$R_{len}(b) \subseteq R_{len}(a)$$

Show the range constraints generated by the program given in Figure 2.
Note: You will have to use sets of constraints that model the library functions `fgets` and `strcpy`. I am assuming you know the semantics of `strcpy`. Description of `strlen` and `fgets` is given below:

```
strlen() returns the number of characters upto, but not including
till the nearest '\0'

char *fgets(char *s, int size, FILE *stream);
fgets() reads in at most size-1 characters
from stream and stores them into the buffer pointed to by s.
Reading stops after an EOF or a newline.  If a newline is read,
it is stored into the buffer.  A '\0' is stored
after the last character in the buffer.
```

6.4

Solving a system S of range constraints means finding the "tightest" possible ranges that respect

```
(1)  main(int argc, char* argv[]){
(2)      char header[2048], buf[1024],
               *cc1, *cc2, *ptr;
(3)      int counter;
(4)      FILE *fp;
(5)      ...
(6)      ptr = fgets(header, 2048, fp);
(7)      cc1 = copy_buffer(header);
(8)      ptr = fgets (buf, 1024, fp);
(9)      cc2 = copy_buffer(buf);
(10) }
(11)
(12) char *copy_buffer(char *buffer){
(13)     char *copy;
(14)     copy = (char *) malloc(strlen(buffer));
(15)     strcpy(copy, buffer);
(16)     return copy;
(17) }
```

Figure 2: Example Program

the constraints in S. For example, consider the following system S_1 of range constraints:

$$\begin{aligned} (4,4) &\subseteq R(a) \\ (8,8) &\subseteq R(b) \\ R(b) &\subseteq R(a) \end{aligned}$$

The following assignment of ranges is the "tightest" that respects constraints in S_1:

$$\begin{aligned} R(a) &= (4,8) \\ R(b) &= (8,8) \end{aligned}$$

Notice that the following assignment of ranges also respects the constraints in S_1, but does not assign the "tightest" possible ranges.

$$\begin{aligned} R(a) &= (1,9) \\ R(b) &= (5,8) \end{aligned}$$

Suppose there is procedure P for solving a system of range constraints, i.e., procedure P returns the "tightest" possible ranges that respect the constraints in a system S. How will you use procedure P to discover buffer overruns?

Part (e): Consider the range constraints from part(c). Give a graph algorithm to solve the range constrains generated in part(c). You need only explain your algorithm with respect to the specific set of constraints generated in part (c).

Subject: Fwd: questions about PL qual
From: Justin Hsu <email@justinh.su>
Date: 9/17/2018, 12:20 PM
To: Angela Thorp <thorp@cs.wisc.edu>

Hi Angela,

Can you please put the two clarifications (under Q1 and Q3.1) on the testing room blackboard for the PL qual?

Thanks so much and sorry for the extremely late notice---we just noticed a few ambiguities!

Cheers,

Justin

---------- Forwarded message ---------
From: Justin Hsu <email@justinh.su>
Date: Mon, Sep 17, 2018, 12:13 PM
Subject: Re: questions about PL qual
To: Aws Albarghouthi <aws@cs.wisc.edu>
Cc: Loris D'Antoni <loris@cs.wisc.edu>, Thomas Reps <reps@cs.wisc.edu>, Ben Liblit <liblit@cs.wisc.edu>, Somesh Jha <jha@cs.wisc.edu>

OK. Unless I hear otherwise (in the next 45 mins), we're making the following two clarifications:

Q1: Assume all programs are terminating.

Q3.1: The argType relation should support subtyping both in the input and output type positions.

I'll try to put these notes up on the blackboard in the testing room.

Justin