Programming Languages and Compilers
Qualifying Examination

Monday, February 1, 2016

Answer 4 of 6 questions.[1]

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]Note: The full exam had six questions. Participants only answered the four questions included in this document.

Question 1: (Function Caching and Hash-Consing)

Note: *Function caching* is also known as *memoization*.

Part (a):

For each of the following statements, say whether it is true or false, and give a brief justification of your answer. For purposes of computing costs, you may assume that the cost of a hashed lookup is constant.

1. For certain functional programs, the introduction of function caching will cause an exponential-time algorithm to become a linear-time algorithm.

2. For certain functional programs, the introduction of hashed-consing will cause an exponential-time algorithm to become a linear-time algorithm.

Part (b):

Describe the conditions that must hold for function caching to be a viable option in a given program, or in the run-time environment of a given programming language.

Part (c):

Describe the conditions that must hold for hash-consing to be a viable option in a given program, or in the run-time environment of a given programming language.

Part (d):

Although function caching and hash-consing both use a hash-based dictionary to perform lookups, the role of the dictionary is different in these two implementation techniques. Explain how the hashed dictionary is used for different purposes in the two techniques.

Suggestion: Give your answer in terms of the invariant or invariants that each technique maintains.

Part (e):

Explain why using both function caching and hash-consing in a given program (or in the run-time environment of a given programming language) can produce a "non-linear" speed-up.

That is, for the sake of discussion suppose that introducing function caching alone (i.e., without introducing hash-consing) cuts the running time of an application from 100 seconds to 70 seconds. Suppose further that introducing hash-consing alone (i.e., without introducing function caching) cuts the running time of the application from 100 seconds to 80 seconds. The question posed in this part asks you to explain why it is possible that when function caching and hash-consing are both introduced, the application might run in 5 seconds (rather than in 56 seconds, which one would expect if they have independent, linear effects on the running time).

Question 2: (Lambda Calculus)

Recall that lambda expressions can be reduced either using normal order reduction (NOR)—i.e., reduce the leftmost-outermost redex—or using applicative-order reduction (AOR)—i.e., reduce the leftmost-innermost redex. Two reduction strategies, S_1 and S_2 are considered to be equivalent iff for every lambda expression e, either both S_1 and S_2 reduce e to normal form, or neither does (i.e., neither terminates).

Part (a):

What are the advantages of NOR over AOR and vice versa? Give examples to illustrate your answers.

Part (b):

Is the strategy "reduce the rightmost-outermost redex" equivalent to NOR? If yes, briefly justify your answer. If no, give an example of a lambda expression for which one strategy leads to a normal form while the other strategy fails to terminate.

Part (c):

Is the strategy "reduce the rightmost-innermost redex" equivalent to AOR? If yes, briefly justify your answer. If no, give an example of a lambda expression for which one strategy leads to a normal form while the other strategy fails to terminate.

Part (d):

Recall that lambda-term a is *equal to* lambda-term b when a can be converted into b via a sequence of (possibly interleaved) α, β-reductions and α, β-expansions.

For each of the following statements, say whether it is true or false, and give a brief justification of your answer.

1. Every lambda term is equal to a lambda term that is in normal form.
2. Every lambda term is equal to a lambda term that is not in normal form.
3. Every lambda term has some lambda term as its fixed point.
4. Every lambda term is the fixed point of some lambda term.
5. There is a lambda term that is its own fixed point.

Question 3: Power of Static Analyses

Suppose we want to statically prevent reading uninitialized local variables in C. We have several different levels of analysis sophistication to choose from. In all variants below, note that we are interested in analyses that can prove that the undesired behavior *must* happen, not merely that it *may* happen.

Part (a): Flow-Insensitive

Give an example code fragment for which a **flow-insensitive** analysis is powerful enough to prove that an uninitialized local variable *must* be read.

Part (b): Flow-Sensitive

Give an example code fragment for which a **flow-sensitive** analysis is powerful enough to prove that an uninitialized local variable *must* be read, but for which a flow-insensitive analysis is too weak. Briefly explain why flow-insensitivity is not sufficient.

Part (c): Path-Sensitive

Give an example code fragment for which a **path-sensitive** analysis is powerful enough to prove that an uninitialized local variable *must* be read, but for which a flow-sensitive analysis is too weak. Briefly explain why flow-sensitivity is not sufficient.

Part (d): Context-Sensitive

Give an example code fragment for which a **context-sensitive** analysis is powerful enough to prove that an uninitialized local variable *must* be read, but for which a context-insensitive analysis is too weak. Briefly explain why context-insensitivity is not sufficient.

Question 4: Temporal Logic

Recall that given a set of atomic propositions AP, the set of all LTL (Linear Temporal Logic) formulas over AP is generated by the syntax:

$$\phi ::= true \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc\phi \mid \phi_1 \mathrm{U} \phi_2$$

where $a \in AP$. Informally, $\bigcirc\phi$ is true if ϕ is true in the next state and $\phi_1 \mathrm{U} \phi_2$ is true iff ϕ_1 is true until ϕ_2 becomes true.

The semantics of LTL can be defined over infinite sequences over the alphabet 2^{AP}. Given an infinite sequence $\sigma = s_0 s_1 s_2 \ldots$ we write $\sigma \vDash \phi$ to say that the formula ϕ holds on the infinite sequence σ. For example, $s_0 s_1 s_2 \ldots \vDash a$ iff $a \in s_0$.

Part (a):

Formally define the semantics of all the LTL operators.

Part (b):

Given a sequence $\sigma = s_0 s_1 s_2 \ldots$, let $\sigma[i]$ be the sequence $s_i s_{i+1} \ldots$. Typically LTL also allows the following two extra operators:

- $\Box\phi$, which is true for a sequence σ iff, for every $i \geq 0$, ϕ is true for $\sigma[i]$;
- $\Diamond\phi$, which is true for a sequence σ iff, for some $i \geq 0$, ϕ is true for $\sigma[i]$.

For this part of the question:

1. Show how $\Box\phi$ and $\Diamond\phi$ can be defined using only the operators appearing in regular LTL (see definition above). Informally argue for the correctness of your encoding.
2. Show how $\Box\phi$ can be defined using just $\Diamond\phi$ and negation. Informally argue for the correctness of your encoding.

Part (c):

Recall that given a set of atomic propositions AP, the set of all CTL (Computation Tree Logic) formulas over AP is generated by the syntax:

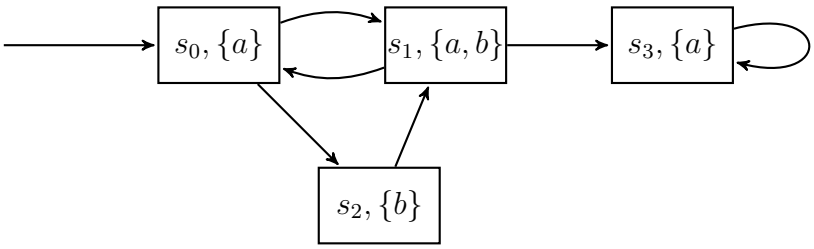
$$\phi ::= true \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \mathbf{E}\psi \mid \mathbf{A}\psi$$
$$\psi ::= \mathbf{X}\ \phi \mid \phi_1\ \mathbf{U}\ \phi_2 \mid \mathbf{G}\ \phi$$

where $a \in AP$.

The semantics of CTL state formulas (those in the grammar ϕ) is defined over the set of states Q of a Kripke structure $M = (Q, \rightarrow)$, while the semantics of the path formulas (those in the grammar ψ) is defined in terms of paths in the Kripke structure M. For example, **EX** ϕ is true at a

state q, if there exists a path starting at state q and such that the next element following q satisfies ϕ. Similarly, $\mathbf{A}(\phi_1\ \mathbf{U}\ \phi_2)$ is true at a state q if all paths starting at state q satisfy $\phi_1\ \mathbf{U}\ \phi_2$. $\mathbf{EG}\ \phi$ is true at a state q if there exists a path $\sigma = q_0 q_1 \ldots$ starting at state q, such that $q = q_0$, and for every $i \geq 0$, q_i satisfies ϕ.

Let M be the following Kripke structure with atomic propositions a and b. Each box gives the name of a state (s_i) and the subset of atomic propositions that are true in that state.



The next two problems ask you to identify the states of M that satisfy some CTL formula and decide whether M itself satisfies the same formula. Use the systematic CTL model checking algorithm; solving using your intuition alone will earn no points. Show your work, not just a final answer. For example, if you rewrite a formula into an equivalent form, show the rewritten form. List the states satisfying each subformula as you build up to an answer for the original formula. Full points require both a correct final answer as well as work clearly demonstrating the systematic, algorithmic steps leading to that answer.

- Identify the set of states $\{s : M, s \vDash \mathbf{E}(a\mathbf{U}b)\}$. Does $M \vDash \mathbf{E}(a\mathbf{U}b)$?
- Identify the set of states $\{s : M, s \vDash \mathbf{EX}\ \mathbf{E}(b\ \mathbf{U}\ (\mathbf{AG}\ a))\}$. Does $M \vDash \mathbf{EX}\ \mathbf{E}(b\ \mathbf{U}\ (\mathbf{AG}\ a))$?

Part (d):

Recall that two temporal logic formulas ϕ and ψ are equivalent iff for every Kripke structure M, M is a model of ϕ iff M is a model of ψ. For example, the CTL formula $AF\phi$ and the LTL formula $F\phi$ are equivalent.

Draw a Kripke structure that shows that the CTL formula $\mathbf{AF}\ \mathbf{AX}\ a$ and the LTL formula $\Diamond \bigcirc a$ are not equivalent.