

CS263. Final Exam

(solutions due on April 30 at 10:30AM.)

April 25, 2007

This is an open book exam. You must work alone for this exam. Use your judgment as to what constitutes fair use of the resources at your disposition. For example, browsing the web for the solutions is not fair. If you use papers other than the readings for the class, you must cite those in your solutions. Please try to keep your solutions simple. You may lose points for unnecessarily complicated solutions.

Exercise 1: Consider the following nondeterministic language of guarded commands. You will have to provide the static and dynamic semantics for this language. The language contains integer expressions (literals, variables, and simple arithmetic expressions), boolean expressions and commands. There are local variables, and a single global variable g , all of integer type.

$e ::= n \mid x \mid e_1 + e_2 \mid e_1 * e_2$	Integer expressions
$\text{true} \mid \text{false} \mid e = 0 \mid e \geq 0$	Boolean expressions
$c ::= \text{skip}$	Do nothing
$\mid c_1; c_2$	Sequencing
$\mid x := e$	Assignment
$\mid e \rightarrow c$	Execute c if e is true
$\mid e_1 \rightarrow c_1 \# \dots \# e_n \rightarrow c_n$	Non-deterministic choice
$\mid \text{do } e \rightarrow c \text{ od}$	Looping
$\mid \text{call } P$	Function call
$\mid \text{local } x \text{ in } c$	Local variables

There are several unusual commands in this language. First there is the guarded command “ $e \rightarrow c$ ” that can make progress only when the boolean guard e is satisfied. In that case the command c is executed, otherwise the command gets stuck. A choice command $e_1 \rightarrow c_1 \# \dots \# e_n \rightarrow c_n$ chooses

non-deterministically one of the commands whose guards are satisfied. The command gets stuck if all guards are false. Getting stuck like this is not considered an error in this language (think of this as looping forever at that point).

Another unusual construct is the looping construct “do $e \rightarrow c$ od”. This repeatedly executes c as long as e is true. If e is false, then the loop terminates.

To simplify the language we are assuming that we have only one (possibly recursive) procedure in our system. The procedure is called P and its body is c_P . Furthermore we assume that the procedure does not have arguments and results but that it communicates with the context through the global variable g . We cannot have procedure calls inside expressions but only as commands.

The last form of commands is used to introduce new local variables with *static* scoping.

1. Static semantics. You must define the static semantics of the guarded command language. You must have two typing judgments, one of for expressions, and one for commands.

2. Dynamic semantics. Next, you must define the dynamic semantics of the guarded command language. Define an appropriate notion of execution state and define a small-step semantics for the commands and a natural-style (large-step) semantics for expressions.

3. Weakest preconditions. Define the axiomatic semantic rules for all commands in the form of weakest preconditions. Since this is a non-deterministic language, the weakest precondition is the predicate that is satisfied exactly by those states from which *all* possible terminating executions establish the postcondition. A stuck execution is treated as non-terminating. If you need to use the fixed-point theorem you need not prove the cpo status of the underlying domain and the continuity of the function involved.

Exercise 2: Consider an arbitrary imperative language of commands c . What are the relationships (i.e. equivalences or implications) between the following pairs of predicates (wp is the weakest precondition function):

$$\begin{array}{ll} wp(c, P_1) \wedge wp(c, P_2) & \text{and } wp(c, P_1 \wedge P_2) \\ wp(c, P_1) \vee wp(c, P_2) & \text{and } wp(c, P_1 \vee P_2) \\ \neg wp(c, P) & \text{and } wp(c, \neg P) \end{array}$$

Does the answer depend on whether the language is deterministic or non-deterministic? Prove your answers.

Exercise 3: Consider the IMP command “`while b do x:=e`” (where the only variable that can occur in b and e is x). You will have to prove for this command soundness and completeness results for the verification condition. Recall that the verification condition is simpler to compute because it relies on programmer provided loop invariants. Prove the following two facts:

1. For any choice of the loop invariant, the verification condition for this command is always stronger than the weakest precondition.
2. There is a choice of the loop invariant for which the verification condition is equivalent with the weakest precondition.

You must first state the above facts formally (as closed logic formulas). Then you must prove them (if you use induction make sure you state precisely what kind of induction you use and what is the induction hypothesis). You can assume that the postconditions involve only the variable x .

Exercise 4: We obtained a recursive equation for the semantics of the `while` statement in IMP using the unwinding equation. Then we argued that the equation admits too many solutions and we made the informal observation that all of the solutions disagree only on inputs for which the `while` does not terminate. Now we’ll explore this issue formally.

Consider a flat domain D and F a continuous function ($F \in (D \rightarrow D) \rightarrow (D \rightarrow D)$). We’ll refer to the least-fixed point of F as $\text{LFP}(F)$. Recall that we defined the semantics of `while` to be just such a least fixed point for some function F . Prove or disprove both implications in the following statement for some arbitrary continuous function $f : (D \rightarrow D)$:

$$f = F(f) \quad \text{iff} \quad \forall x \in D. \text{LFP}(F)(x) = \perp \text{ or } f(x) = \text{LFP}(F)(x)$$

Exercise 5: Consider the following variant of simply-typed lambda calculus:

$$\begin{aligned} e &::= x \mid n \mid e_1 + e_2 \mid e_1 \geq e_2 \mid \lambda x : \tau. e \mid e_1 e_2 \\ \tau &::= \text{int} \mid \Pi x : \tau_1. \tau_2 \mid \{x : \tau \mid P\} \end{aligned}$$

The type $\{x : \tau \mid P\}$ denotes the subset of elements x of type τ that satisfy the predicate P . Here x is a bound variable whose scope is P .

For example, the type $\{x : \text{int} \mid x \geq 0\}$ represents non-negative integers.

1. Describe an appropriate subtyping judgment for this language. Try to allow as many subtypings as possible.

2. Design a decision procedure for subtyping (under appropriate restrictions for the language of predicates P).