Programming Languages

Fall 2003 Qualifiers

August 26, 2003

Question 1

Introduction

An interesting construct has been suggested to handle concurrent executions in the context of functional programming languages. The construct is called `future`. Intuitively, the `future` construct is used to indicate that all the arguments of a given function call should be evaluated simultaneously. For example, if we perform a function call of the type (this is my own syntax...):

$$\texttt{future:}f(\,g(1)\,,\,2+3\,,\,9\,)$$

then 3 threads are going to be spawned, one in charge of computing the value of `g(1)`, one in charge of computing the value of `2+3`, and one in charge of computing the value of `9`. Once the three threads have reached their conclusions, then the function `f` will be called.

Denotational Semantics

The abstract syntax of the language is described by the following grammar:

$$\begin{array}{rcl}
\langle \mathit{expression} \rangle & \rightarrow & \texttt{let}\ \ \langle \mathit{id} \rangle = \langle \mathit{expression} \rangle\ \ \texttt{in}\ \ \langle \mathit{expression} \rangle \\
& | & \texttt{lambda}\ \ (\langle \mathit{id} \rangle)\ \langle \mathit{expression} \rangle \\
& | & (\langle \mathit{expression} \rangle\ \ \langle \mathit{expression} \rangle) \\
& | & (\texttt{future:}\langle \mathit{expression} \rangle\ \ \langle \mathit{expression} \rangle) \\
& | & \langle \mathit{numeral} \rangle \\
& | & \langle \mathit{expression} \rangle\ +\ \langle \mathit{expression} \rangle \\
& | & \langle \mathit{id} \rangle \\
& | & \mathit{nil} \\
& | & \langle \mathit{expression} \rangle\ \ \texttt{cons}\ \ \langle \mathit{expression} \rangle \\
& | & \texttt{head}\ \ \langle \mathit{expression} \rangle \\
& | & \texttt{tail}\ \ \langle \mathit{expression} \rangle
\end{array}$$

Observe that, with the exception of the `future`, this is essentially the same language studied in class and discussed in chapter 7 of Schmidt's book.

The denotational semantics is essentially identical to the one described in Schmidt's book. In particular, the semantics for the `future` construct can be simply given as

$$\mathbf{E}[\texttt{future:}f(e_1,\ldots,e_n)] = \left(\begin{array}{l} \lambda s.cases\ (access\ s\ f)\ of \\ \quad isFunction(fun) \rightarrow fun(\mathbf{E}[e_1]s,\ldots,\mathbf{E}[e_n]s)\ \Box \\ \quad \ldots \end{array} \right)$$

Note that all the arguments are evaluated in parallel (on the same store).

Implementation [40pts]

Discuss how the `future` construct can be implemented in the context of a language as described in the previous question. Your implementation should rely on the translation of the language to a C program. In particular, assume that your implementation should rely on a standard thread library (e.g., pthreads), that provides mechanisms to start a thread (e.g., something like `pthread_create` that starts a new thread, which executes a specified function) executing a given function and wait for termination of a thread (e.g., something like `pthread_wait` which will wait for termination of the specified thread).

Extension [40pts]

Assume that we extend the language with the introduction of a new case of expression:

$$\texttt{setq}\quad \langle id \rangle\ \langle expression \rangle$$

This expression will return as result the value of the $\langle expression \rangle$ but at the same time it will modify the assignment of value to the identifier $\langle id \rangle$ (i.e., it essentially behaves in the same way as the statement

$$id = expression$$

in the C language, where the value of the expression is returned as result and the value of id is modified).

Discuss how the denotational semantics of this new language can be developed. Focus only on the difference w.r.t. the semantics you developed in the first question.

Axiomatic Semantics for parallel constructs [20pts]

Let us look at a parallel execution construct similar to future. Let us assume that we have a construct with the following syntax:

$$future(c_1,\ldots,c_k)$$

where $c_1,\ldots,c_k$ are commands in a standard imperative language. The construct will execute all the k commands concurrently. We would like to develop a new rule for Hoare partial correctness axiomatization (see pg. 66 of Gumb's book) to deal with this *future* construct. The rule should have the following format:

$$\frac{\ldots}{\{p\}future(c_1, \ldots, c_k)\{q\}}$$

Justify your answer.