CS350 HW3

Due: October 20, 2011

1. Semantics of Trigger Creation for Lazy Execution. Extend your program of Assignment 2 to support lazy execution. Support the following operation. [byneed [subr (p)] [ident (x)]

   You can assume that the program is sequential except for laziness, so you need not implement the semantic multistack or the thread scheduler. You need not program trigger activation semantics.

   [25 points]

2. Consider the nondeteministic choice operation described below.

   $$\{\text{Choose } [X_1\#S_1 \ X_2\#S_2 \ \ldots \ X_n\#S_n]\}$$

   Here, $X_1, \ldots, X_n$ are boolean-valued variables, and $S_1, \ldots, S_n$ are statements. You can assume that the list is non-empty.

   The operation should behave as follows. If $X^{(1)}, \ldots, X^{(k)}$ is the subsequence of variables bound to `true`, then nondeterministically select one of $S^{(1)}, \ldots, S^{(k)}$, and execute it. If every variable is bound to `false`, raise a "missingClause" exception. If no variable is bound, the statement blocks.

   If this operation is possible, give its translation in Oz syntax. If it is not possible, explain which limitation of Oz prevents us from implementing this. Assume facilities provided in the message-passing model of Oz.

   [10 points]

3. Section 5.7.3 gives the translation of Erlang's receive without timeouts, into Oz. Supplement the pattern matching mechanism described with guards. The receive operation should block until a pattern is matched. If a pattern is matched and the corresponding guard is satisfied, then the message is removed from the mailbox and the body executed. If no pattern is matched or some matched pattern fails the guard, the receive blocks waiting for a suitable message.

   Please consult section 6.19.9 of the Erlang specification, at `www.erlang.org/download/erl_spec47.ps.gz`

   [10 points]

4. Erlang Programming.

   (a) Implement a finite-state automaton for the regular expression $a(ba)^*b$ as a module. This module should export a public function `is_accepted/1` which takes a list of characters as argument, and returns true if the string is accepted by the finite automaton, and false if it is not accepted.

   For example,

```
is_accepted([a,b]) == true
is_accepted([a])   == false
```

   [15 points]

   (b) Let us denote a list of characters like `[a b]` as a string. Write an Erlang program which will take a list of strings, spawn one process per string, and return the list of results of the finite-state automata running on the strings.

   For example,

```
results([[a] [a,b]]) == [true,false].
```

   [10 points]

5. Write an erlang function that will discard all messages till the last message, and return the last message.

   [10 points]