COMPUTER SCIENCE 322 (Winter Term 2004)
Compiler Construction
Prof. Levy

Problem Set 7

Due Friday 19 March

Reading Assignment: Dragon Book 6.1-6.3,6.5-6.7

Programming Assignment

Here is a grammar for a richer version of our TinyML language.

expr → *let-expr* | *cond-expr* | *comp-expr* | *arith-expr* | *bool-expr* | *app-expr* | **(** *expr* **)** | **id** | **number** | **boolean**
let-expr → **let** *bindings* **in** *expr* **end**
cond-expr → **if** *expr* **then** *expr* **else** *expr*
comp-expr → *expr* > *expr* | *expr* < *expr*
arith-expr → *expr* + *expr* | *expr* * *expr*
bool-expr → *expr* **and** *expr* | *expr* **or** *expr*
app-expr → *expr* **(** *expr* **)**
bindings → *binding* | *binding* **;** *bindings*
binding → **val** *decl* = *expr*
binding → **fun** *decl* **(** *decls* **)** = *expr*
decls → *decl* | *decl* **,** *decls*
decl → **id :** *type* | **id**
type → **int** | **bool**

Copy your `TinyML.flex` and `TinyML.cup` files from the last assignment, and add the necessary lexical and grammar rules to parse this language. You'll also need to add precedence rules for the greater-than and less-than symbols and the left-paren symbol to avoid shift/reduce conflicts (I'll let you figure out what the relative precedence for these should be.) Copy your best symbol-table implementation from the last assignment into `SymbolTable.java`, and make sure to rename the class declaration and remove the `implements SymbolTable`. When you have the parser working, add parse actions to implement type checking as in (the notes for) Chapter 6. We don't have polymorphism in our language, but it should still be easiest to use the type-checking and unification algorithms from the chapter, rather than coming up with something on your own. Start small (at the bottom of the parse tree), and work your way up to the full type-checker for all expressions.

For the time being, we'll focus on type-checking, and not worry about recursively defined environments; *i.e.*, your type-checker should work with programs like

```
let fun f(a) = a + 1 in f(3) end
```

but doesn't have to work with programs like

```
let fun f(a) = a + 1 in  let fun g(b) = b + 2 in f(g(4)) end end
```

As the class notes suggest, your type-checking (and hence symbol-table access) should be done in the parser, and no longer in the lexical analyzer. So you should remove your symbol-table code from `TinyML.flex` after you copy it. Then you can add the following code to your `TinyML.cup`, right after the `import` statement:

```
parser code {:

  public SymbolTable table;

  public parser(java_cup.runtime.Scanner s, SymbolTable t) {
    super(s);
    this.table = t;
  }

:}
```

Now in your semantic rules, you can access the symbol table as `this.parser.table`. See the CUP documentation for how to annotate your grammar to access identifier names, and other useful values, in your rules.

You have some flexibility as to how you organize your type-checking code, but I suggest using a class hierarchy of type-tree nodes: `AppNode`, `TupleNode`, `LeafNode`, etc. To support unification, your abstract `TypeNode` should have a `next` pointer to another `TypeNode`. However your solution works, it should report a type error explicitly, rather than simply crashing with a null-pointer exception or something equally uninformative.

Turn in your `TinyML.flex`, `TinyML.cup`, `SymbolTable.java`, and any other Java code you write to complete the assignment. If you implement type-checking in a different way from what I've described, please also submit a brief README file with your submission, explaining how your solution works.