Compiler Design Description Assignment 1

Richard Delaney

Nov 13

Definition of Problem

In the Assignment we were given, immediately many different key parts of the assignment jumped out, I will try to detail through this design description how I approached and carried out these different sections of the project.

In my designing of an attributed grammer, I very much started at the base level of functionality required and worked my way up to a fully functional command-line driven interpreter as described. The first task I tackled was the basic arithmetic, In lectures we had already seen the basic design for an E T Grammer which can carry out the necessary arithmetic expressions.

The productions of the grammer are simply:

1. $\langle E \rangle \rightarrow \langle E \rangle + \langle T \rangle$
2. $\langle E \rangle \rightarrow \langle T \rangle$
3. $\langle T \rangle \rightarrow \langle T \rangle * \langle P \rangle$
4. $\langle T \rangle \rightarrow \langle P \rangle$
5. $\langle P \rangle \rightarrow (\langle E \rangle)$
6. $\langle P \rangle \rightarrow Const$

I defined this grammer as best I could in my .atg file and compiled it using Coco/R which generated a scanner and parser which matched the grammer. After getting basic arithmetic operations successfully functioning, I moved onto the next step of finding a means of storing expression values in a identifier of some type.

From our studies so far, We've seen that a Symbol-Table is whats required in this case, I had a look at the Test suite provided by coco/r and decided their take on a symbol table was overly complex for the project, I looked at the problem domain and summarised that the only type we needed was int. I chose to go with the mod operation rather than division and keep all numbers whole. This meant that instead of needing a symbol table that kept track of the type of a certain variable, all we needed was some way to retrieve the integer value of a variable given its name. I decided using a hash map was the best way in

which to acheive this. I went with the STL libraries and a hash map with strings as keys and ints as values. This made variable instantiation and retrieval very easy, by searching the hash table on the name of the variable supplied.

The grammer which dealt with assignment was as follows:

VarDecl = Ident ":=" Expr;

The next part was combining the two so that expressions themselves could contain variables, I acheived this by including Ident in the bottom production, i.e Factor in my .atg or $\langle P \rangle in the \langle E \rangle \langle T \rangle$ grammar. This production can be seen in my atg file.

The last part I had to deal with in terms of the grammer was that of displaying the variables in the 3 different formats, hexidecimal, octal and decimal.

As I've already said, the symbol table made it easy to retrieve the values of the variables and to display it I used the helpful printf which displays in octal,hexidecimal and decimal depending on which format you provide.

The only remaining piece was making it into a command-line driven interpreter, this was somewhat of a stumbling block and took some work before I got it to a satisfactory user position, First I tried to feed the scanner simply using the stdin stream. This worked only for two statements. Instead, I decided to go with a loop of entry, with a simple fgets which is then fed into the scanner and parsed, the trick here is that the pointer to the symbol table in parser is always pointing to the same symbol table allowing the variables to have a continuing life from statement to statement.