Birla Institute of Technology and Science, Pilani

K K Birla Goa Campus

CS F363 Compiler Construction (Special) Comprehensive Examination, Regular

24 May 2021, 2 hours, 70 marks

This is based on the SDT scheme we have been using throughout for a vector calculator. For advanced error handling, we read [Aho et al., 2006, Section 4.8.3] and some related ideas in handling ambiguities in the larger Section 4.8 therein.

Panic Mode Recovery The recipe given in that section cite above:

1. Scan down the stack until a state S with a goto on some nonterminal A is found.
2. Discard zero or more input symbols until a symbol `a` is found that can legitimately follow A.
3. Stack the state $GOTO(S, A)$ and resume normal parsing.
4. If more than one nonterminal A can be candidates for this, break the tie by choosing the one that needs least shedding.

This achieves parsing at the cost of some "phrase" or sub-expression.

Compre Exam Task

Make a suitable (or adapt the already given) grammar for the vector calculator, implement the parser-interpreter. Then go on to implement the error recovery scheme as depicted in the previous section.

The Language

$$
\begin{aligned}
L &: L\ L_1 && (1)\\
&\mid \epsilon && (2)\\
L_1 &: E\ '\backslash n' && (3)\\
&\mid '\backslash n' && (4)\\
E &: \texttt{VAR}\ '='\ E && (5)\\
&\mid E\ '+'\ T && (6)\\
&\mid T && (7)\\
T &: T\ '*'\ F && (8)\\
&\mid F && (9)\\
F &: \texttt{NUM} && (10)\\
&\mid '('\ E\ ')' && (11)\\
&\mid '['\ L_t\ ']' && (12)\\
&\mid F\ '\backslash''' && (13)\\
L_t &: E\ L_t && (14)\\
&\mid E && (15)
\end{aligned}
$$

Here rule no. (13) is the transpose: the escaping of the single quote within a pair of single quotes makes it the single quote character token.

To understand the transpose, e.g., this is how `GNU/octave` handles it:

```
octave:1> a=[1 2 3 4]
a =

1   2   3   4

octave:2> a'
ans =

1
2
3
4

octave:3> b=a'
b =

1
2
3
4

octave:4> c=[b b b b]
c =

1   1   1   1
2   2   2   2
3   3   3   3
```

```
4   4   4   4

octave:5> d=[b 2*b 3*b 4*b]
d =

1    2    3    4
2    4    6    8
3    6    9   12
4    8   12   16
```

Remember, we are neither emulating `GNU/octave` nor are we trying to be compatible with its grammar. Only the transpose idea is taken from there. The cross-product of vectors and matrices is not to be implemented. Our product is the elementwise list product.

Exam Questions

Q1 In the bottom-up parser construction the assignment rule, rule no. (5) as

$$E \ : \ \texttt{VAR}\,'='\ E$$

is something special: it can appear as a statement in itself but it can also be a subexpression.

(a) Does it lead to any conflict? If your answer is yes, specify the conflict, the state (the set of items with their dots therein) wherein it occurs. If your answer is no, then explain why it changes nothing. The answer is short, your effort may be long. **[Marks 20]**
(*Hint:* `bison -r` *gives you many reports including states, items, conflicts, resolution etc.*)

(b) If we replace E on the right hand side by F, in the same rule, what happens? The answer for this part is also to be written like part (a): explain the conflict if you expect it there, or explain why there will not be a conflict. **[Marks 10]**

(b) Again, if we replace E on the right hand side by T instead, in the same rule, what happens? Answer in the same vein. **[Marks 5]**

Q2. Feed the input given at the bottom through the calculator that you have developed and give the output. Feed the file as a redirected `stdin`. If your calculator lacks some functionality as sought, it will give error messages, no problem. **[Marks 20]**

Q3. Explain the error messages that happened during the above run, and explain if you have implemented any error recovery scheme, or anything that can be implemented (don't repeat what is there in the textbook or in the material on `quanta`, give a concrete scheme). Focus on the semantic dilemmas – e.g. a variable is the assignmnent destination and also appears on the right hand side as a subexpression. Does the `$$,$1,$2...` stack mechanism in `bison` work seamlessly here? Try different tweaks there and give those examples. *Do not write an essay.* **[Marks 15]**

Altrnatively, you may give your scheme of error recovery that was the homework assignment given in the last week of April. Or, alternatively: you may give a scheme how to change the grammar and the interpreter such that expressions can span multiple lines and are evaluated when they are complete; this also means you need to allow multiple whole expressions separated by some delimiter on the same line. Several values will appear in the output in that sequence, accordingly.

References

[Aho et al., 2006] Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA.

The Input File

```
1+2
(1+2)
1+2*3
(1+2)*3
[1+2]
[1 2]
[1 + 2]
[(1+2)]
[(1 + 2)]
[(1 + 2)3]
[(1 + 2)*3]
[(1 + 2) *3]
a=[(1 + 2) *3]
b = [(1 + 2) *3]
a=4
a=[ 1 2 3 ]
b = a'
[b b b]
a = [a a a]
a = [1 2 3]
a = [a' a' a']
a = [1 2 3]
a = [a' a' a']'
```

END.