Problem Set 2: Practical Parsing and Code Generation for SaM Solution Key

C S 395T

08 September 2021

This is the problem set for week 2 of C S 395T SIMPL. It is intended to help you learn the material by working out more examples and exercises than is possible to cover in the videos. Feel free to work individually or in groups. Ask questions on Piazza. You are not required to submit anything, and the problem set doesn't count directly towards your course grade. The solution key for the problem set will be made available one week after its release.

Several of the questions in this problem set refer to the following input character strings: I_1 = `((32+100)*(16-4))`, I_2 = `((32+(10*10))*(16-4))`, and I_3 = `((32+(100*16))-4)`.

1. *Input character strings and token streams.*
   For each input character string I_k $(1 \leq k \leq 3)$, write down the corresponding token stream S_k that `SamTokenizer` would generate. What are the compression ratios for each input string, where *compression ratio* is defined as the number of characters in the input string divided by the number of resulting tokens?

   Solution:

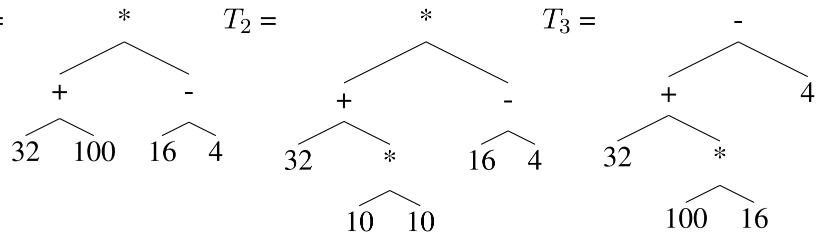
$$S_1 = [\boxed{\texttt{(}}\boxed{\texttt{(}}\boxed{\texttt{32}}\boxed{\texttt{+}}\boxed{\texttt{100}}\boxed{\texttt{)}}\boxed{\texttt{*}}\boxed{\texttt{(}}\boxed{\texttt{16}}\boxed{\texttt{-}}\boxed{\texttt{4}}\boxed{\texttt{)}}\boxed{\texttt{)}}];\ \text{CR} = \tfrac{17}{13}.$$

$$S_2 = [\boxed{\texttt{(}}\boxed{\texttt{(}}\boxed{\texttt{32}}\boxed{\texttt{+}}\boxed{\texttt{(}}\boxed{\texttt{10}}\boxed{\texttt{*}}\boxed{\texttt{10}}\boxed{\texttt{)}}\boxed{\texttt{)}}\boxed{\texttt{*}}\boxed{\texttt{(}}\boxed{\texttt{16}}\boxed{\texttt{-}}\boxed{\texttt{4}}\boxed{\texttt{)}}\boxed{\texttt{)}}];\ \text{CR} = \tfrac{20}{17}.$$

$$S_3 = [\boxed{\texttt{(}}\boxed{\texttt{(}}\boxed{\texttt{32}}\boxed{\texttt{+}}\boxed{\texttt{(}}\boxed{\texttt{100}}\boxed{\texttt{*}}\boxed{\texttt{16}}\boxed{\texttt{)}}\boxed{\texttt{)}}\boxed{\texttt{-}}\boxed{\texttt{4}}\boxed{\texttt{)}};\ \text{CR} = \tfrac{17}{13}.$$

2. *Abstract syntax trees.*
   For each input character string I_k $(1 \leq k \leq 3)$, show its corresponding abstract syntax tree T_k when parsed with the expression grammar of LiveOak. In what way are T_1 and T_2 the same? In what way are they different?

   Solution: T_1 = T_2 = T_3 =



   T_1 and T_2 are the same in that they evaluate to the same value. They are structurally different.

3. *Parsing and lookahead.*
   Trace the individual steps of parsing and AST generation for input string I_1 using a recursive-descent parser driven by the

expression grammar of LiveOak. At each step, indicate the current and lookahead tokens, and sketch the shape of the parse tree and AST.

Solution: See Table 1.

4. *Code generation.*
Show the SaM code generated for input string I_2. Indicate the correspondence between code fragments and the AST nodes where they are generated and/or assembled.

   Solution: The SaM code is as follows.

```
PUSHIMM 32
PUSHIMM 10
PUSHIMM 10
MUL
ADD
PUSHIMM 16
PUSHIMM 4
SUB
MUL
```

   The rest of the problem is left as an exercise for the student.

5. *Code shape.*
Consider the following piece of LiveOak-2 code.

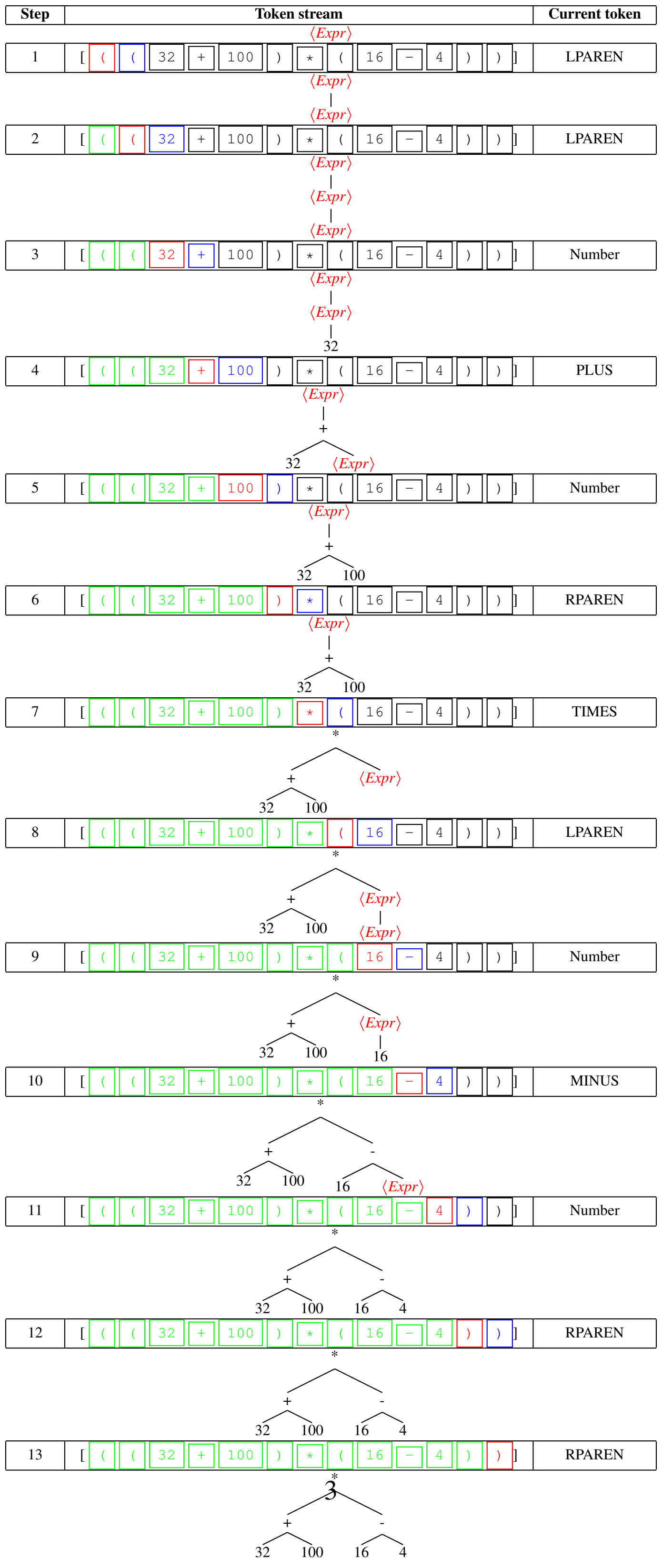
```
int main() {
        int a, b, c;

        a = 431; b = 123; c = 345;
        while((b<a)) {
                while((b<c))
                        b = (b+1);
                if((b = 345))
                        break;
                else
                        b = (b + 1);
        }
        return b;
}
```

   Work through the actions of the recursive-descent parser and the SaM code generator on this input. Show the structure of the AST and the contents of the symbol table. Indicate how the various code fragments are generated, stitched together, and passed around in the tree representation of the program.

   Solution: Left as an exercise for the student.

Table 1: Step-by-step snapshots of AST construction for input string I_1. The current token is in red, the lookahead token in blue, and processed tokens in green. The AST shown above a step is in its state before consuming the current token, while the AST shown below is in its state after consuming the current token.



Step	Token stream	Current token
1	[((32 + 100) * (16 − 4))]	LPAREN
2	[((32 + 100) * (16 − 4))]	LPAREN
3	[((32 + 100) * (16 − 4))]	Number
4	[((32 + 100) * (16 − 4))]	PLUS
5	[((32 + 100) * (16 − 4))]	Number
6	[((32 + 100) * (16 − 4))]	RPAREN
7	[((32 + 100) * (16 − 4))]	TIMES
8	[((32 + 100) * (16 − 4))]	LPAREN
9	[((32 + 100) * (16 − 4))]	Number
10	[((32 + 100) * (16 − 4))]	MINUS
11	[((32 + 100) * (16 − 4))]	Number
12	[((32 + 100) * (16 − 4))]	RPAREN
13	[((32 + 100) * (16 − 4))]	RPAREN