Parser Parser Parser

Zach Homans
Kyle McConnaughay

Problem Statement

- Writing Grammars is Simple
- Writing Parsers is Difficult and Boring
  - Tons of Reused Code
  - Multiple Components
  - Mindnumbing
  - Strange Indentation

```
and parse_cterm ts =
    (case parse_term ts
of NONE => NONE
 | SOME (e1,ts) =>
   (case expect_PLUS ts
     of NONE =>
  (case expect_MINUS ts
    of NONE => SOME (e1,ts)
     | SOME ts =>
       (case parse_term ts
         of NONE => NONE
    | SOME (e2,ts) => SOME (call2 "sub" e1 e2, ts)))
      | SOME ts =>
  (case parse_term ts
    of NONE => NONE
     | SOME (e2,ts) => SOME (call2 "add" e1 e2, ts))))
```

Proposed Solution

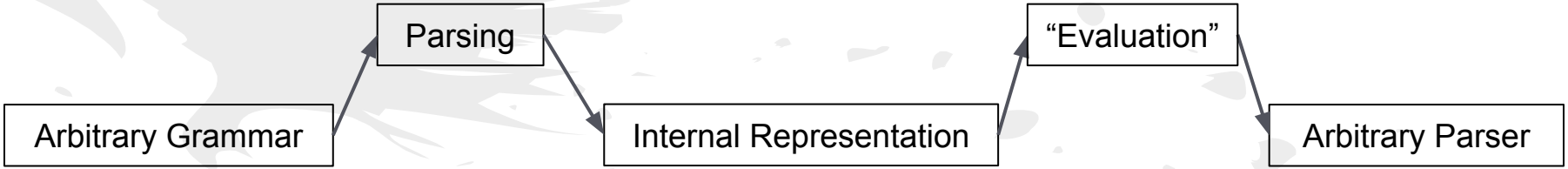
Create a mini-language that takes a grammar and produces a corresponding parser.

```
(T_WHITESPACE, "( |\\n|\\t)+", NONE);
(T_BAR, "\\|", UNIT);
(T_COMMA, ",", UNIT);
(T_EQUAL, "::=", UNIT);
(T_LPAREN, "\\(", UNIT);
(T_RPAREN, "\\)", UNIT);
(T_SEMICOLON, ";", UNIT);
(T_SQUIGGLE, "~", UNIT);
(T_STR, "\"([^\\\\\"]|\\\\.)*\"", STRING);
(T_SYM, "[a-zA-Z][a-zA-Z0-9_]*", SYM)

~

(file ::= (term_list T_SQUIGGLE nonterm_list, "I.EFile (output1, output3)"));
(terminal ::= (T_LPAREN T_SYM T_COMMA T_STR T_COMMA T_SYM T_RPAREN, "I.ETerm (
output2, output4, output6)"));
(term_list ::= (terminal T_SEMICOLON term_list, "output1::output3")
             | (terminal, "[output1]"));
(token_list ::= (T_SYM token_list, "output1::output2")
              | (T_SYM, "[output1]"));
(rule ::= (T_LPAREN token_list T_COMMA T_STR T_RPAREN, "I.EMap (output2,
output4)"));
(rule_list ::= (rule T_BAR rule_list, "output1::output3")
             | (rule, "[output1]"));
(nonterminal ::= (T_LPAREN T_SYM T_EQUAL rule_list T_RPAREN, "I.ENon (output2,
output4)"));
(nonterm_list ::= (nonterminal T_SEMICOLON nonterm_list, "output1::output3")
                | (nonterminal, "[output1]"))
```

Structure of the Solution



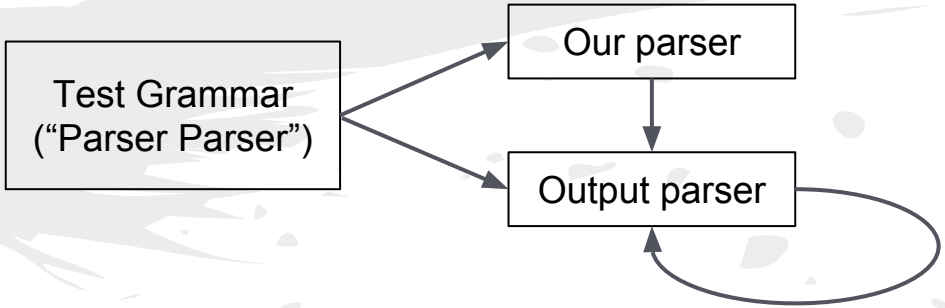
Test Grammar

```
(T_WHITESPACE, "( |\\n|\\t)+", NONE);
(T_BAR, "\\|", UNIT);
(T_COMMA, ",", UNIT);
(T_EQUAL, "::=", UNIT);
(T_LPAREN, "\\(", UNIT);
(T_RPAREN, "\\)", UNIT);
(T_SEMICOLON, ";", UNIT);
(T_SQUIGGLE, "~", UNIT);
(T_STR, "\"([^\\\\\"]|\\\\.)*\"", STRING);
(T_SYM, "[a-zA-Z][a-zA-Z0-9_]*", SYM)

~

(file ::= (term_list T_SQUIGGLE nonterm_list, "I.EFile (output1, output3)"));
(terminal ::= (T_LPAREN T_SYM T_COMMA T_STR T_COMMA T_SYM T_RPAREN, "I.ETerm (
output2, output4, output6)"));
(term_list ::= (terminal T_SEMICOLON term_list, "output1::output3")
             | (terminal, "[output1]"));
(token_list ::= (T_SYM token_list, "output1::output2")
              | (T_SYM, "[output1]"));
(rule ::= (T_LPAREN token_list T_COMMA T_STR T_RPAREN, "I.EMap (output2,
output4)"));
(rule_list ::= (rule T_BAR rule_list, "output1::output3")
             | (rule, "[output1]"));
(nonterminal ::= (T_LPAREN T_SYM T_EQUAL rule_list T_RPAREN, "I.ENon (output2,
output4)"));
(nonterm_list ::= (nonterminal T_SEMICOLON nonterm_list, "output1::output3")
                | (nonterminal, "[output1]"))
```

Parsers All The Way Down



Yo dawg, I heard you like parsers.

So we made a parser that parses a parser parser and makes a parser that can parse and produce more parser parsers.

Sample Result

```
fun parse_T_SQUIGGLE ((T_SQUIGGLE)::ts) = SOME ((), ts)
    | parse_T_SQUIGGLE _ = NONE

fun parse_T_STR ((T_STR t)::ts) = SOME (t, ts)
    | parse_T_STR _ = NONE

fun parse_T_SYM ((T_SYM t)::ts) = SOME (t, ts)
    | parse_T_SYM _ = NONE

and parse_nonterm_list ts = let
    fun map2 ts =
        (case parse_nonterminal ts of NONE => NONE
            | SOME (output1, ts) =>
        (case parse_T_SEMICOLON ts of NONE => NONE
            | SOME (output2, ts) =>
        (case parse_nonterm_list ts of NONE => NONE
            | SOME (output3, ts) => SOME (output1::output3, ts))))
    fun map1 ts =
        (case parse_nonterminal ts of NONE => NONE
            | SOME (output1, ts) => SOME ([output1], ts))
in
    choose [map2,map1] ts
end

and parse_nonterminal ts = let
    fun map1 ts =
        (case parse_T_LPAREN ts of NONE => NONE
            | SOME (output1, ts) =>
        (case parse_T_SYM ts of NONE => NONE
            | SOME (output2, ts) =>
        (case parse_T_EQUAL ts of NONE => NONE
            | SOME (output3, ts) =>
        (case parse_rule_list ts of NONE => NONE
            | SOME (output4, ts) =>
        (case parse_T_RPAREN ts of NONE => NONE
            | SOME (output5, ts) => SOME (I.ENon (output2, output4), ts))))
              ))
in
    choose [map1] ts
end
```

Interesting Challenges

- Regular expressions for strings
  - \"([^\\\\\"]|\\\\.)*\"
- Matching variables associated with tokens with the internal representation included in the parser
  - expect_SYM vs. expect_SEMICOLON
  - “output1”, “output2”, etc.

Questions?