Compilation 2013

Parser Generators, Conflict Management, and ML-Yacc

Erik Ernst

Aarhus University

Parser generators, ML-Yacc

- LR parsers are tedious to write, but can be generated, e.g., by ML-Yacc
- Input: A parser specification
  - Grammar rules (context-free, typically LALR(1))
  - Directives (manage types, conflicts, ..)
  - User code, supporting semantic actions
- Output: Source code of a parser, ready to use
- Hurdle: Conflict management!

Example Grammar

- Grammar:

P	→	***L***	***L***	→	***S***
S	→	**id := id**	***L***	→	***L ; S***
S	→	**while id do *S***			
S	→	**begin *L* end**			
S	→	**if id then *S***			
S	→	**if id then *S* else *S***			

- Expressing the same thing for ML-Yacc
  - Rule format: `stm : WHILE ID DO stm ()`
  - Sections:

```
user declarations
%%
parser declarations
%%
grammar rules
```

Example Grammar Spec

- ML-Yacc specification:

```
%%
%term  ID | WHILE | BEGIN | END | DO | IF | THEN | ELSE | SEMI | ASSIGN | EOF
%nonterm  prog | stm | stmlist
%pos int
%verbose
%start prog
%eop EOF %noshift EOF
%%
prog : stmlist                                  ()
stm : ID ASSIGN ID                              ()
    | WHILE ID DO stm                           ()
    | BEGIN stmlist END                         ()
    | IF ID THEN stm                            ()
    | IF ID THEN stm ELSE stm                   ()
stmlist : stm                                   ()
        | stmlist SEMI stm                      ()
```

Example Grammar Spec

- 'Verbose' ensures feedback, esp. on conflicts:

```
1 shift/reduce conflict

error:  state 17: shift/reduce conflict
(shift ELSE, reduce by rule 4)

state 0:

        prog : . stmlist

        ID      shift 6
        WHILE   shift 5
        BEGIN   shift 4
        IF      shift 3

        prog    goto 21
        stm     goto 2
        stmlist goto 1

        .       error
...
```

```
state 17:

        stm : IF ID THEN stm .  (reduce by rule 4)
        stm : IF ID THEN stm . ELSE stm

        ELSE    shift 19
        .       reduce by rule 4

...

state 21:

        EOF     accept
        .       error

15 of 57 action table entries left after
compaction
9 goto table entries
```

Revisit Ambiguous Grammar

- Grammar:

E	→	**id**	***E***	→	***E*** **+** ***E***
E	→	**num**	***E***	→	***E*** **–** ***E***
E	→	***E*** ***** ***E***	***E***	→	**(** ***E*** **)**
E	→	***E*** **/** ***E***			

- Much worse: 16 shift/reduce conflicts!
- Last time: Rewritten *(E,T,F)*
- Can be managed using conflict resolution hints
- Default: Prefer shift for sh/rd, first rule for rd/rd

Wishes for Revisited Grammar

- Grammar:

E	→	id	*E*	→	*E* + *E*
E	→	num	*E*	→	*E* – *E*
E	→	*E* * *E*	*E*	→	(*E*)
E	→	*E* / *E*			

- Multiplication, division *binds stronger than* addition, subtraction (precedence)
- Operators left associative
- May also specify no associativity

Shift-reduce and Precedence

- A partial LR(1) state:

E	$\rightarrow$	$E * E \bullet$	$+$
E	$\rightarrow$	$E \bullet + E$	*(any)*

- If we prefer shift: Stack is $E * E$ becomes $E * E +$ and later $E * E + E$ which must reduce the addition

- Must prefer reduce for right precedence: $E * E$ becomes E and later $E + E$ which has reduced the multiplication

Shift-reduce and Associativity

- A partial LR(1) state:

E	→	***E + E •***	**+**
E	→	***E • + E***	***(any)***

- If we prefer shift: Stack is ***E + E*** becomes ***E + E +*** and later ***E + E + E*** reducing the rightmost addition

- Must prefer reduce for right precedence: ***E + E*** becomes ***E*** and later ***E + E*** which has reduced the leftmost addition first

Non-Associativity

- A partial LR(1) state:

E	$\rightarrow$	$E - E \bullet$	$-$
E	$\rightarrow$	$E \bullet - E$	*(any)*

- Make that position in the parsing table an error

- Point `2-3-4` is inherently error-prone, just outlaw it! Similarly for `i == j == k`

Precedence & Associativity Control

- Related specification declarations:

```
%%
%term  INT | PLUS | MINUS | TIMUS | UMINUS | EXP | EQ | NEQ | EOF
%nonterm  exp
%start  exp
%eop EOF
%nonassoc EQ NEQ
%left PLUS MINUS
%left TIMES
%right EXP
%left UMINUS                                  never returned from lexer!
%%
stm : ...
    | MINUS exp  %prec UMINUS      ()
```

- Use `%prec` to force precedence on rule

Error Handling

- Starting point: Do not just report one error and stop, keep running and report many
- Local strategy: Correct at location that made parser engine enter error state
- Global strategy: Perform minimal change that makes program syntactically correct
- Actions: Change parser stack, change input, change “inverse lookahead”

Error Handling Using ‘error’ Symbol

- A local strategy, known from YACC
- Idea: Grammar rules contain ‘error’ tokens, e.g.

E	→	**id**	***L***	→	***L*** **;** ***E***
E	→	***E*** **+** ***E***			
E	→	**(** ***E*** **)**	***E***	→	**(error)**
L	→	***E***	***E***	→	**error ;** ***E***

- Generator: ‘error’ terminal, generate usual shift
- Engine: When error, pop until action for ‘error’ is shift; shift it; discard input until non-error action; resume parsing

Burke-Fisher Error Handling

- A limited global strategy
  - Choose *K*, create *K*-place buffer
  - Maintain two parse stacks, *K* steps apart
  - Semantic actions: only "delayed" stack (side-effects!)
  - On error: Try all possible deletions/insertions/changes on the buffer
  - Complexity not bad: *N* tokens => (1+2*N*)*K*
- Advantage: Grammar, table unchanged, only engine algorithm changed
- Need default values (ID, INT)
- May use `%change` directives: >1 token

Summary

- LR parsing nice, implementation tedious
- Use generated code! E.g., ML-Yacc
- Specification file (3 sections, %%)
- Conflicts reported – shift/reduce, reduce/reduce
- Control parser table: precedence, associativity
- Error handling: local, global
- Using ‘error’ token
- Using Burke-Fisher “try all edits” approach