Better Parsing Through Lexical Conflict Resolution

Nathan Keynes

November 2, 2007

Contents

- The problem - Lexical conflicts
- Assorted solutions
- Automatic resolution
- Results

Assumed background:

- Basic compiler theory (cs324 or equivalent)

Conventional Parsing Model

- Separate lexical & syntactical modules
- Information loss due to separation
- “False” lexical conflicts

Conflict Example

Grammar:

```
subrange : '[' integer '..' integer ']' ;
integer : ``[0-9]+'' ;
float : ``[0-9]+\.[0-9]*''
```

Example Derived Lex Specification:

```
\[              { return LBRACKET; }
\]              { return RBRACKET; }
``..''          { return RANGE; }
[0-9]+          { return INTEGER; }
[0-9]+\.[0-9]*  { return FLOAT; }
```

Longest-match Conflict

Input: [12..30]

Tokenize: [12. . 30]

Result: LBRACKET, FLOAT, ERROR, INTEGER RBRACKET

This cannot now be parsed with the given grammar.

Identity Conflict

```
line: 'BEGIN' ':' ``[A-Za-z]+'' ;
```

Derived Lex Specification:

```
 BEGIN       { return BEGIN; }
 [A-Za-z]+   { return IDENT; }
 ':'         { return COLON; }
```

How do you scan 'BEGIN:BEGIN'?

Possible Solutions

- Change the language
- User specified context
- Scannerless parsing
- Automatically generated context

Why have languages like this?

- Greater flexibility
- Easier for end-users
- Why not?

User specified context

- messy
- unnecessarily complex
- hard to maintain

```
\[                { BEGIN(SR); return LBRACKET; }
\]                { return RBRACKET; }
``..''            { return RANGE; }
<SR,0>[0-9]+      { BEGIN(0); return INTEGER; }
[0-9]+\.[0-9]*    { return FLOAT; }
```

Scannerless parsing

- NSLR(1) Scannerless (Salomon & Cormack)[SC89]
- GLR Scannerless (Visser)[Vis97]

Pros/Cons:

- clean and easy to use
- non-canonical parse in most cases
- performance problems

Automatic Context

- partly proposed by Nawrocki[Naw91]
- easy to use
- canonical parse
- increased generator/parser complexity
- less powerful than scannerless methods?

Resolution Process

- Compute Lexical DFA (retaining conflicts) and LALR(2) parser DPDA
- For each DFA & LALR state combination, check if the conflict can be resolved for that state
- Record the correct action in a conflict table
- Produce a "minimal" copy of the DFA for each parser state
- Minimize all DFA copies together using standard algorithm

Identity Conflicts

For each DFA state, parser state, need two sets:

- NOW = all tokens accepted in the DFA state
- ACCEPTS = all tokens accepted in the parser state

Resolution = $NOW \cap ACCEPTS$

Longest Match Conflicts

For each DFA edge need the following sets:

- NOW = all tokens accepted in the source state
- NEXT = all tokens accepted in any successor state
- ALT = all tokens accepted in any successor of the start state following a transition on the same symbol.

For each parser state need the following sets:

- ACCEPTS = all tokens accepted (shift or valid lookahead)
- FOLLOWS = all tokens accepted after shifting a member of NOW

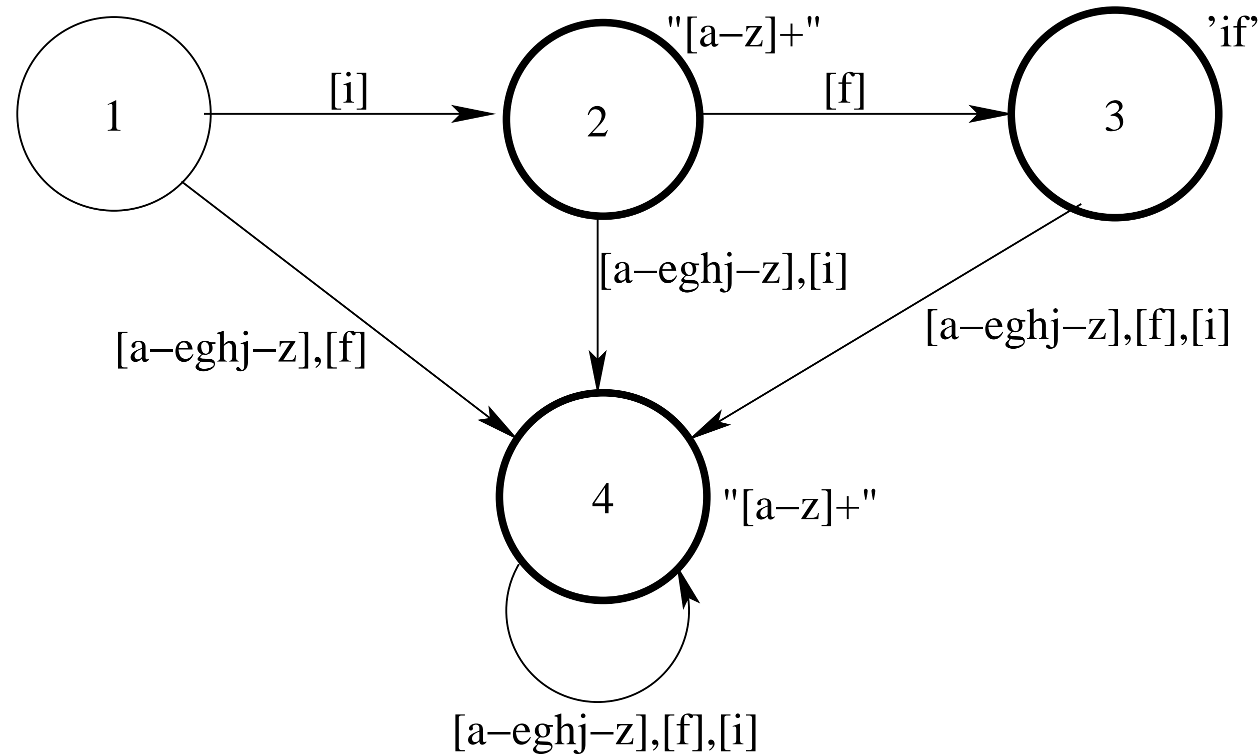

Figure 1: Combined DFA for 'if' and "[a-z]+"

For transition (3,[i]): NOW = {'if'},
NEXT = {"[a-z]+"}, ALT = {'if',"[a-z]+"}.

Then the resolution can be given by:

- if NOW is disjoint from ACCEPTS, then SHIFT (accepting now is guaranteed to produce an immediate syntax error)
- else if NEXT is disjoint from ACCEPTS, then ACCEPT (any future acceptable symbol from here will produce an error)
- else if ALT is disjoint from FOLLOWS, then SHIFT (accepting results in the next token producing a guaranteed error)
- else DONTCARE (unresolvable conflict - either decision is locally viable)

Results

- Keyd configuration file (unix daemon)
- ISO standard Pascal
- Parser runtime performance

Performance timings were obtained from a Pentium III/500Mhz, with no compiler optimizations unless otherwise stated.

Keyd

For a smallish configuration-type language (keyd), with 46 terminals, 22 nonterminals, and 69 productions:

- 152 of 156 DFA states have identity conflicts, 118 (77.6%) resolved
- 5236 DFA edges have longest match conflicts, 2707 (51.7%) resolved

Generation time was 2.4 seconds. Final DFA has 24 start states and 369 total states.

ISO Pascal

For ISO standard Pascal, 67 terminals, 135 nonterminals, 254 productions:

- 39 of 187 DFA states have identity conflicts, 11 (28%) resolved
- 3663 DFA edges have longest match conflicts, 1537 (42%) resolved

Generation time was 15 seconds. Final DFA has 61 start states and 393 total states.

Note that this is not technically a standards-conforming Pascal parser as currently specified.

Performance

A 4000 line, 117Kb Pascal file was used to compare the runtime performance of the generated parser against the equivalent bison/flex parser.

- test system, no compilers opts: 0.076 seconds
- bison/flex, no compiler opts: 0.048 seconds
- test system, -O2 compiler opts: 0.047 seconds
- bison/flex, -O2 compiler opts: 0.040 seconds

With compiler optimizations, the test system is just outside 15% of the bison/flex version - with no automata optimizations.

Further Work

- Automatic right context
- Exclusion or similar rules
- Generator optimizations

References

[Naw91] Jerzy R. Nawrocki. Conflict detection and resolution in a lexical analyzer generator. *Information Processing Letters*, 38(6):323–329, June 1991.

[SC89] Daniel J. Salomon and Gordon V. Cormack. Scannerless nslr(1) parsing of programming languages. *SIGPLAN Notices*, 24(7):170–178, July 1989.

[Vis97] Eelco Visser. Scannerless generalized-lr parsing. Technical Report P9707, University of Amsterdam Programming Research Group,

August 1997.