Elkhound: A Fast, Practical GLR Parser Generator

Scott McPeak
smcpeak@cs.berkeley.edu
9/16/02 OSQ Lunch

So what's wrong with Bison?

- LALR(1) is the problem
  - Restrictive subset of context-free grammars
  - Grammar hacking breaks conceptual structure
  - Can't resolve conflicts automatically if actions are present
  - LR is not closed under composition (union)
  - Fixing LR conflicts is hard: time, expertise

Ambiguous Grammars

- Use of ambiguity can simplify grammar
  - e.g. "$E \rightarrow E + E$", plus a rule for associativity
- Ambiguity can delay hard choices
  - Type/variable name ambiguity in C: "(a) & (b)"
  - C++: constructors, function-style casts, etc.
  - Other hard languages: Javascript, Perl
  - Natural languages?

Generalized LR (GLR)

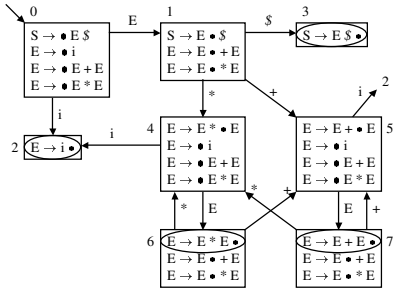
- Developed in 80's; natural language parsing
- Conceptually simple
- Uses *any* context-free grammar
- Ambiguous grammars → parse *forest*
- Efficient: same as LR in best case
- Worst case: $O(2^n)$
- Earley (1970) best is $\Theta(n^2)$, worst is $O(n^3)$

Review: LR Parsing

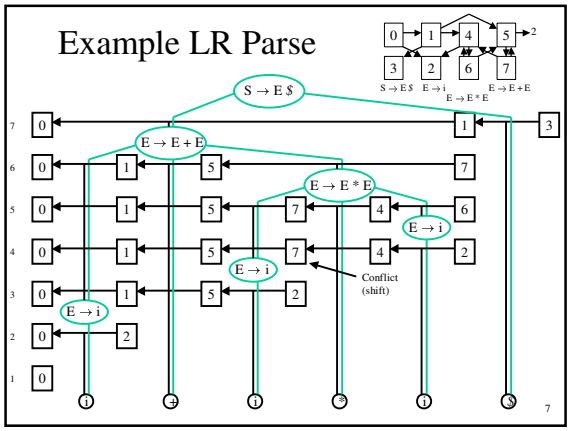
- "L": left-to-right parsing of input
- "R": build rightmost derivation (in reverse)
- Build parse tables ahead of time
- On each token, either
  - shift it, pushing it onto the parse stack, or
  - reduce symbols at top of stack, via some production

Example: Arithmetic

Grammar
$S \rightarrow E\ \$$
$E \rightarrow i$
$E \rightarrow E + E$
$E \rightarrow E * E$



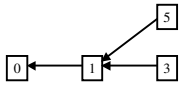
Example LR Parse



7

GLR: Graph-structured stack

- Idea: pursue all possible parses at once
  - Allow stack to be forked into multiple "parsers"
- Alternate between shifts and reduces
- If two parsers enter same state, merge them



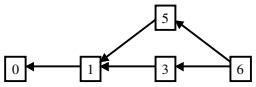
Stack #1 contains 5, 1, 0

Stack #2 contains 3, 1, 0

8

GLR: Graph-structured stack

- Idea: pursue all possible parses at once
  - Allow stack to be forked into multiple "parsers"
- Alternate between shifts and reduces
- If two parsers enter same state, merge them

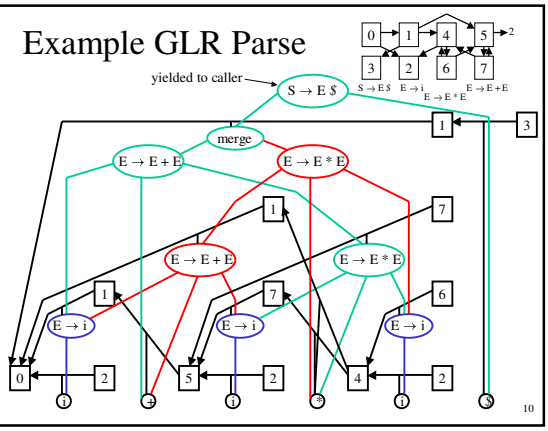


Stack #1 contains 6, 5, 1, 0

Stack #2 contains 6, 3, 1, 0

9

Example GLR Parse



10

Aside: Nondeterminism

- GLR extends LR by making the stack nondeterministic
- Other examples:

DFA	NFA	finite control
LL	LR	finite control
LR	GLR	pushdown stack

11

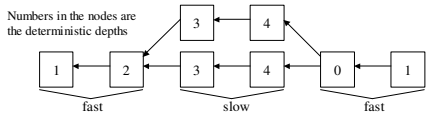
Optimization: Hybrid LR/GLR

- Full GLR is slower than LR due to the cost of interpreting the GSS
- But grammars are likely to be mostly deterministic (mostly linear stack)
- Question: How to recognize when deterministic action is possible?

12

Deterministic Depth

- Answer: In each stack node, remember how deep the stack's determinism goes, e.g.



- Use LR if there's only one active parser, and
  action is a shift, or
  action is reduce by α, len(α) < det_depth

13

Programmatic Interface to GLR

- Other GLR parsers yield parse trees
  - Use a lot of memory
  - Not ideal for later processing stages
  - Commit to a given tree representation
- Challenges with a reduction action model
  - How to undo actions?
  - How to manage merging?
  - How to manage subtree sharing?

14

Elkhound's Interface

- Elkhound lets the user supply:
  - reduction action: one for each production, yields a semantic value (like Bison)
  - `merge()`: given two competing interpretations, return one value
  - `dup()`: prepare a value for being shared
  - `del()`: cancel (delete) a semantic value
- Claim: can build any interface on these

15

Example Elkhound Specification

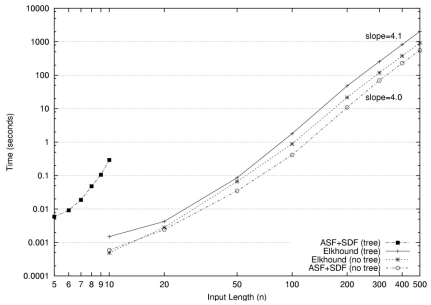
Grammar: E → E + E | b

```
// start symbol
nonterm[PTreeNode*] StartSymbol -> tree:E EOF [ return tree; ]

nonterm[PTreeNode*] E {
  merge(t1, t2)          [ t1->addAlternative(t2); return t1; ]
  del(t)                 []        // rely on garbage collector
  dup(t)                 [ return t; ]

  -> a:E "+" b:E         [ return new PTreeNode("E -> E + E", a, b); ]
  -> "b"                 [ return new PTreeNode("E -> b"); ]
}
```

16

Nondeterministic Performance

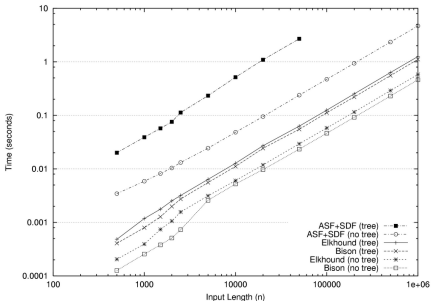
Grammar: E → E + E | b Input: $b(+b)^n$



17

Deterministic Performance

Grammar: E → E + F | F Input: $a(+a)^n$
F → a | (E)



18

Experience Parsing C/C++

- Can we just use the Standard's grammar?
  - Yes: put it in and it works!
  - No: it's not a parsing grammar
    - Fails to make many important distinctions
    - *Massive* number of unnecessary ambiguities
- I've modified the grammar for use with C
  - Ambiguity is useful for parsing `__attribute__`
- What about C++?
  - Need a real C++ type-checker

19

Conclusion

- Elkhound is as fast as Bison but far more capable due to the GLR algorithm
- Two contributions presented:
  - Hybrid LR/GLR optimization
  - General programmatic interface to GLR
- It's available for download now!

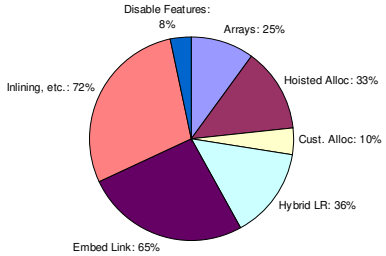
  `www.cs.berkeley.edu/~smcpeak/elkhound`

20

(blank slide)

21

Optimization Techniques



22