Bottom-up parsing

Construct parse tree for input from leaves up

• reducing a string of tokens to single start symbol (inverse of deriving a string of tokens from start symbol)

"Shift-reduce" strategy:

- read ("shift") tokens until seen r.h.s. of "correct" production
- reduce handle to l.h.s. nonterminal, then continue
- · done when all input read and reduced to start nonterminal

LR parsing

LR(k) parsing

- · Left-to-right scan of input, Rightmost derivation
- k tokens of lookahead

Strictly more general than LL(k)

- gets to look at whole rhs of production before deciding what to do, not just first k tokens of rhs
- can handle left recursion and common prefixes fine

Still as efficient as any top-down or bottom-up parsing method

Complex to implement

Craig Chambers

Craig Chambers

• need automatic tools to construct parser from grammar

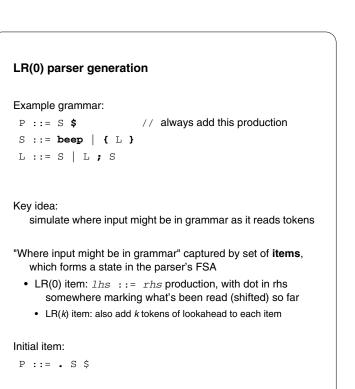
69

CSE 401

CSE 401

Craig Chambers

CSE 401



71

LR parsing tables

Construct parsing tables implementing a FSA with a stack

68

- · rows: states of parser
- · columns: token(s) of lookahead
- entries: action of parser
 - shift, goto state X
 - reduce production "X ::= RHS"
 - accept
 - error

Algorithm to construct FSA similar to algorithm to build DFA from NFA

· each state represents set of possible places in parsing

LR(*k*) algorithm builds huge tables

- LALR(k) algorithm has fewer states \Rightarrow smaller tables
 - less general than LR(k), but still good in practice
 - · size of tables acceptable in practice

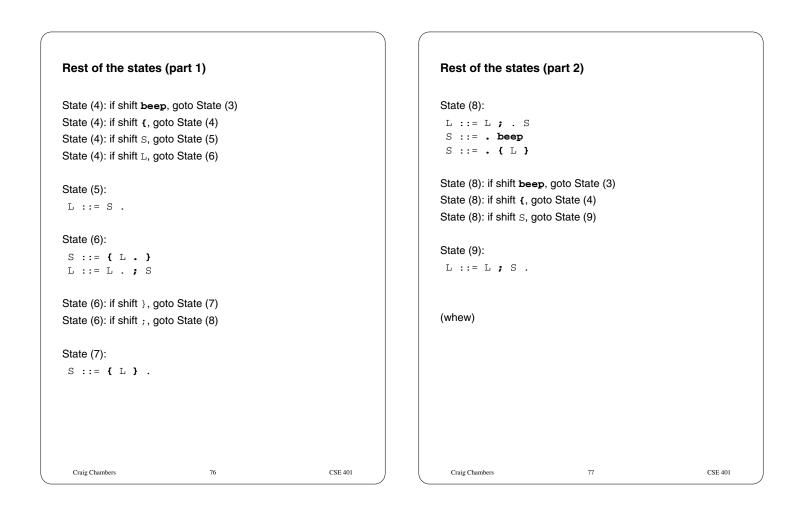
k == 1 in practice

- most parser generators, including ${\tt yacc}$ and ${\tt jflex},$ are LALR(1)

70

| Closure | | State transitions | | | | |
|--|---------------|---------------------------------------|---------------------------|-----------------|--|--|
| Initial state is closure of initial item | | | ompute new state(s) fo | or each symbol | | |
| closure: if dot before non-terminal, add all pro | oductions for | (terminal and non | -terminal) after dot | | | |
| non-terminal with dot at the start | | state transitions | correspond to shift act | ions | | |
| "epsilon transitions" | | | | | | |
| | | New item derived from | n old item by shifting o | dot over symbol | | |
| Initial state (1): | | do closure to cor | npute new state | | | |
| P ::= . S \$ | | | | | | |
| S ::= . beep | | Initial state (1): | | | | |
| S ::= . { L } | | P ::= • S \$ | S ::= . beep | S ::= . { L } | | |
| | | State (2) reached on | transition that shifts S: | | | |
| | | P ::= S . \$ | | | | |
| | | State (3) reached on | transition that shifts be | eep: | | |
| | | S ::= beep . | | | | |
| | | State (4) reached on | transition that shifts {: | | | |
| | | S ::= { . L } | | | | |
| | | L ::= . S | | | | |
| | | L ::= . L ; S | | | | |
| | | S ::= . beep S ::= . { L } | | | | |
| | | 5 ::= • (L) | | | | |
| Craig Chambers 72 | CSE 401 | Craig Chambers | 73 | CSE 401 | | |

Accepting transitions **Reducing states** If state has *lhs* ::= *rhs* . item, then add transition labeled $\ensuremath{\$}$ to the accept action then it has a reduce *lhs* ::= *rhs* action Example: Example: P ::= S . \$ ${\tt S}$::= beep . has transition labeled \$ to accept action has reduce S ::= beep action No label; this state always reduces this production • what if other items in this state shift, or accept? • what if other items in this state reduce differently? Craig Chambers 74 CSE 401 Craig Chambers 75



Building table from the states & transitions

Create a row for each state Create a column for each terminal, non-terminal, and \$

For every "state (*i*): if shift *x* goto state (*j*)" transition:

- if *x* is a terminal, put "shift, goto *j*" action in row *i*, column *x*
- if *x* is a non-terminal, put "goto *j*" action in row *i*, column *x*

For every "state (*i*): if \$ accept" transition:

• put "accept" action in row i, column \$

For every "state (*i*): reduce *lhs* ::= *rhs*" action:

• put "reduce *lhs* ::= *rhs*" action in all columns of row *i*

Table for this grammar

| State | { | } | beep | ; | S | L | \$ |
|-------|--------------------|------|------|------|----|----|----|
| 1 | s,g4 | | s,g3 | | g2 | | |
| 2 | | | | | | | a! |
| 3 | reduce S ::= beep | | | | | | |
| 4 | s,g4 | | s,g3 | | g5 | g6 | |
| 5 | reduce L ::= S | | | | | | |
| 6 | | s,g7 | | s,g8 | | | |
| 7 | reduce S ::= { L } | | | | | | |
| 8 | s,g4 | | s,g3 | | g9 | | |
| 9 | reduce L ::= L ; S | | | | | | |

78

79

| Example | | | Problems in shift-reduce parsing | | | |
|------------------------|----------|---------|---|----|---------|--|
| Input: { beep ; { beee | p } } \$ | | Can write grammars that cannot be handled with shift-reduce parsing | | | |
| | | | Shift/reduce conflict: • state has both shift action(s) and reduce actions | | | |
| | | | Reduce/reduce conflict: • state has more than one reduce action | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| Craig Chambers | 80 | CSE 401 | Craig Chambers | 81 | CSE 401 | |
| | 00 | C3E 401 | Crarg ChallOers | 01 | C3E 401 | |

Shift/reduce conflicts

LR(0) example:

E ::= E + T | T

State:

E ::= E . + T E ::= T .

Can shift + Can reduce \mathbb{E} : : = \mathbb{T}

LR(k) example:

```
S ::= if E then S |
if E then S else S | ...
```

State:

```
S ::= if E then S .
S ::= if E then S . else S
```

```
Can shift else
Can reduce S ::= if E then S
```

82

Avoiding shift/reduce conflicts

Can rewrite grammar to remove conflict

• E.g. MatchedStmt vs. UnmatchedStmt

Can resolve in favor of shift action

- tries to find longest r.h.s. before reducing
- · works well in practice
- yacc, jflex, et al. do this

Craig Chambers

83

Reduce/reduce conflicts Avoiding reduce/reduce conflicts Example: Can rewrite grammar to remove conflict ::= Type **id** ; | LHS **=** Expr ; | ... • can be hard Stmt . . . • e.g. C/C++ declaration vs. expression problem LHS ::= id | LHS [Expr] | ... • e.g. MiniJava array declaration vs. array store problem . . . ::= **id** | Type **[]** | ... Туре Can resolve in favor of one of the reduce actions • but which? - $\tt yacc, jflex,$ et al. pick reduce action for production listed State: textually first in specification Type ::= \mathbf{id} . LHS ::= **id** . Can reduce Type ::= id Can reduce LHS ::= id Craig Chambers 84 CSE 401 Craig Chambers 85 CSE 401