Bottom-Up Parsing

COMP3002

School of Computer Science

Carleton University

Canada's Capital University

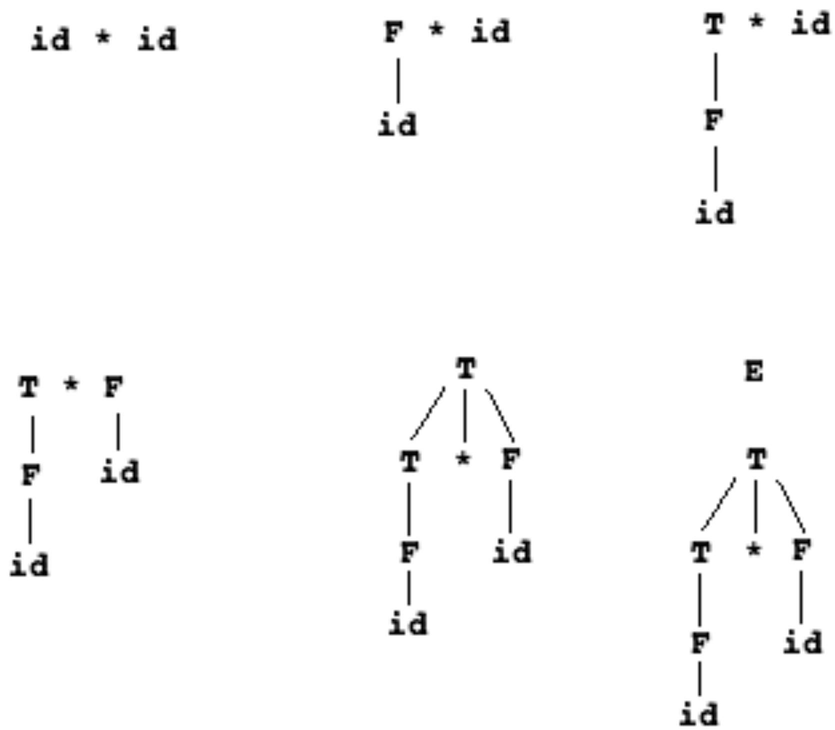
Bottom-Up Parsing

- We start with the leaves of the parse tree (individual tokens) and work our way up to the root.

Carleton University
Canada's Capital University

Example

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid \textbf{id}$$



Reductions

- Find a **handle**
  - Elements in the string that form the right-hand side of a production in the grammar.
- Replace
  - Replace the handle with the left-hand side of the grammar it matches.
- Stop
  - When we end up with only the start symbol, we stop.
- Derivation in reverse
  - Reverse the reductions to get a sequence of derivations.
  - Specifically, a rightmost derivation.

Example (again…)

- We constructed the tree using 5 reductions…

id * **id**

F * **id**

T * **id**

T * *F*

F

E

$$
\begin{array}{l}
E \rightarrow E + T \mid T \\
T \rightarrow T * F \mid F \\
F \rightarrow (E) \mid \mathbf{id}
\end{array}
$$

Carleton UNIVERSITY
Canada's Capital University

Shift-Reduce Parsing

- Shift
  - Shift the next element on the input to the top of the stack.
- Reduce
  - There is a handle on top of the stack. Replace those elements with the left-hand side of the associated production.
- Accept
  - There is no more input to process.
  - The stack consists only of the start symbol.
- Error
  - Syntax error discovered.

Carleton UNIVERSITY
Canada's Capital University

Example (yet again!)

$$
\begin{aligned}
E &\rightarrow E + T \mid T \\
T &\rightarrow T * F \mid F \\
F &\rightarrow (E) \mid \mathbf{id}
\end{aligned}
$$

Carleton
UNIVERSITY
Canada's Capital University

Shift-Reduce Conflict

- Can't decide whether to shift or reduce…

stmt	→	**if** *expr* **then** *stmt*
	|	**if** *expr* **then** *stmt* **else** *stmt*
	|	**other**

- Consider following stack configuration…

STACK	INPUT
$ … **if** *expr* **then** *stmt*	**else** … $

Reduce-Reduce Conflict

- What production to reduce by?

$$
\begin{array}{ccc}
 & \ldots & \\
E & \rightarrow & S \\
F & \rightarrow & S \\
 & \ldots &
\end{array}
$$

- What production to reduce by?

STACK	**INPUT**
$ … **S**	… $

LR Parsers

- LR(k) Parsers
  - L for left-to-right scanning
  - R for rightmost derivation
  - k symbols of lookahead
- Different types…
  - Simple LR (SLR)
  - Canonical-LR
  - LALR

Carleton UNIVERSITY
Canada's Capital University

Why is this good?

- Lookahead is easier
  - LR(k) looks ahead *k* symbols in a right-sentential form, and matches a production.
  - LL(k) tries to recognize a production from the first *k* characters of the string it derives.
  - So, more grammars.
- Error Handling
  - Detect syntax errors as soon as they occur.

Carleton UNIVERSITY
Canada's Capital University

Conflict Resolution

- Construct a finite automaton (FA) that recognizes the right-hand-side of productions by scanning the input from **right to left**.
- Items
  - An item of G is a production of G with a dot at some position of the body.
  - Eg, $A \rightarrow X\,.\,YZ$
  - A state in our FA is a set of items.
- This is ***Simple LR (SLR) Parsing***

Carleton UNIVERSITY
Canada's Capital University

Break

Carleton UNIVERSITY
Canada's Capital University

Constructing the Finite Automaton

We need the Canonical Collection of LR(0) Items.

1. Augment the grammar
2. CLOSURE of items
3. GOTO function between items

Carleton UNIVERSITY
Canada's Capital University

Augmented Grammars

- To augment grammar G with start symbol S, we add a new production $S' \rightarrow S$ and make S' the new start symbol.

E’	→	E
E	→	E + T | T
T	→	T * F | F
F	→	(E) | **id**

Carleton UNIVERSITY
Canada’s Capital University

Closure

- CLOSURE(I), where I is a set of items for a grammar G.
  1. Initially, add every item in I to CLOSURE(I)
  2. If $A \rightarrow x\ .\ B\ y$ is in CLOSURE(I), and $B \rightarrow .\ w$ is a production, add $B \rightarrow .w$ to CLOSURE(I) if it isn't there already.
  3. Apply Rule 2 until no more new items are added to CLOSURE(I).

Carleton UNIVERSITY
Canada's Capital University

Example

- If $I = \{ E' \rightarrow .E \}$, then CLOSURE(I):

$E' \rightarrow E$
$E \rightarrow .\, E + T$
$E \rightarrow .\, T$
$T \rightarrow .T * F$
$T \rightarrow .\, F$
$F \rightarrow .(\, E\,)$
$F \rightarrow .\, \textbf{id}$

E'	→	E
E	→	E + T | T
T	→	T * F | F
F	→	(E) | **id**

Carleton UNIVERSITY
Canada's Capital University

The GOTO Function

- *GOTO(I, X)* defined where *I* is an item and *X* is a grammar symbol.
- Defines the transitions between sets of **items** in the finite automaton.
- If [$A \rightarrow \mathbf{a} \,.\, X\, \mathbf{b}$] is in I, *GOTO(I, X)* contains CLOSURE($A \rightarrow \mathbf{a}\, X \,.\, \mathbf{b}$)

Carleton UNIVERSITY
Canada's Capital University

Example

I

$E' \rightarrow E\,.$

$E \rightarrow E\,.+T$

$GOTO(I, +)$

$E \rightarrow E + .\,T$

$T \rightarrow .\,T * F$

$T \rightarrow .\,F$

$F \rightarrow .\,(E)$

$F \rightarrow .\,\mathbf{id}$

E'	→	E
E	→	E + T | T
T	→	T * F | F
F	→	(E) | **id**

Carleton UNIVERSITY
Canada's Capital University

A fat groundhog

Carleton UNIVERSITY
Canada's Capital University

Canonical Collection of LR(0) items

```
C = CLOSURE( { S' → . S } )
repeat
    for each set of items I in C
        for each grammar symbol X
            if GOTO(I,X) is not empty and not in C
                add GOTO(I,X) to C
until no new sets of items are added to see
```

Carleton UNIVERSITY
Canada's Capital University

244

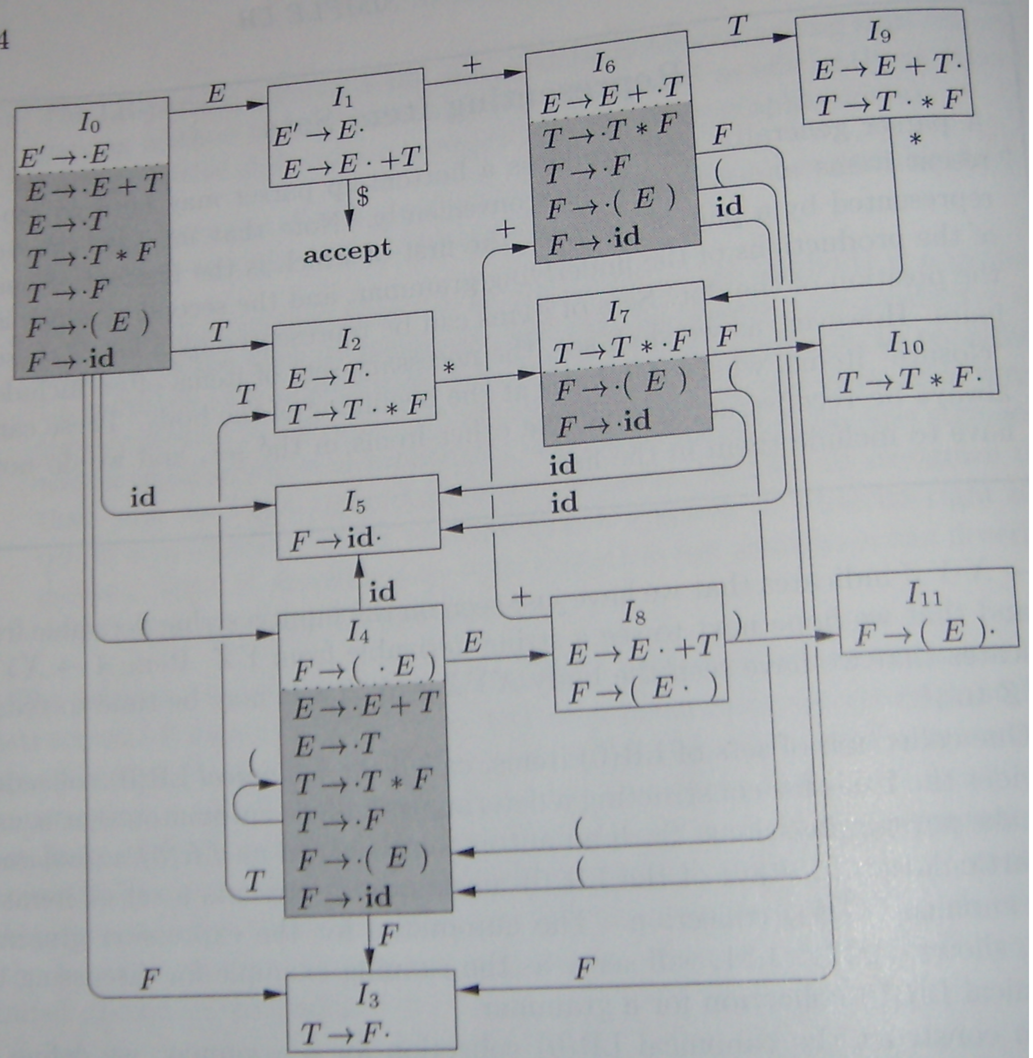


Figure 4.31: LR(0) automaton for the expression grammar (4.1)

Carleton
UNIVERSITY
Canada's Capital University

Simple LR Parsing

- Parse Table
  - ACTION and GOTO functions
  - Built from the finite automaton
- GOTO
  - Defined as before
- ACTION
  - If $[A \rightarrow A\ .\ \mathbf{x}\ B\ ]$ is in I_i, and GOTO(I_i, x) = I_j, then
    - ACTION(i, **x**) = "shift j"
  - If $[A \rightarrow X\ .\ ]$ is in I_i, then
    - ACTION(i, a) = "reduce A →X" for all a in FOLLOW(A).

Carleton UNIVERSITY
Canada's Capital University

Elements of SLR Parser

- Stack
  - Maintains a stack of **states**
  - Used to resolve conflicts.
- Symbols
  - Grammar symbols corresponding to states on the stack

Carleton UNIVERSITY
Canada's Capital University

Shift-Reduce Parsing

- ACTION[s,a] = shift j
  - Push j onto the stack
  - Append a to the input symbols
- ACTION[s,a] = reduce $A \rightarrow X$
  - Pop |X| symbols off the stack
  - Let t be state on top of the stack
  - Push GOTO[t, A] onto the stack
- Accept, Error
  - As before

Carleton UNIVERSITY
Canada's Capital University

Example

STATE	action						goto		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Fig. 4.31. Parsing table for expression grammar.

Carleton UNIVERSITY
Canada's Capital University

Canonical LR Parsing

- In SLR, we always reduce by [$A \rightarrow B$.] on input a if it is in *FOLLOW(A)*.
- However, there may be some prefix *XYZA* that can never be followed by a.

- In Canonical LR Parsing, for each **item** we store a **lookahead** that we have to see before reducing. Eg, [$A \rightarrow B$. (a)]

Carleton UNIVERSITY
Canada's Capital University

LALR

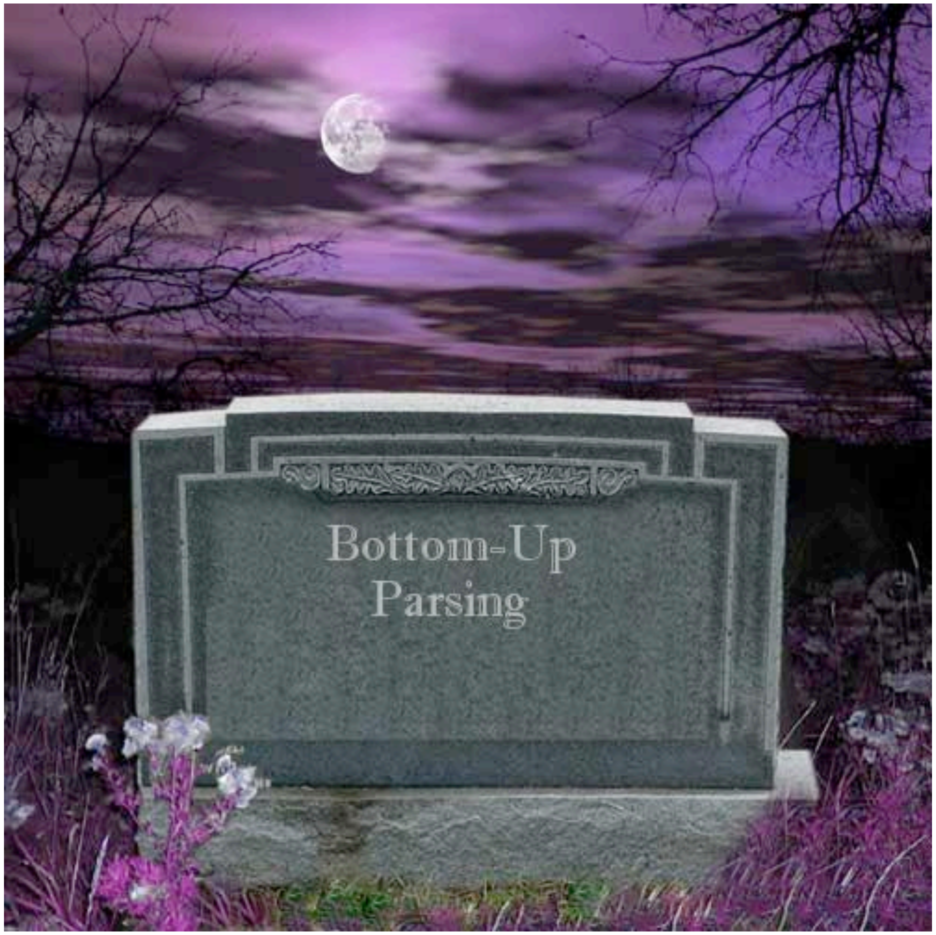
- Lookahead LR
- Canonical LR tables are typically an order of magnitude larger than SLR tables.
- Construct Canonical LR table, prune them.

Final Thoughts

- Hard to implement
  - Compared to LL(k) parsers.
  - In practice, don't construct them.
  - Instead, use parser generators.
- More powerful
  - Every LL(k) grammar is LR(k)
  - Reverse not necessarily true.

Carleton UNIVERSITY
Canada's Capital University

Fin!



Carleton
UNIVERSITY
Canada's Capital University