Top-down syntax analysis (pg. 1)

Lecture topics

I Derivations
II Parser classification
III LL(1) parser tables
IV Left-factoring

I Derivations

Example grammer G_1:

$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to (E) \mid id$

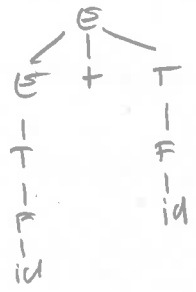
Left-most derivation for id+id

$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T$
$\Rightarrow id+T \Rightarrow id+F \Rightarrow id+id$

Right-most derivation for id+id:

$E \Rightarrow E+T \Rightarrow E+F \Rightarrow E+id$
$\Rightarrow T+id \Rightarrow F+id \Rightarrow id+id$

Parse tree is the same for both cases:



Most parsers work left-to-right
Recursive-descent parsers find left-most derivation
Problem with left recursive grammars:

$E \Rightarrow E+T \Rightarrow E+T+T$
$\Rightarrow E+T+T+T \Rightarrow \cdots$

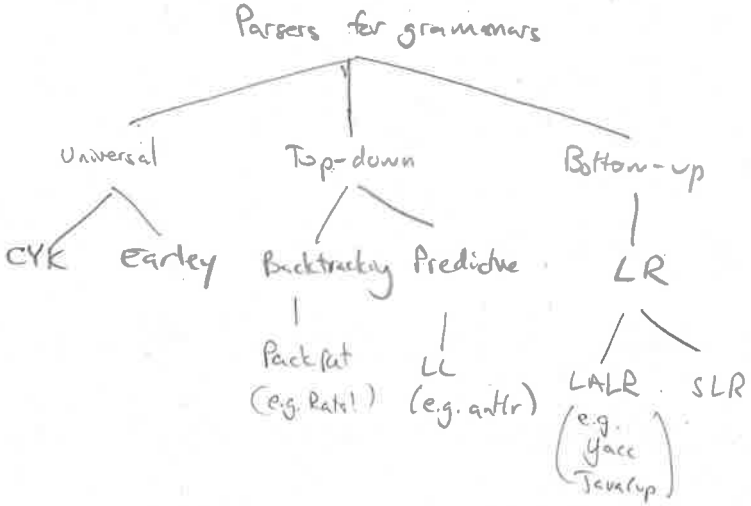
Reminder: Right-recursive predictive grammar G_2:

$E \to TE'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to *FT' \mid \varepsilon$
$F \to (E) \mid id$

Also known as "left recursion elimination"
Example for a comma-seperated list:

$L \to L' \mid \varepsilon$
$L' \to L', a \mid a$ $\Rightarrow$ $L \to aL' \mid \varepsilon$
$L' \to ,aL' \mid \varepsilon$

II Parser classification



LL = left-to-right, left most derivation
Differences:

- performance (Packrat uses memoization, to keep backtracking fast)
- expressiveness (Earley works for all context free grammars)
- ease of use (LL requires both left-recursion elimination and left-factoring)

III LL(1) Parser tables

Helper sets for predictive parsing:

First(α): set of terminals or ε that can begin in strings derived from α
Follow(A): set of terminals or \$ that can appear to the right of A.

\$ = end-of-input marker

Top down syntax analysis (pg. 2)

Example based on G_2:

	FIRST	FOLLOW
E	(, id	), $
E'	+, ε	), $
T	(, id	+,), $
T'	*, ε	+,), $
F	(, id	*, +,), $

Predictive parsing table:

row = left-most non-terminal
column = next input symbol
entry = derivation rule

	id	+	*	(	)	$
E	TE'	/	/	TE'	/	/
E'	/	+TE'	/	/	ε	ε
T	FT'	/	/	FT'	/	/
T'	/	ε	*FT'	/	ε	ε
F	id	/	/	(E)	/	/

Example: parsing id * id
(for LL parser, we show stack top at left)

Stack	Input	Action
E	id * id	E → TE'
TE'	id * id	T → FT'
FT'E'	id * id	F → id
id T'E'	id * id	match(id)
T'E'	* id	T' → * FT'
* FT'E'	* id	match(*)
FT'E'	id	F → id
id T'E'	id	match id
T'E'	$	T' → ε
E'	$	E' → ε
$	$	done

III Left-factoring

Example grammar G_3:

S → if E then S else S
| if E then S

Problem: cannot pick alternative based on FIRST-sets
⇒ need look-ahead > 1, not LL(1)

Solution: rewrite grammar

S → if E then S optElse
optElse → else S | ε

Necessary when using predictive parsers (e.g. antlr)

Not necessary for packrat parsers.