Compiler Construction
Lent Term 2014
Lectures 11---14 (of 16)
CORRECTIONS

Corrections to slides missing prime marks ....

Timothy G. Griffin
tgg22@cam.ac.uk
Computer Laboratory
University of Cambridge

Eliminating Left Recursion

(G2)
S :: = E$

E ::= E + T
| E – T
| T

T ::= T * F
| T / F
| F

F ::= NUM
| ID
| (E)

Note that
E ::= T and
E ::= E + T
will cause problems
since FIRST(T) will be included
in FIRST(E + T) ---- so how can
we decide which poduction
To use based on next token?

Solution: eliminate “left recursion”!

E ::= T E’

E’ ::= + T E’
|

(G6)
S :: = E$

E ::= T E’

E’ ::= + T E’
| – T E’
|

T ::= F T’

T’ ::= * F T’
| / F T’
|

F ::= NUM
| ID
| (E)

Eliminate left recursion

First, Follow, nullable table for G6

	Nullable	FIRST	FOLLOW
S	False	{ (, ID, NUM }	{}
E	False	{ (, ID, NUM }	{), $ }
E’	True	{ +, - }	{), $ }
T	False	{ (, ID, NUM }	{), +, -, $ }
T’	True	{ *, / }	{), +, -, $ }
F	False	{ (, ID, NUM }	{), *, /, +, -, $ }

(G6)
S :: = E$

E ::= T E’

E’ ::= + T E’
| – T E’
|

T ::= F T’

T’ ::= * F T’
| / F T’
|

F ::= NUM
| ID
| (E)

Predictive Parsing Table for G6

Table[X, T] = Set of productions

X ::= Y1…Yk in Table[X, T]
if T in FIRST[Y1 … Yk]
or if (T in FOLLOW[X] and nullable[Y1 … Yk])

NOTE: this could lead to more than one entry! If so, out of luck --- can' t do recursive descent parsing!

	+	*	(	)	ID	NUM	$
S			S ::= E$		S ::= E$	S ::= E$	
E			E ::= T E'		E ::= T E'	E ::= T E'	
E'	E' ::= + T E'			E' ::=			E' ::=
T			T ::= F T'		T ::= F T'	T ::= F T'	
T'	T' ::=	T' ::= * F T'		T' ::=			T' ::=
F			F ::= (E)		F ::= ID	F ::= NUM	

(entries for /, - are similar…)

Left-most derivation is constructed by recursive descent

(G6)
S :: = E$

E ::= T E'

E' ::= + T E'
| – T E'
|

T ::= F T'

T' ::= * F T'
| / F T'
|

F ::= NUM
| ID
| (E)

Left-most derivation

S → E$
→ T E' $
→ F T' E' $
→ (E) T' E' $
→ (T E') T' E' $
→ (F T' E') T' E' $
→ (17 T' E') T' E' $
→ (17 E') T' E' $
→ (17 + T E') T' E' $
→ (17 + F T' E') T' E' $
→ (17 + 4 T' E') T' E' $
→ (17 + 4 E') T' E' $
→ (17 + 4) T' E' $
→ (17 + 4) * F T' E' $
→ ...
→ ...
→ (17 + 4) * (2 – 10) T' E' $
→ (17 + 4) * (2 – 10) E' $
→ (17 + 4) * (2 – 10)

call S()
  on '(' call E()
    on '(' call T()
  .l..
  ...

As a stack machine

S → E$
→ T E' $
→ F T' E' $
→ (E) T' E' $
→ (T E') T' E' $
→ (F T' E') T' E' $
→ (17 T' E') T' E' $
→ (17 E') T' E' $
→ (17 + T E') T' E' $
→ (17 + F T' E') T' E' $
→ (17 + 4 T' E') T' E' $
→ (17 + 4 E') T' E' $
→ (17 + 4) T' E' $
→ (17 + 4) * F T' E' $
→ …
→ …
→ (17 + 4) * (2 – 10) T' E' $
→ (17 + 4) * (2 – 10) E' $
→ (17 + 4) * (2 – 10)

	E$
	T E' $
	F T' E' $
(	E) T' E' $
(	T E') T' E' $
(	F T' E') T' E' $
(17	T' E') T' E' $
(17	E') T' E' $
(17 +	T E') T' E' $
(17 +	F T' E') T' E' $
(17 + 4	T' E') T' E' $
(17 + 4	E') T' E' $
(17 + 4)	T' E' $
(17 + 4) *	F T' E' $
…	
…	
(17 + 4) * (2 – 10)	T' E' $
(17 + 4) * (2 – 10)	E' $
(17 + 4) * (2 – 10)	