



Compilers

Bottom-Up Parsing

- Bottom-up parsing is more general than (deterministic) top-down parsing
 - And just as efficient
 - Builds on ideas in top-down parsing
- Bottom-up is the preferred method

- Bottom-up parsers don't need left-factored grammars
- Revert to the “natural” grammar for our example:

$$E \rightarrow T + E \mid T$$

$$T \rightarrow \text{int} * T \mid \text{int} \mid (E)$$

- Consider the string: $\text{int} * \text{int} + \text{int}$

Bottom-up parsing *reduces* a string to the start symbol
by inverting productions

int * int + int

int * T + int

T + int

T + T

T + E

E

$T \rightarrow \text{int}$

$T \rightarrow \text{int} * T$

$T \rightarrow \text{int}$

$E \rightarrow T$

$E \rightarrow T + E$

Note the productions, read backwards, trace a rightmost derivation

int * int + int

int * T + int

T + int

T + T

T + E

E

$T \rightarrow \text{int}$

$T \rightarrow \text{int} * T$

$T \rightarrow \text{int}$

$E \rightarrow T$

$E \rightarrow T + E$

Important Fact #1 about bottom-up parsing:

A bottom-up parser traces a rightmost derivation in reverse

Bottom-Up Parsing

int * int + int

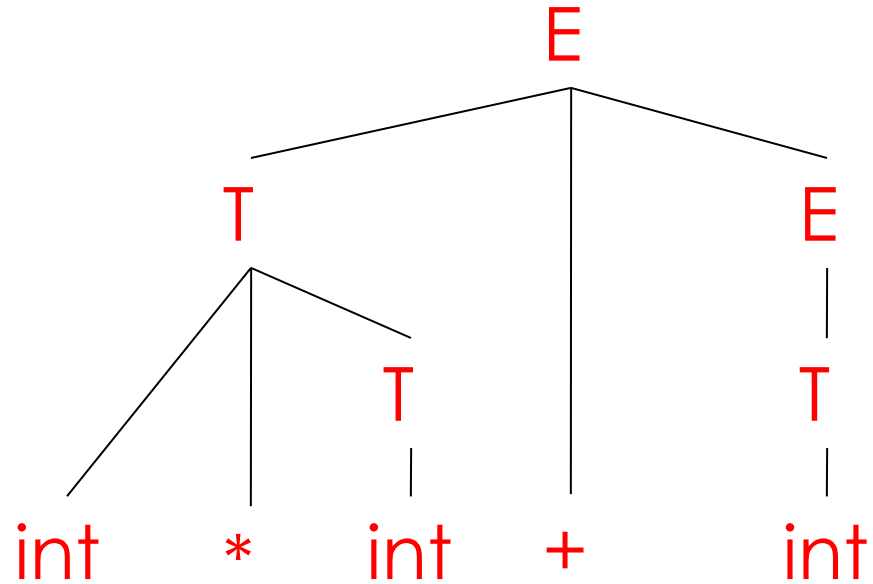
int * T + int

T + int

T + T

T + E

E



Bottom-Up Parsing

int * int + int

int * int + int

Bottom-Up Parsing

int * int + int

int * T + int

int * int + int

T

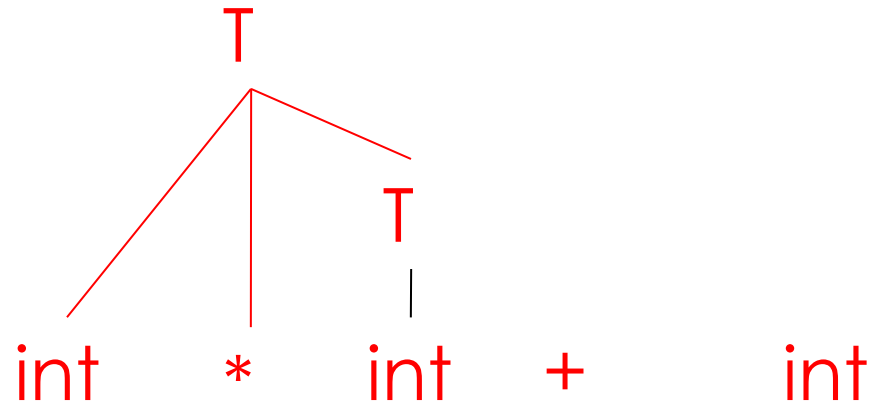
|

Bottom-Up Parsing

int * int + int

int * T + int

T + int



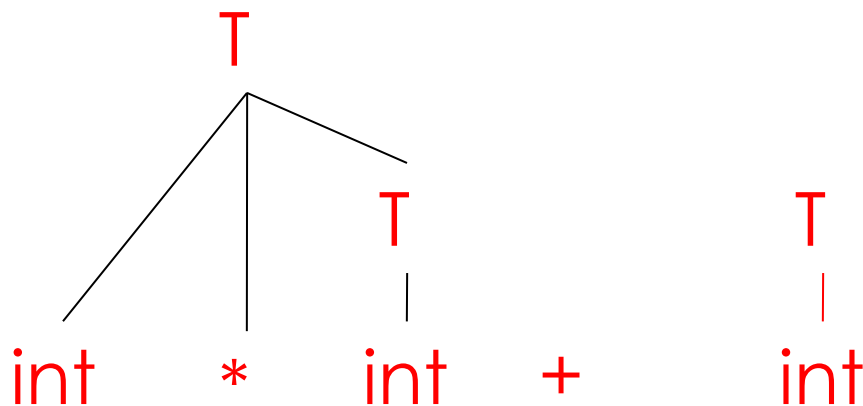
Bottom-Up Parsing

int * int + int

int * T + int

T + int

T + T



Bottom-Up Parsing

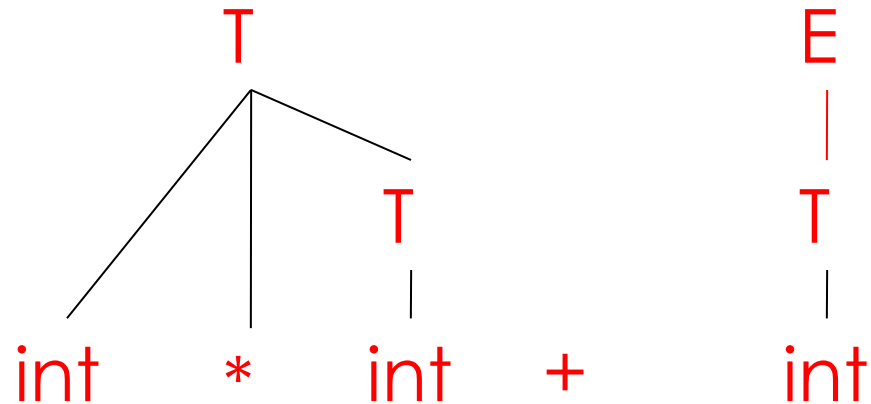
int * int + int

int * T + int

T + int

T + T

T + E



Bottom-Up Parsing

int * int + int

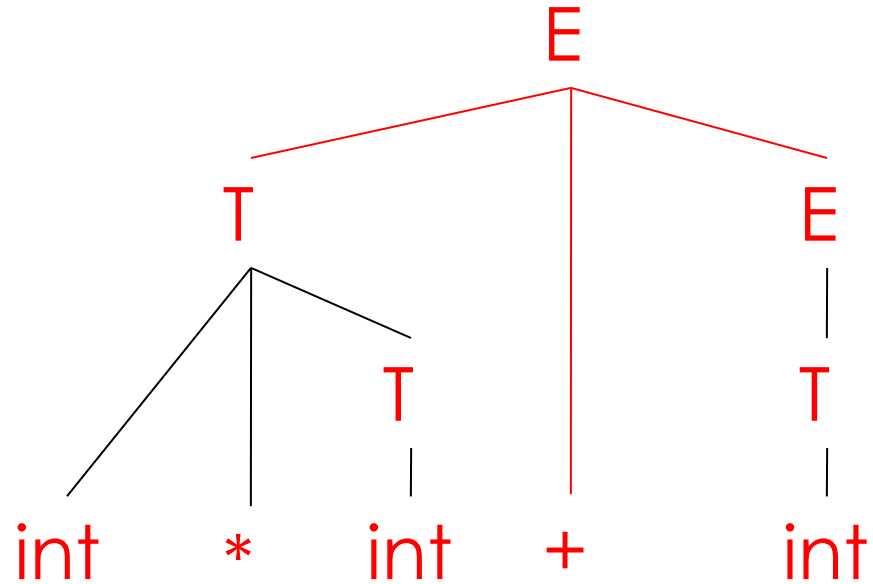
int * T + int

T + int

T + T

T + E

E



For the given grammar, what is the correct series of reductions for the string: $-(id + id) + id$

Bottom-Up Parsing

$$E \rightarrow E' \mid E' + E$$

$$E' \rightarrow -E' \mid id \mid (E)$$

