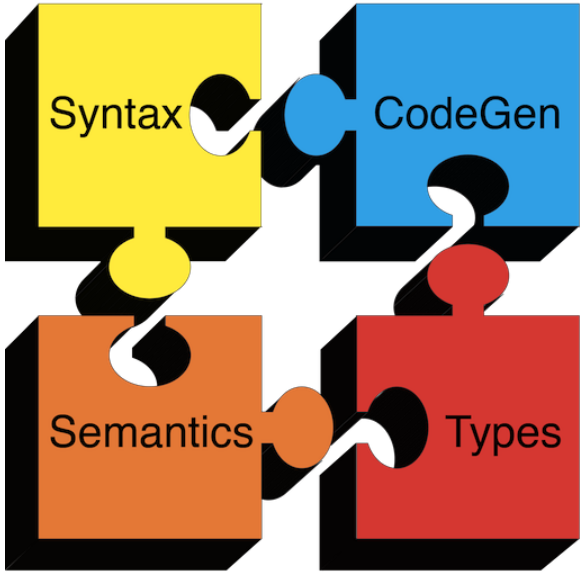


Compilers

Shift-Reduce Parsing

Alex Aiken

Important Fact #1 about bottom-up parsing:

A bottom-up parser traces a rightmost derivation in reverse

Shift-Reduce Parsing

Important Fact #1 has an interesting consequence:

- Let $\alpha\beta\omega$ be a step of a bottom-up parse
- Assume the next reduction is by $X \rightarrow \beta$
- Then ω is a string of terminals

Why? Because $\alpha X \omega \rightarrow \alpha\beta\omega$ is a step in a right-most derivation

Shift-Reduce Parsing

- Idea: Split string into two substrings
  - Right substring is as yet unexamined by parsing
  - Left substring has terminals and non-terminals
  - The dividing point is marked by a |

unexamined input

$\alpha X | w$

Shift-Reduce Parsing

Bottom-up parsing uses only two kinds of actions:

Shift

Reduce

Shift-Reduce Parsing

- *Shift:* Move | one place to the right
  - Shifts a terminal to the left string

ABC|xyz ⇒ ABCx|yz

Shift-Reduce Parsing

- Apply an inverse production at the right end of the left string
  - If $A \rightarrow xy$ is a production, then

$$Cbxy|ijk \Rightarrow CbA|ijk$$

Shift-Reduce Parsing

int * int | + int	reduce T → int
int * T | + int	reduce T → int * T
T + int |	reduce T → int
T + T |	reduce E → T
T + E |	reduce E → T + E
E |	

Shift-Reduce Parsing

|int * int + int	shift
int | * int + int	shift
int * | int + int	shift
int * int | + int	reduce T → int
int * T | + int	reduce T → int * T
T | + int	shift
T + | int	shift
T + int |	reduce T → int
T + T |	reduce E → T
T + E |	reduce E → T + E
E |	

Shift-Reduce Parsing

|int * int + int

int * int + int

Shift-Reduce Parsing

|int * int + int

int | * int + int

int ↑ * int + int

Shift-Reduce Parsing

|int * int + int

int | * int + int

int * | int + int

int * int + int

↑

Shift-Reduce Parsing

|int * int + int

int | * int + int

int * | int + int

int * int | + int

int * int + int

↑

Shift-Reduce Parsing

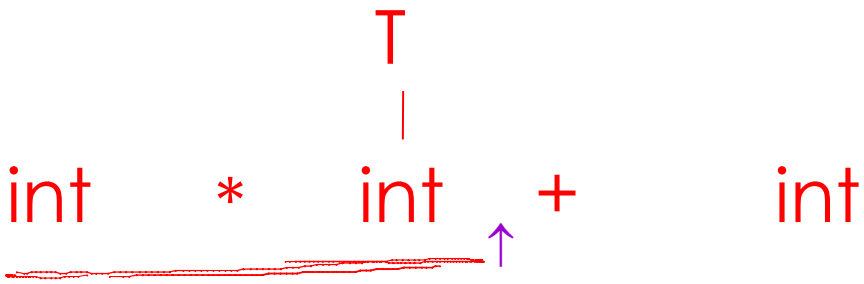
|int * int + int

int | * int + int

int * | int + int

int * int | + int

int * T | + int



Shift-Reduce Parsing

|int * int + int

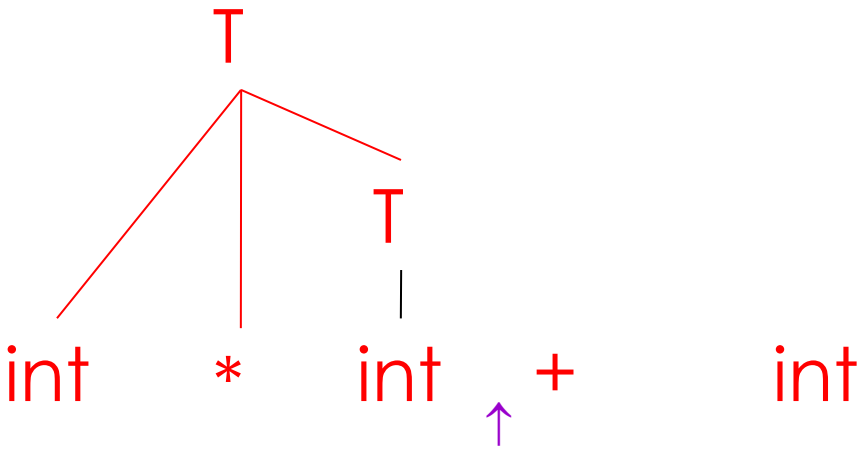
int | * int + int

int * | int + int

int * int | + int

int * T | + int

T | + int



Shift-Reduce Parsing

|int * int + int

int | * int + int

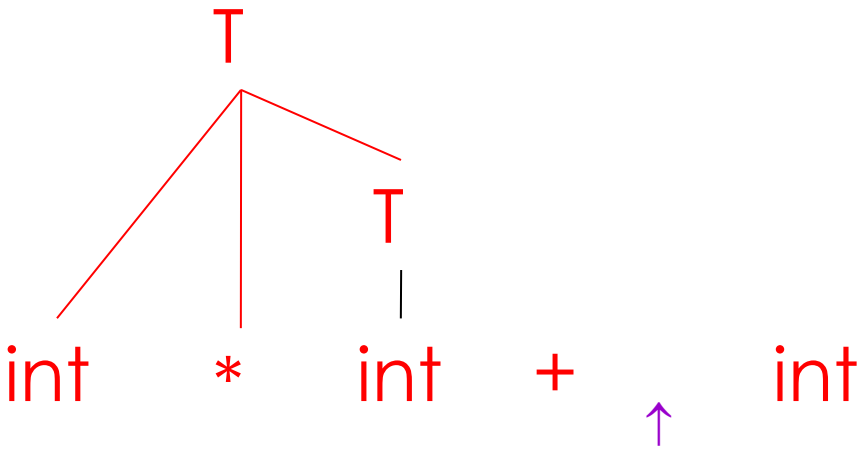
int * | int + int

int * int | + int

int * T | + int

T | + int

T + | int



Shift-Reduce Parsing

|int * int + int

int | * int + int

int * | int + int

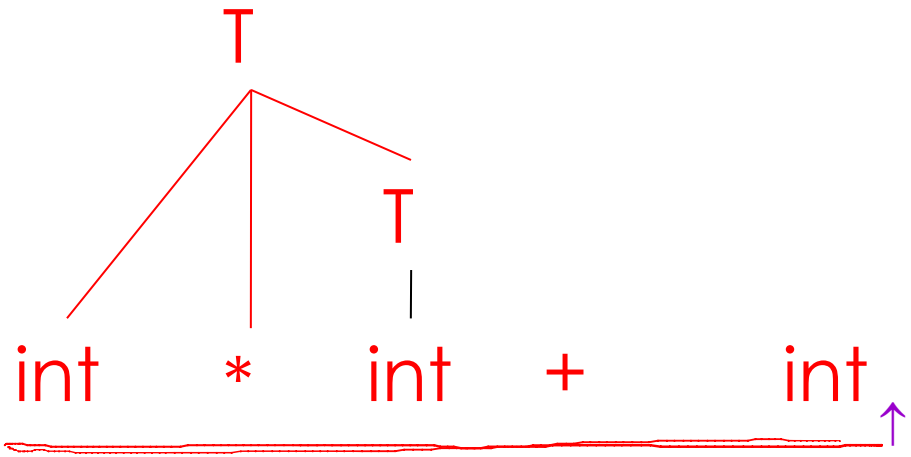
int * int | + int

int * T | + int

T | + int

T + | int

T + int |



Shift-Reduce Parsing

|int * int + int

int | * int + int

int * | int + int

int * int | + int

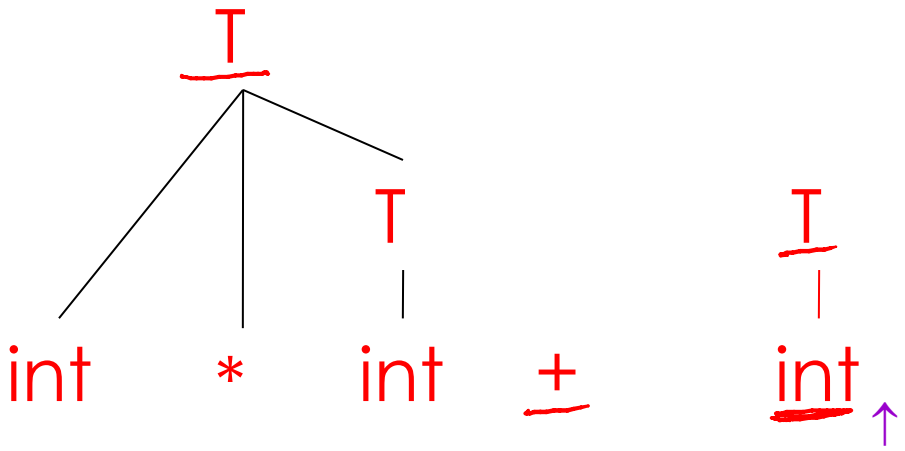
int * T | + int

T | + int

T + | int

T + int |

T + T |



Shift-Reduce Parsing

|int * int + int

int | * int + int

int * | int + int

int * int | + int

int * T | + int

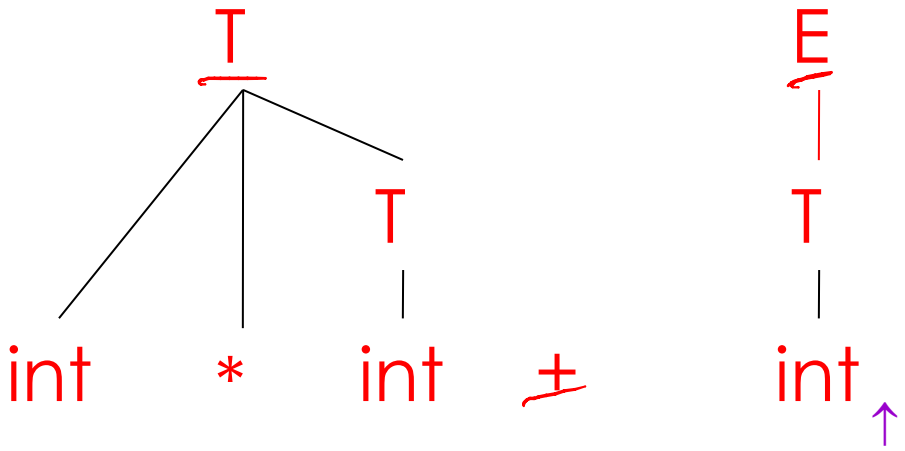
T | + int

T + | int

T + int |

T + T |

T + E |



Shift-Reduce Parsing

|int * int + int

int | * int + int

int * | int + int

int * int | + int

int * T | + int

T | + int

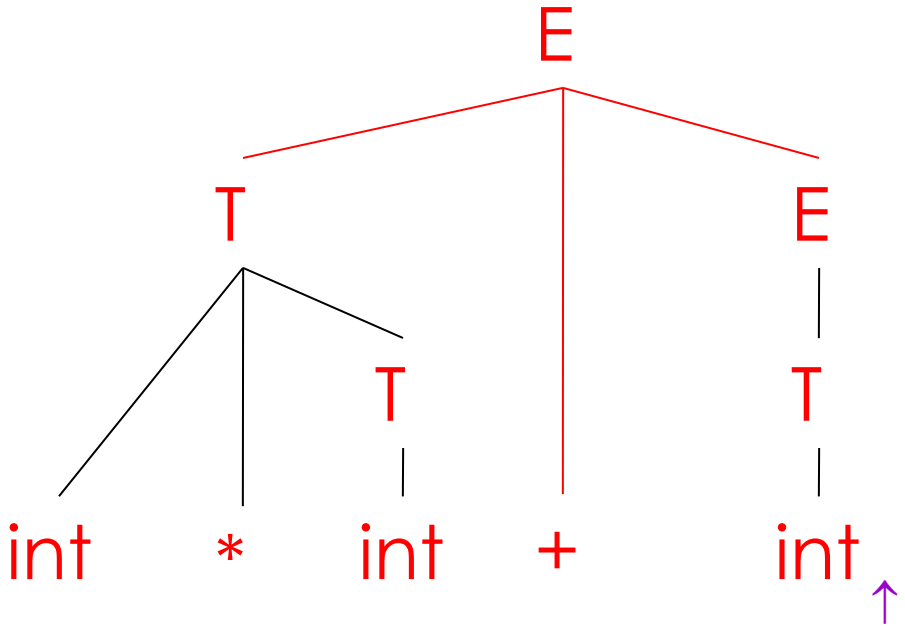
T + | int

T + int |

T + T |

T + E |

E |



Alex Aiken

Shift-Reduce Parsing

For the given grammar, what is the correct shift-reduce parse for the string: id + -id

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

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

|id + -id
id|+ -id
E' +|-id
E' + -|id
E' + -id|
E' + -E'|
E' + E'|
E' + E|
E|

|id + -id
id|+ -id
id +|-id
id + -|id
id + -id|
id + -E'|
id + E'|
id + E|
E' + E|
E|

|id + -id
|E' + -id
E'|+ -id
E' +|-id
E' + -|id
E' + -|E'
E' +|-E'
E' +|E'
E' +|E
E'|+ E
|E' + E
|E

|id + -id
id|+ -id
E'|+ -id
E' +|-id
E' + -|id
E' + -id|
E' + -E'|
E' + E'|
E' + E|
E|

Shift-Reduce Parsing

- Left string can be implemented by a stack
  - Top of the stack is the |
- Shift pushes a terminal on the stack
- Reduce
  - pops symbols off of the stack (production rhs)
  - pushes a non-terminal on the stack (production lhs)

Shift-Reduce Parsing

- In a given state, more than one action (shift or reduce) may lead to a valid parse
- If it is legal to shift or reduce, there is a *shift-reduce* conflict
- If it is legal to reduce by two different productions, there is a *reduce-reduce* conflict Bad