RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

# CS415 Compilers
# Syntax Analysis
# Bottom-up Parsing

These slides are based on slides copyrighted by Keith Cooper, Ken Kennedy & Linda Torczon at Rice University

# Parsing (Syntax Analysis)

EAC Chapters 3.4

# Parsing Techniques: Bottom-up parsers

*LR(1), operator precedence*

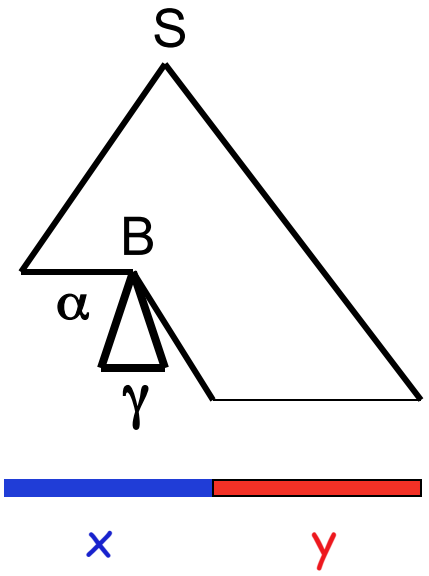
1 input symbol lookahead

construct rightmost deriviation (backwards)

input: read left-to-right

rule B ::= γ

$$S \Rightarrow^*_{rm} \alpha B y \Rightarrow_{rm} \alpha \gamma y \Rightarrow^*_{rm} x\ y$$



# Finding Reductions

Consider the simple grammar

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | a *A B* e |
| 2 | *A* | → | *A* b c |
| 3 | | \| | b |
| 4 | *B* | → | d |

And the input string abbcde

| *Sentential Form* | *Next Reduction* *Prod'n* | *Pos'n* |
|---|---|---|
| abbcde | | |
| | | |
| | | |
| | | |
| *Goal* | | |

*The trick is scanning the input and finding the next reduction*
*The mechanism for doing this must be efficient*

# Finding Reductions

Consider the simple grammar

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | a *A B* e |
| 2 | *A* | → | *A* b c |
| 3 | | \| | b |
| 4 | *B* | → | d |

And the input string abbcde

| *Sentential Form* | *Next Reduction* | |
|---|---|---|
| | *Prod'n* | *Pos'n* |
| abbcde | 3 | 2 |
| | | |
| | | |
| | | |
| *Goal* | | |

*The trick is scanning the input and finding the next reduction*
*The mechanism for doing this must be efficient*

# Finding Reductions

Consider the simple grammar

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | a *A B* e |
| 2 | *A* | → | *A* b c |
| 3 | | \| | b |
| 4 | *B* | → | d |

And the input string abbcde

| *Sentential Form* | *Next Reduction* *Prod'n* | *Pos'n* |
|---|---|---|
| abbcde | 3 | 2 |
| a *A* bcde | | |
| *Goal* | | |

*The trick is scanning the input and finding the next reduction*
*The mechanism for doing this must be efficient*

# Finding Reductions

Consider the simple grammar

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | a *A B* e |
| 2 | *A* | → | *A* b c |
| 3 | | \| | b |
| 4 | *B* | → | d |

And the input string abbcde

| *Sentential Form* | *Next Reduction* *Prod'n* | *Pos'n* |
|---|---|---|
| abbcde | 3 | 2 |
| a *A* bcde | 2 | 4 |
| *Goal* | | |

*The trick is scanning the input and finding the next reduction*
*The mechanism for doing this must be efficient*

# Finding Reductions

Consider the simple grammar

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | a *A B* e |
| 2 | *A* | → | *A* b c |
| 3 | | \| | b |
| 4 | *B* | → | d |

And the input string abbcde

| *Sentential Form* | *Next Reduction* *Prod'n* | *Pos'n* |
|---|---|---|
| abbcde | 3 | 2 |
| a *A* bcde | 2 | 4 |
| a *A* de | | |
| | | |
| *Goal* | | |

*The trick is scanning the input and finding the next reduction*
*The mechanism for doing this must be efficient*

# Finding Reductions

Consider the simple grammar

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | a *A B* e |
| 2 | *A* | → | *A* b c |
| 3 | | \| | b |
| 4 | *B* | → | d |

And the input string abbcde

| *Sentential Form* | *Next Reduction* *Prod'n* | *Pos'n* |
|---|---|---|
| abbcde | 3 | 2 |
| a *A* bcde | 2 | 4 |
| a *A* de | 4 | 3 |
| *Goal* | | |

*The trick is scanning the input and finding the next reduction*
*The mechanism for doing this must be efficient*

# Finding Reductions

Consider the simple grammar

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | a *A B* e |
| 2 | *A* | → | *A* b c |
| 3 | | \| | b |
| 4 | *B* | → | d |

And the input string abbcde

| *Sentential Form* | *Next Reduction* *Prod'n* | *Pos'n* |
|---|---|---|
| abbcde | 3 | 2 |
| a *A* bcde | 2 | 4 |
| a *A* de | 4 | 3 |
| a *A B* e | | |
| *Goal* | | |

*The trick is scanning the input and finding the next reduction*
*The mechanism for doing this must be efficient*

# Finding Reductions

Consider the simple grammar

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | a *A B* e |
| 2 | *A* | → | *A* b c |
| 3 | | \| | b |
| 4 | *B* | → | d |

And the input string abbcde

| *Sentential Form* | *Next Reduction* | |
|---|---|---|
| | *Prod'n* | *Pos'n* |
| abbcde | 3 | 2 |
| a *A* bcde | 2 | |
| a *A* de | 4 | |
| a *A B* e | 1 | |
| *Goal* | | |

*The trick is scanning the input and finding the next reduction*
*The mechanism for doing this must be efficient*

# Finding Reductions (Handles)

The parser must find a substring $\beta$ of the tree's frontier that

*matches some production $A \rightarrow \beta$ that occurs as one step in the rightmost derivation*

Informally, we call this substring $\beta$ a *handle*

Formally,

A *handle* of a right-sentential form $\gamma$ is a pair $\langle A \rightarrow \beta, k \rangle$ where $A \rightarrow \beta \in P$ and $k$ is the position in $\gamma$ of $\beta$'s rightmost symbol.

If $\langle A \rightarrow \beta, k \rangle$ is a handle, then replacing $\beta$ at $k$ with $A$ produces the right sentential form from which $\gamma$ is derived in the rightmost derivation.

Because $\gamma$ is a right-sentential form, the substring to the right of a handle contains only terminal symbols

$\Rightarrow$ the parser doesn't need to scan past the handle (only lookahead)

$\Rightarrow$ The right end of the handle will be on top of the stack, not within the stack. Need lookahead to determine whether we reached the handle.

# Finding Reductions (Handles)

Critical Insight

> *If G is unambiguous, then every right-sentential form has a unique handle.*

If we can find those handles, we can build a derivation !

Sketch of Proof:

1. *G* is unambiguous ⇒ rightmost derivation is unique
2. ⇒ a unique production $A \rightarrow \beta$ applied to derive $\gamma_i$ from $\gamma_{i-1}$
3. ⇒ a unique position *k* at which $A \rightarrow \beta$ is applied
4. ⇒ a unique handle $\langle A \rightarrow \beta, k \rangle$

This all follows from the definitions

# Example (a very busy slide)

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | *Expr* |
| 2 | *Expr* | → | *Expr* + *Term* |
| 3 | | \| | *Expr* - *Term* |
| 4 | | \| | *Term* |
| 5 | *Term* | → | *Term* * *Factor* |
| 6 | | \| | *Term* / *Factor* |
| 7 | | \| | *Factor* |
| 8 | *Factor* | → | number |
| 9 | | \| | id |
| 10 | | \| | ( *Expr* ) |

*The expression grammar*

| *Prod'n.* | *Sentential Form* | *Handle* |
|---|---|---|
| — | *Goal* | — |
| 1 | *Expr* | |
| 3 | *Expr* - *Term* | |
| 5 | *Expr* - *Term* * *Factor* | |
| 9 | *Expr* - *Term* * <id,y> | |
| 7 | *Expr* - *Factor* * <id,y> | |
| 8 | *Expr* - <num,2> * <id,y> | |
| 4 | *Term* - <num,2> * <id,y> | |
| 7 | *Factor* - <num,2> * <id,y> | |
| 9 | <id,x> - <num,2> * <id,y> | |

*Handles for rightmost derivation of* x - 2 * y

# Example (a very busy slide)

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | *Expr* |
| 2 | *Expr* | → | *Expr* + *Term* |
| 3 | | \| | *Expr* - *Term* |
| 4 | | \| | *Term* |
| 5 | *Term* | → | *Term* * *Factor* |
| 6 | | \| | *Term* / *Factor* |
| 7 | | \| | *Factor* |
| 8 | *Factor* | → | number |
| 9 | | \| | id |
| 10 | | \| | ( *Expr* ) |

*The expression grammar*

| *Prod'n.* | *Sentential Form* | *Handle* |
|---|---|---|
| — | *Goal* | — |
| 1 | *Expr* | |
| 3 | *Expr* - *Term* | |
| 5 | *Expr* - *Term* * *Factor* | |
| 9 | *Expr* - *Term* * <id,y> | |
| 7 | *Expr* - *Factor* * <id,y> | |
| 8 | *Expr* - <num,2> * <id,y> | |
| 4 | *Term* - <num,2> * <id,y> | |
| 7 | *Factor* - <num,2> * <id,y> | |
| 9 | <id,x> - <num,2> * <id,y> | 9,1 |

*Handles for rightmost derivation of* x - 2 * y

# Example (a very busy slide)

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | *Expr* |
| 2 | *Expr* | → | *Expr* + *Term* |
| 3 | | \| | *Expr* - *Term* |
| 4 | | \| | *Term* |
| 5 | *Term* | → | *Term* * *Factor* |
| 6 | | \| | *Term* / *Factor* |
| 7 | | \| | *Factor* |
| 8 | *Factor* | → | number |
| 9 | | \| | id |
| 10 | | \| | ( *Expr* ) |

*The expression grammar*

| *Prod'n.* | *Sentential Form* | *Handle* |
|---|---|---|
| — | *Goal* | — |
| 1 | *Expr* | |
| 3 | *Expr* - *Term* | |
| 5 | *Expr* - *Term* * *Factor* | |
| 9 | *Expr* - *Term* * <id,y> | |
| 7 | *Expr* - *Factor* * <id,y> | |
| 8 | *Expr* - <num,2> * <id,y> | |
| 4 | *Term* - <num,2> * <id,y> | |
| 7 | *Factor* - <num,2> * <id,y> | 7,1 |
| 9 | <id,x> - <num,2> * <id,y> | 9,1 |

*Handles for rightmost derivation of* x - 2 * y

# Example (a very busy slide)

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | *Expr* |
| 2 | *Expr* | → | *Expr* + *Term* |
| 3 | | \| | *Expr* - *Term* |
| 4 | | \| | *Term* |
| 5 | *Term* | → | *Term* * *Factor* |
| 6 | | \| | *Term* / *Factor* |
| 7 | | \| | *Factor* |
| 8 | *Factor* | → | number |
| 9 | | \| | id |
| 10 | | \| | ( *Expr* ) |

*The expression grammar*

| *Prod'n.* | *Sentential Form* | *Handle* |
|---|---|---|
| — | *Goal* | — |
| 1 | *Expr* | |
| 3 | *Expr* - *Term* | |
| 5 | *Expr* - *Term* * *Factor* | |
| 9 | *Expr* - *Term* * <id,y> | |
| 7 | *Expr* - *Factor* * <id,y> | |
| 8 | *Expr* - <num,2> * <id,y> | |
| 4 | *Term* - <num,2> * <id,y> | 4,1 |
| 7 | *Factor* - <num,2> * <id,y> | 7,1 |
| 9 | <id,x> - <num,2> * <id,y> | 9,1 |

*Handles for rightmost derivation of* x - 2 * y

# Example (a very busy slide)

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | *Expr* |
| 2 | *Expr* | → | *Expr* + *Term* |
| 3 | | \| | *Expr* - *Term* |
| 4 | | \| | *Term* |
| 5 | *Term* | → | *Term* * *Factor* |
| 6 | | \| | *Term* / *Factor* |
| 7 | | \| | *Factor* |
| 8 | *Factor* | → | number |
| 9 | | \| | id |
| 10 | | \| | ( *Expr* ) |

*The expression grammar*

| *Prod'n.* | *Sentential Form* | *Handle* |
|---|---|---|
| — | *Goal* | — |
| 1 | *Expr* | |
| 3 | *Expr* - *Term* | |
| 5 | *Expr* - *Term* * *Factor* | |
| 9 | *Expr* - *Term* * <id,y> | |
| 7 | *Expr* - *Factor* * <id,y> | |
| 8 | *Expr* - <num,2> * <id,y> | 8,3 |
| 4 | *Term* - <num,2> * <id,y> | 4,1 |
| 7 | *Factor* - <num,2> * <id,y> | 7,1 |
| 9 | <id,x> - <num,2> * <id,y> | 9,1 |

*Handles for rightmost derivation of* x - 2 * y

# Example (a very busy slide)

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | *Expr* |
| 2 | *Expr* | → | *Expr* + *Term* |
| 3 | | \| | *Expr* - *Term* |
| 4 | | \| | *Term* |
| 5 | *Term* | → | *Term* * *Factor* |
| 6 | | \| | *Term* / *Factor* |
| 7 | | \| | *Factor* |
| 8 | *Factor* | → | number |
| 9 | | \| | id |
| 10 | | \| | ( *Expr* ) |

*The expression grammar*

| *Prod'n.* | *Sentential Form* | *Handle* |
|---|---|---|
| — | *Goal* | — |
| 1 | *Expr* | |
| 3 | *Expr* - *Term* | |
| 5 | *Expr* - *Term* * *Factor* | |
| 9 | *Expr* - *Term* * <id,y> | |
| 7 | *Expr* - *Factor* * <id,y> | 7,3 |
| 8 | *Expr* - <num,2> * <id,y> | 8,3 |
| 4 | *Term* - <num,2> * <id,y> | 4,1 |
| 7 | *Factor* - <num,2> * <id,y> | 7,1 |
| 9 | <id,x> - <num,2> * <id,y> | 9,1 |

*Handles for rightmost derivation of* x - 2 * y

# Example (a very busy slide)

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | *Expr* |
| 2 | *Expr* | → | *Expr* + *Term* |
| 3 | | \| | *Expr* - *Term* |
| 4 | | \| | *Term* |
| 5 | *Term* | → | *Term* * *Factor* |
| 6 | | \| | *Term* / *Factor* |
| 7 | | \| | *Factor* |
| 8 | *Factor* | → | number |
| 9 | | \| | id |
| 10 | | \| | ( *Expr* ) |

*The expression grammar*

| *Prod'n.* | *Sentential Form* | *Handle* |
|---|---|---|
| — | *Goal* | — |
| 1 | *Expr* | |
| 3 | *Expr* - *Term* | |
| 5 | *Expr* - *Term* * *Factor* | |
| 9 | *Expr* - *Term* * <id,y> | 9,5 |
| 7 | *Expr* - *Factor* * <id,y> | 7,3 |
| 8 | *Expr* - <num,2> * <id,y> | 8,3 |
| 4 | *Term* - <num,2> * <id,y> | 4,1 |
| 7 | *Factor* - <num,2> * <id,y> | 7,1 |
| 9 | <id,x> - <num,2> * <id,y> | 9,1 |

*Handles for rightmost derivation of* x - 2 * y

# Example (a very busy slide)

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | *Expr* |
| 2 | *Expr* | → | *Expr* + *Term* |
| 3 | | \| | *Expr* - *Term* |
| 4 | | \| | *Term* |
| 5 | *Term* | → | *Term* * *Factor* |
| 6 | | \| | *Term* / *Factor* |
| 7 | | \| | *Factor* |
| 8 | *Factor* | → | number |
| 9 | | \| | id |
| 10 | | \| | ( *Expr* ) |

*The expression grammar*

| *Prod'n.* | *Sentential Form* | *Handle* |
|---|---|---|
| — | *Goal* | — |
| 1 | *Expr* | |
| 3 | *Expr* - *Term* | |
| 5 | *Expr* - *Term* * *Factor* | 5,5 |
| 9 | *Expr* - *Term* * <id,y> | 9,5 |
| 7 | *Expr* - *Factor* * <id,y> | 7,3 |
| 8 | *Expr* - <num,2> * <id,y> | 8,3 |
| 4 | *Term* - <num,2> * <id,y> | 4,1 |
| 7 | *Factor* - <num,2> * <id,y> | 7,1 |
| 9 | <id,x> - <num,2> * <id,y> | 9,1 |

*Handles for rightmost derivation of* x - 2 * y

# Example (a very busy slide)

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | *Expr* |
| 2 | *Expr* | → | *Expr* + *Term* |
| 3 | | \| | *Expr* - *Term* |
| 4 | | \| | *Term* |
| 5 | *Term* | → | *Term* * *Factor* |
| 6 | | \| | *Term* / *Factor* |
| 7 | | \| | *Factor* |
| 8 | *Factor* | → | number |
| 9 | | \| | id |
| 10 | | \| | ( *Expr* ) |

*The expression grammar*

| *Prod'n.* | *Sentential Form* | *Handle* |
|---|---|---|
| — | *Goal* | — |
| 1 | *Expr* | |
| 3 | *Expr* - *Term* | 3,3 |
| 5 | *Expr* - *Term* * *Factor* | 5,5 |
| 9 | *Expr* - *Term* * <id,y> | 9,5 |
| 7 | *Expr* - *Factor* * <id,y> | 7,3 |
| 8 | *Expr* - <num,2> * <id,y> | 8,3 |
| 4 | *Term* - <num,2> * <id,y> | 4,1 |
| 7 | *Factor* - <num,2> * <id,y> | 7,1 |
| 9 | <id,x> - <num,2> * <id,y> | 9,1 |

*Handles for rightmost derivation of* x - 2 * y

# Example (a very busy slide)

| | | | |
|---|---|---|---|
| 1 | *Goal* | → | *Expr* |
| 2 | *Expr* | → | *Expr* + *Term* |
| 3 | | \| | *Expr* - *Term* |
| 4 | | \| | *Term* |
| 5 | *Term* | → | *Term* * *Factor* |
| 6 | | \| | *Term* / *Factor* |
| 7 | | \| | *Factor* |
| 8 | *Factor* | → | number |
| 9 | | \| | id |
| 10 | | \| | ( *Expr* ) |

*The expression grammar*

| *Prod'n.* | *Sentential Form* | *Handle* |
|---|---|---|
| — | *Goal* | — |
| 1 | *Expr* | 1,1 |
| 3 | *Expr* - *Term* | 3,3 |
| 5 | *Expr* - *Term* * *Factor* | 5,5 |
| 9 | *Expr* - *Term* * <id,y> | 9,5 |
| 7 | *Expr* - *Factor* * <id,y> | 7,3 |
| 8 | *Expr* - <num,2> * <id,y> | 8,3 |
| 4 | *Term* - <num,2> * <id,y> | 4,1 |
| 7 | *Factor* - <num,2> * <id,y> | 7,1 |
| 9 | <id,x> - <num,2> * <id,y> | 9,1 |

*Handles for rightmost derivation of* x - 2 * y

# Handle-pruning, Bottom-up Parsers

The process of discovering a handle & reducing it to the appropriate left-hand side is called *handle pruning*

Handle pruning forms the basis for a bottom-up parsing method

To reconstruct a rightmost derivation

$S \Rightarrow \gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_{n-1} \Rightarrow \gamma_n \Rightarrow w$

Apply the following simple algorithm

*for i ← n to 1 by -1*

*Find the handle* $\langle A_i \rightarrow \beta_i , \mathbf{k}_i \rangle$ in $\gamma_i$

*Replace* $\beta_i$ *with* $A_i$ *to generate* $\gamma_{i-1}$

This takes *2n* steps

# Handle-pruning, Bottom-up Parsers

One implementation technique is the *shift-reduce parser*

```
push INVALID
token ← next_token( )
repeat until (top of stack = Goal and token = EOF)
    if the top of the stack is a handle A→β
        then     // reduce β to A
            pop |β| symbols off the stack
            push A onto the stack
        else if (token ≠ EOF)
            then // shift
                push token
                token ← next_token( )
            else     // need to shift but out of input
            report an error
```

**How do errors show up?**

- **failure to find a handle**
- **hitting EOF & needing to shift (final else clause)**

**Either generates an error**

**Figure 3.7 in EAC**

# Back to x - 2 * y

| Stack | Input | Handle | Action |
|---|---|---|---|
| $ | id – num * id | *none* | shift |
| $ id | – num * id | | |

1. Shift until the top of the stack is the right end of a handle
2. Find the left end of the handle & reduce

# Back to x - 2 * y

| Stack | Input | Handle | Action |
|---|---|---|---|
| $ | id – num * id | *none* | shift |
| $ id | – num * id | 9,1 | red. 9 |
| $ *Factor* | – num * id | 7,1 | red. 7 |
| $ *Term* | – num * id | 4,1 | red. 4 |
| $ *Expr* | – num * id | | |

1. Shift until the top of the stack is the right end of a handle
2. Find the left end of the handle & reduce

# Back to x – 2 * y

| Stack | Input | Handle | Action |
|---|---|---|---|
| $ | id – num * id | *none* | shift |
| $ id | – num * id | 9,1 | red. 9 |
| $ *Factor* | – num * id | 7,1 | red. 7 |
| $ *Term* | – num * id | 4,1 | red. 4 |
| $ *Expr* | – num * id | *none* | shift |
| $ *Expr* – | num * id | *none* | shift |
| $ *Expr* – num | * id | | |

1. Shift until the top of the stack is the right end of a handle
2. Find the left end of the handle & reduce

# Back to x - 2 * y

| Stack | Input | Handle | Action |
|---|---|---|---|
| $ | id – num * id | *none* | shift |
| $ id | – num * id | 9,1 | red. 9 |
| $ *Factor* | – num * id | 7,1 | red. 7 |
| $ *Term* | – num * id | 4,1 | red. 4 |
| $ *Expr* | – num * id | *none* | shift |
| $ *Expr* – | num * id | *none* | shift |
| $ *Expr* – num | * id | 8,3 | red. 8 |
| $ *Expr* – *Factor* | * id | 7,3 | red. 7 |
| $ *Expr* – *Term* | * id | | |

1. Shift until the top of the stack is the right end of a handle
2. Find the left end of the handle & reduce

# Back to x - 2 * y

| Stack | Input | Handle | Action |
|---|---|---|---|
| $ | id – num * id | *none* | shift |
| $ id | – num * id | 9,1 | red. 9 |
| $ *Factor* | – num * id | 7,1 | red. 7 |
| $ *Term* | – num * id | 4,1 | red. 4 |
| $ *Expr* | – num * id | *none* | shift |
| $ *Expr* – | num * id | *none* | shift |
| $ *Expr* – num | * id | 8,3 | red. 8 |
| $ *Expr* – *Factor* | * id | 7,3 | red. 7 |
| $ *Expr* – *Term* | * id | *none* | shift |
| $ *Expr* – *Term* * | id | *none* | shift |
| $ *Expr* – *Term* * id | | | |

1. Shift until the top of the stack is the right end of a handle
2. Find the left end of the handle & reduce

# Back to x – 2 * y

| Stack | Input | Handle | Action |
|---|---|---|---|
| $ | id – num * id | *none* | shift |
| $ id | – num * id | 9,1 | red. 9 |
| $ *Factor* | – num * id | 7,1 | red. 7 |
| $ *Term* | – num * id | 4,1 | red. 4 |
| $ *Expr* | – num * id | *none* | shift |
| $ *Expr* – | num * id | *none* | shift |
| $ *Expr* – num | * id | 8,3 | red. 8 |
| $ *Expr* – *Factor* | * id | 7,3 | red. 7 |
| $ *Expr* – *Term* | * id | *none* | shift |
| $ *Expr* – *Term* * | id | *none* | shift |
| $ *Expr* – *Term* * id | | 9,5 | red. 9 |
| $ *Expr* – *Term* * *Factor* | | 5,5 | red. 5 |
| $ *Expr* – *Term* | | 3,3 | red. 3 |
| $ *Expr* | | 1,1 | red. 1 |
| $ *Goal* | | *none* | accept |

5 shifts +
9 reduces +
1 accept

1. Shift until the top of the stack is the right end of a handle
2. Find the left end of the handle & reduce

# Back to x – 2 * y

| Stack | Input | Handle | Action |
|---|---|---|---|
| $ | id – num * id | *none* | shift |
| $ id | – num * id | 9,1 | red. 9 |
| $ *Factor* | – num * id | 7,1 | red. 7 |
| $ *Term* | – num * id | 4,1 | red. 4 |
| $ *Expr* | – num * id | *none* | shift |
| $ *Expr* – | num * id | *none* | shift |
| $ *Expr* – num | * id | 8,3 | red. 8 |
| $ *Expr* – *Factor* | * id | 7,3 | red. 7 |
| $ *Expr* – *Term* | * id | *none* | shift |
| $ *Expr* – *Term* * | id | *none* | shift |
| $ *Expr* – *Term* * id | | 9,5 | red. 9 |
| $ *Expr* – *Term* * *Factor* | | 5,5 | red. 5 |
| $ *Expr* – *Term* | | 3,3 | red. 3 |
| $ *Expr* | | 1,1 | red. 1 |
| $ *Goal* | | *none* | accept |

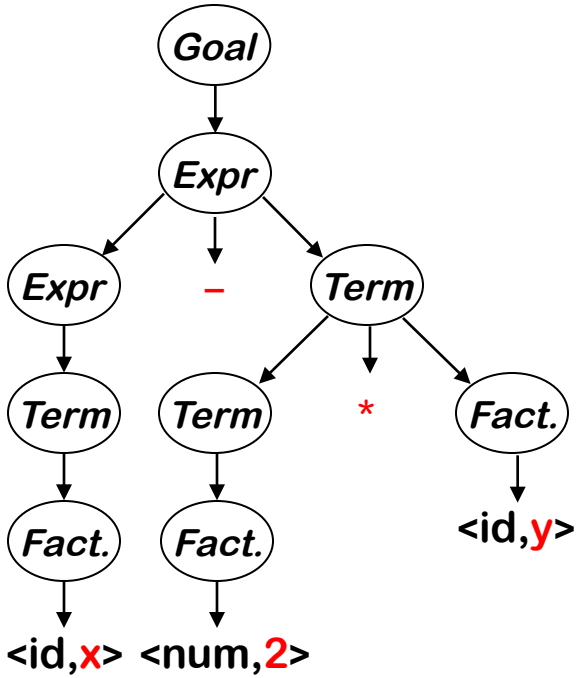
shift here

reduce here

1. Shift until the top of the stack is the right end of a handle
2. Find the left end of the handle & reduce

# Back to x - 2 * y

| Stack | Input | Action |
|---|---|---|
| $ | id – num * id | shift |
| $ id | – num * id | red. 9 |
| $ *Factor* | – num * id | red. 7 |
| $ *Term* | – num * id | red. 4 |
| $ *Expr* | – num * id | shift |
| $ *Expr* – | num * id | shift |
| $ *Expr* – num | * id | red. 8 |
| $ *Expr* – *Factor* | * id | red. 7 |
| $ *Expr* – *Term* | * id | shift |
| $ *Expr* – *Term* * | id | shift |
| $ *Expr* – *Term* * id | | red. 9 |
| $ *Expr* – *Term* * *Factor* | | red. 5 |
| $ *Expr* – *Term* | | red. 3 |
| $ *Expr* | | red. 1 |
| $ *Goal* | | accept |



# Shift-reduce Parsing

*Shift reduce parsers are easily built and easily understood*

A shift-reduce parser has just four actions

- *Shift* — next word is shifted onto the stack
- *Reduce* — right end of handle is at top of stack
  - Locate left end of handle within the stack
  - Pop handle off stack & push appropriate *lhs*
- *Accept* — stop parsing & report success
- *Error* — call an error reporting/recovery routine

*Accept & Error* are simple

*Shift* is just a push and a call to the scanner

*Reduce* takes |*rhs*| pops & 1 push

*If handle-finding requires state, put it in the stack ⇒ 2x work*

Handle finding is key
- handle is on stack
- finite set of handles

⇒ use a DFA !

# LR(1) Parsers

- LR(1) parsers are table-driven, shift-reduce parsers that use a limited right context (1 token) for handle recognition
- LR(1) parsers recognize languages that have an LR(1) grammar

*Informal definition:*

A grammar is LR(1) if, given a rightmost derivation

$$S \Rightarrow \gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_{n-1} \Rightarrow \gamma_n \Rightarrow \textit{sentence}$$

We can

1. *isolate the handle of each right-sentential form* $\gamma_i$, and
2. *determine the production by which to reduce,*

by scanning $\gamma_i$ from left-to-right, going at most 1 symbol beyond the right end of the handle of $\gamma_i$

# Next class

## More Syntax Analysis (bottom-up)

Read EaC: Chapter 3.4