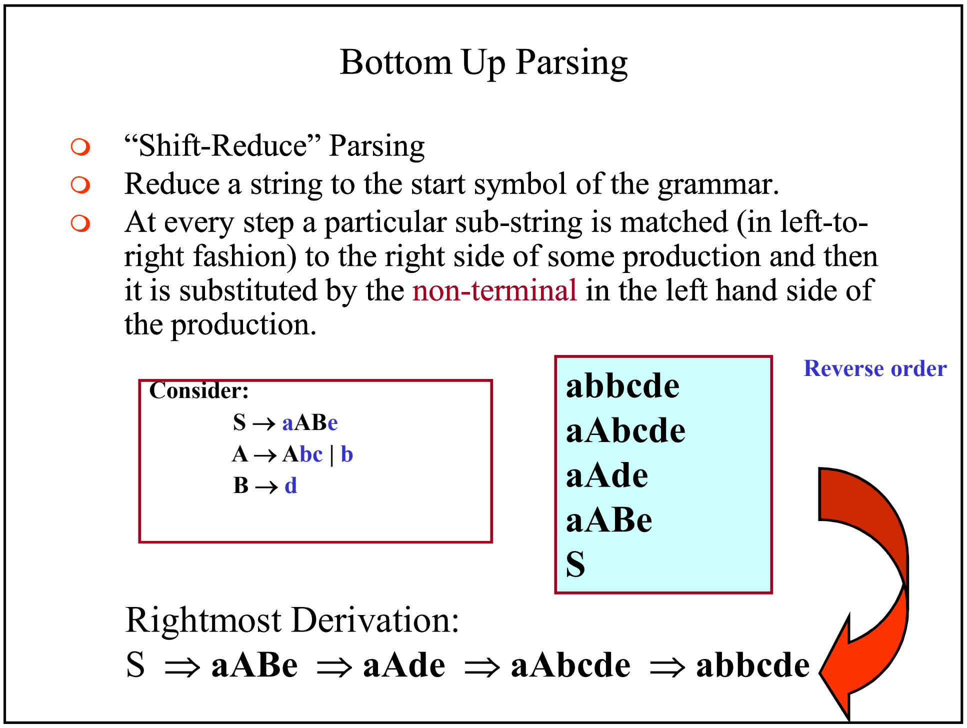
Chapter 4
Syntax Analysis

Bottom-up Parsing

A general style of bottom-up syntax analysis, known as shift-reduce parsing.

Two types of bottom-up parsing:

1. Operator-Precedence parsing
2. LR parsing



Bottom Up Parsing

- "Shift-Reduce" Parsing
- Reduce a string to the start symbol of the grammar.
- At every step a particular sub-string is matched (in left-to-right fashion) to the right side of some production and then it is substituted by the non-terminal in the left hand side of the production.

Consider:

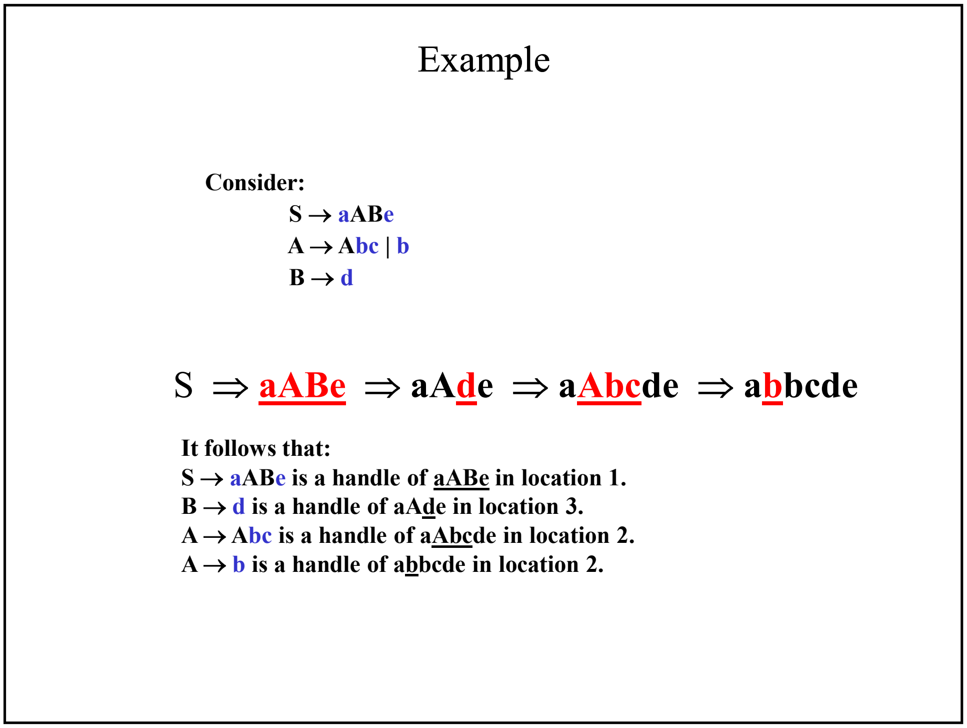
S → aABe
A → Abc | b
B → d

abbcde
aAbcde
aAde
aABe
S

Reverse order

Rightmost Derivation:

$S \Rightarrow \mathbf{aABe} \Rightarrow \mathbf{aAde} \Rightarrow \mathbf{aAbcde} \Rightarrow \mathbf{abbcde}$



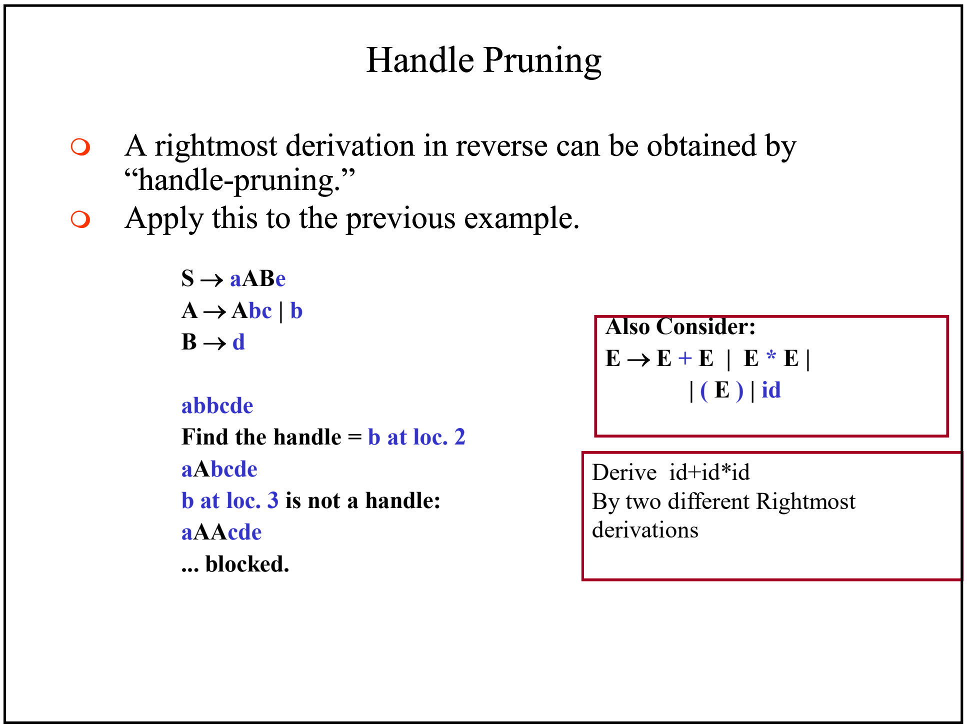
Example

Consider:

S → aABe
A → Abc | b
B → d

$S \Rightarrow \underline{\mathbf{aABe}} \Rightarrow \mathbf{aA}\underline{\mathbf{d}}\mathbf{e} \Rightarrow \mathbf{a}\underline{\mathbf{Abc}}\mathbf{de} \Rightarrow \mathbf{a}\underline{\mathbf{b}}\mathbf{bcde}$

It follows that:
S → aABe is a handle of $\underline{\text{aABe}}$ in location 1.
B → d is a handle of aA$\underline{\text{d}}$e in location 3.
A → Abc is a handle of a$\underline{\text{Abc}}$de in location 2.
A → b is a handle of a$\underline{\text{b}}$bcde in location 2.



Handle Pruning

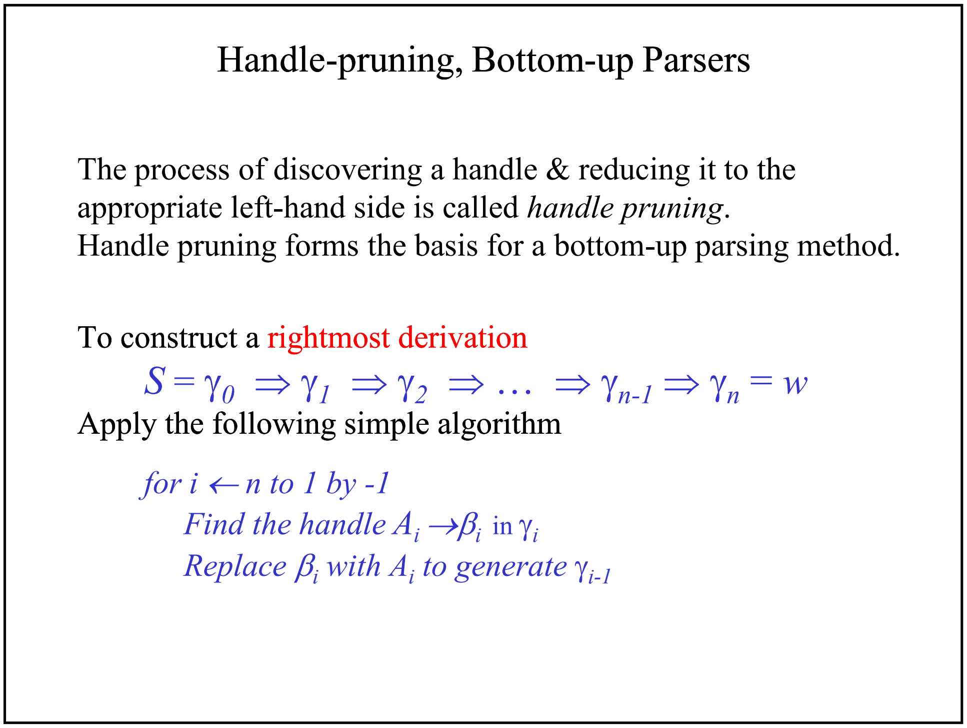
- A rightmost derivation in reverse can be obtained by "handle-pruning."
- Apply this to the previous example.

S → aABe
A → Abc | b
B → d

abbcde
Find the handle = b at loc. 2
aAbcde
b at loc. 3 is not a handle:
aAAcde
... blocked.

Also Consider:
E → E + E | E * E |
| (E) | id

Derive id+id*id
By two different Rightmost derivations



Handle-pruning, Bottom-up Parsers

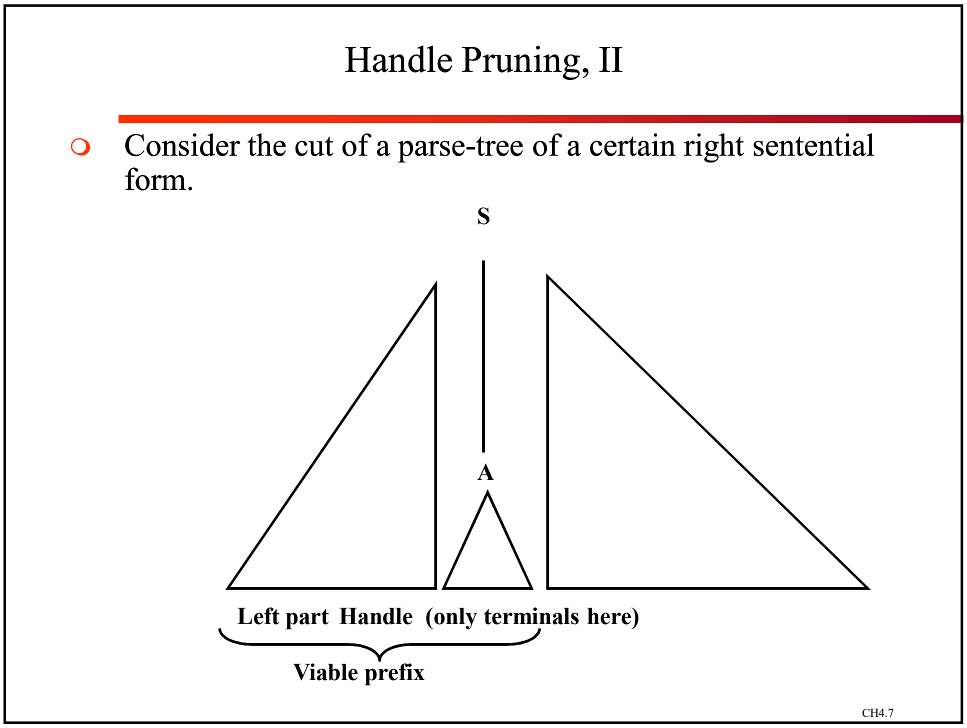
The process of discovering a handle & reducing it to the appropriate left-hand side is called *handle pruning*.
Handle pruning forms the basis for a bottom-up parsing method.

To construct a rightmost derivation

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

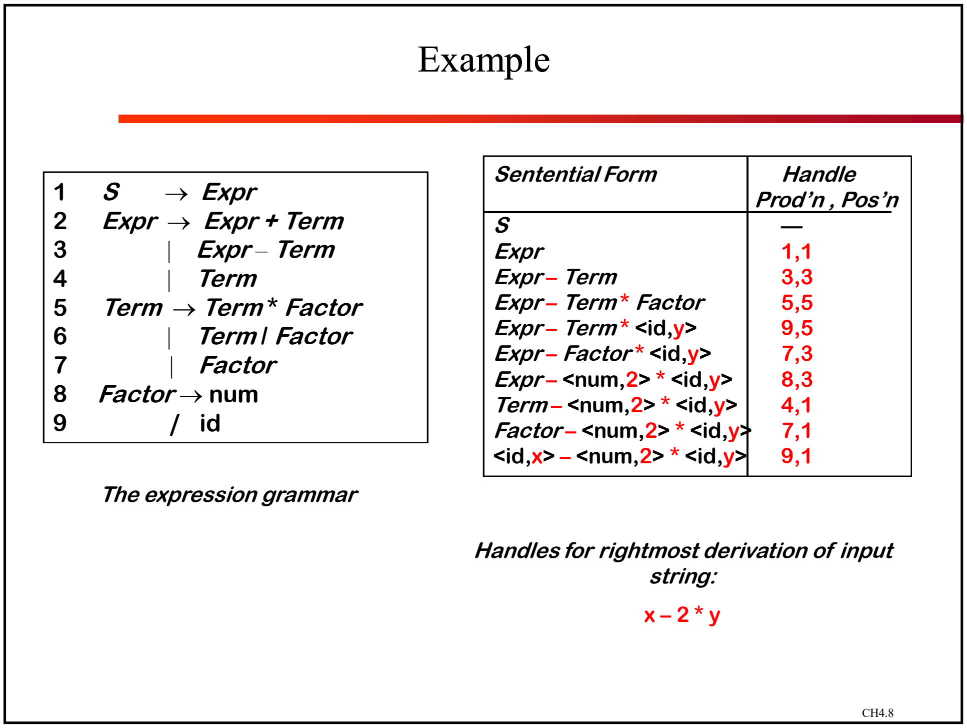
Apply the following simple algorithm

```
for $i \leftarrow n$ to 1 by -1
    Find the handle $A_i \rightarrow \beta_i$ in $\gamma_i$
    Replace $\beta_i$ with $A_i$ to generate $\gamma_{i-1}$
```



Handle Pruning, II

- Consider the cut of a parse-tree of a certain right sentential form.



Example

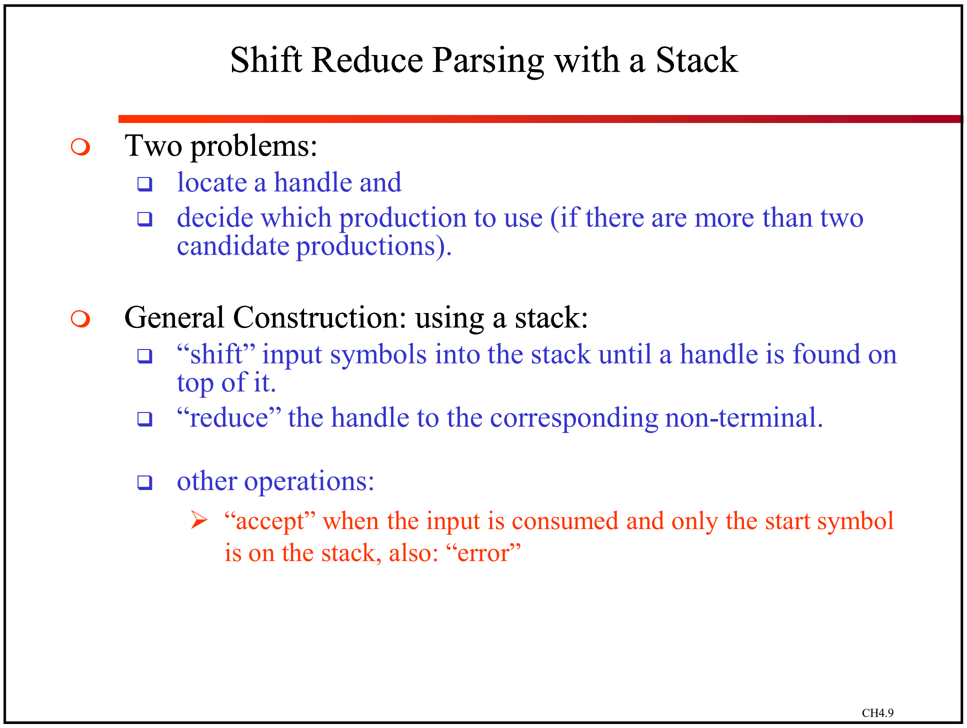
1	*S*	→	*Expr*
2	*Expr*	→	*Expr + Term*
3		|	*Expr – Term*
4		|	*Term*
5	*Term*	→	*Term * Factor*
6		|	*Term / Factor*
7		|	*Factor*
8	*Factor*	→	num
9		/	id

The expression grammar

Sentential Form	*Handle Prod'n , Pos'n*
S	—
Expr	1,1
Expr – Term	3,3
*Expr – Term * Factor*	5,5
*Expr – Term ** <id,y>	9,5
*Expr – Factor ** <id,y>	7,3
Expr – <num,2> * <id,y>	8,3
Term – <num,2> * <id,y>	4,1
Factor – <num,2> * <id,y>	7,1
<id,x> – <num,2> * <id,y>	9,1

Handles for rightmost derivation of input string:

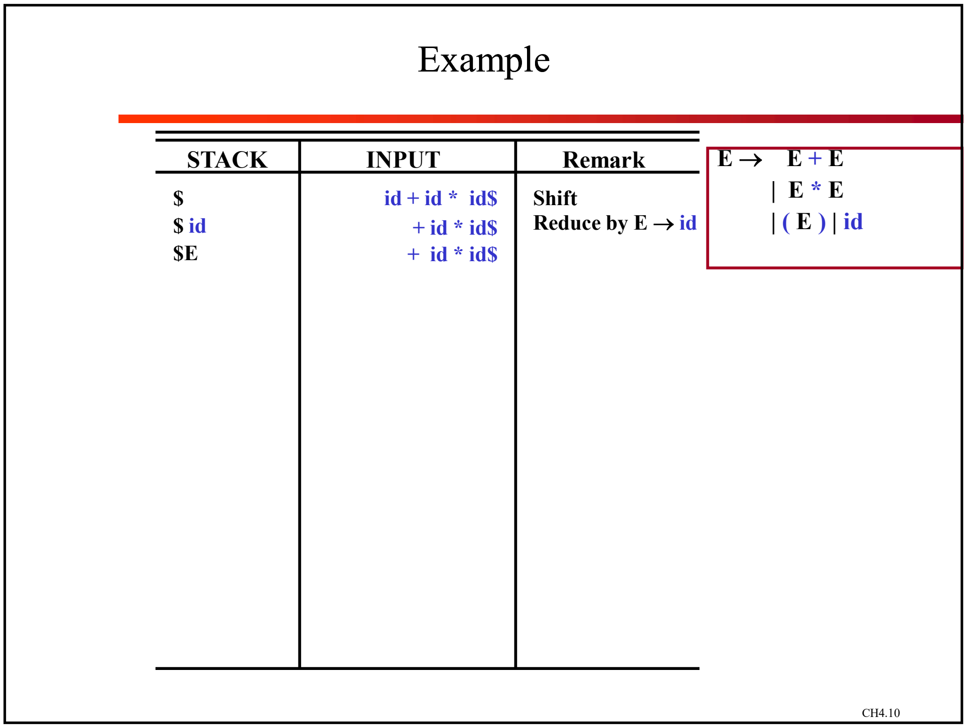
x – 2 * y



Shift Reduce Parsing with a Stack

- Two problems:
  - locate a handle and
  - decide which production to use (if there are more than two candidate productions).
- General Construction: using a stack:
  - “shift” input symbols into the stack until a handle is found on top of it.
  - “reduce” the handle to the corresponding non-terminal.
  - other operations:
    - “accept” when the input is consumed and only the start symbol is on the stack, also: “error”

CH4.9

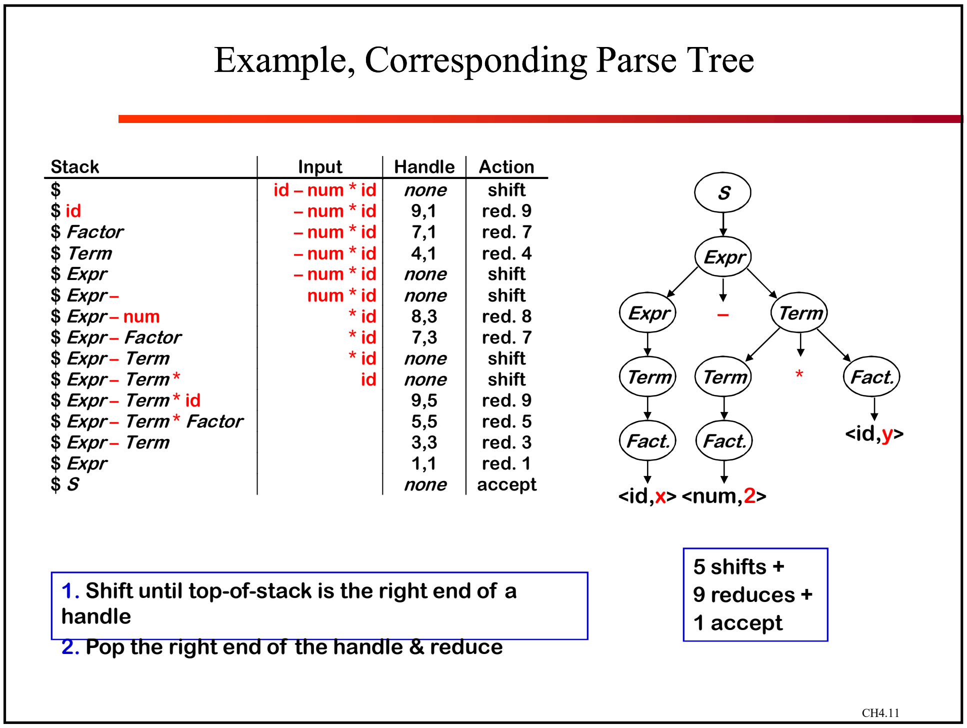


Example

STACK	INPUT	Remark
$	id + id * id$	Shift
$ id	+ id * id$	Reduce by E → id
$E	+ id * id$	

E → E + E
| E * E
| (E) | id

CH4.10



Example, Corresponding Parse Tree

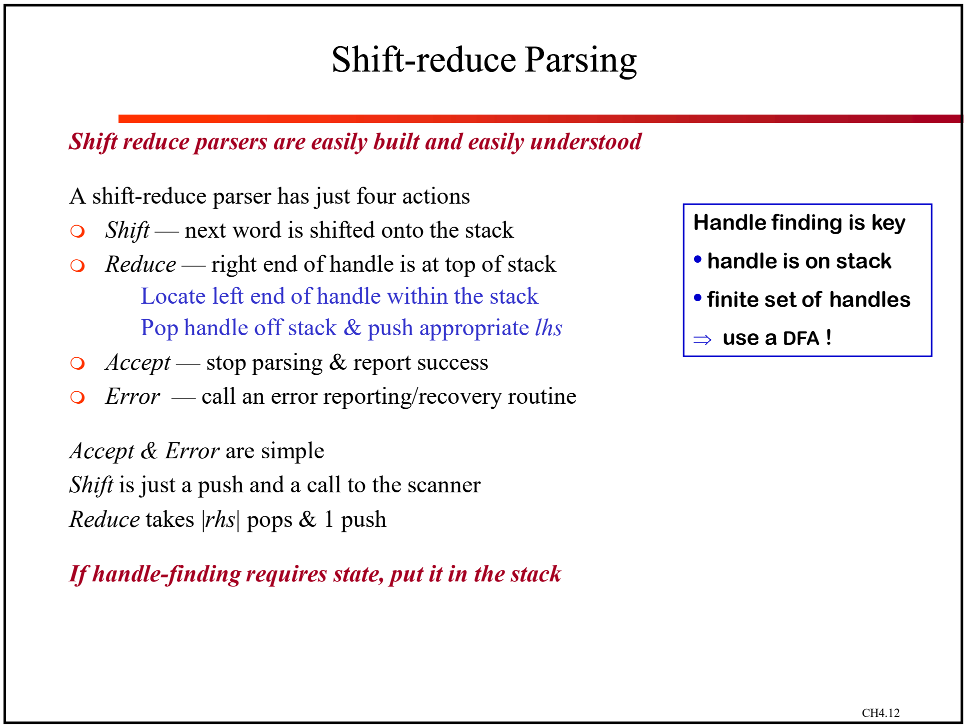
Stack	Input	Handle	Action
$	id – num * id	*none*	shift
$ id	– num * id	9,1	red. 9
$ *Factor*	– num * id	7,1	red. 7
$ *Term*	– num * id	4,1	red. 4
$ *Expr*	– num * id	*none*	shift
$ *Expr* –	num * id	*none*	shift
$ *Expr* – num	* id	8,3	red. 8
$ *Expr* – *Factor*	* id	7,3	red. 7
$ *Expr* – *Term*	* id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

S
Expr
Expr
–
Term
Term
Term
*
Fact.
Fact.
Fact.
<id,y>
<id,x> <num,2>

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

5 shifts +
9 reduces +
1 accept

CH4.11



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

Handle finding is key

- **handle is on stack**
- **finite set of handles**

⇒ **use a DFA !**

CH4.12

More on Shift-Reduce Parsing

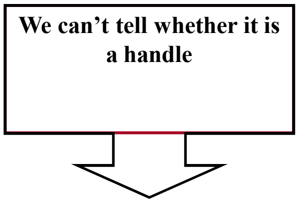
Viable prefixes:

The set of prefixes of a right sentential form that can appear on the stack of a Shift-Reduce parser is called Viable prefixes.

Conflicts

"shift/reduce" or "reduce/reduce"

Example: ***stmt*** → **if *expr* then *stmt***
| **if *expr* then *stmt* else *stmt***
| **other (any other statement)**



We can't tell whether it is a handle

Stack **Input**

if … then stmt **else …**

Shift/ Reduce Conflict

CH4.13