Bottom Up Parsers

Bottom-Up Parsing

- Construct parse tree bottom-up, from leaves (tokens) to root *S*

- Always construct right-most derivation, but in reverse order

- Algorithms:

  shift-reduce

  LR parsing (LR(0), SLR, LR(k), LALR(k), ...)

Shift-Reduce Parsing

Shift-Reduce: look for substrings that match lhs of a production and *reduce* them by applying the production in reverse.

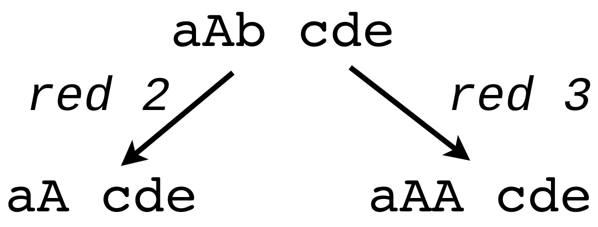
Grammar

1) S → aAcBe
2) A → Ab
3) A → b
4) B → d

Parsing as derivation in reverse

abbcde	*shift*	aAc de	*shift*
a bbcde	*shift*	aAc$\underline{\text{d}}$ e	*red 4*
a$\underline{\text{b}}$ bcde	*red 3*	aAcB e	*shift*
aA bcde	*shift*	$\underline{\text{aAcBe}}$	*red 1*
a$\underline{\text{Ab}}$ cde	*red 2*	S	
aA cde	*shift*		

Ambiguity!



aAb cde

red 2

red 3

aA cde

aAA cde

The underlined substrings are called **handles.**

Some terminology

A rightmost derivation is one where the rightmost nonterminal is replaced at each step. Write $\alpha \Rightarrow \beta$ for a rightmost derivation step.

$$\texttt{S} \Rightarrow^* \alpha\texttt{A}w \Rightarrow \alpha\beta w \Rightarrow^* v \qquad \text{(where } w,v \in \text{T}^* \text{)}$$

Here β is a *handle* -- a substring that is the *lhs* of a production in a rightmost derivation (for v). If $\texttt{S} \Rightarrow \gamma$, then g is called a *right sentential form*.

The handles are the substrings that should be reduced in a shift-reduce parse.

$\texttt{a}\underline{\texttt{Ab}}\ \texttt{cde} \Leftarrow_2 \texttt{aA cde} \Leftarrow^* \texttt{S}$

a handle

$\texttt{aA}\underline{\texttt{b}}\ \texttt{cde} \Leftarrow_3 \texttt{aAA cde}$

not a handle

$\texttt{aAcde}$ is a right sentential form, while $\texttt{aAAcde}$ is not.

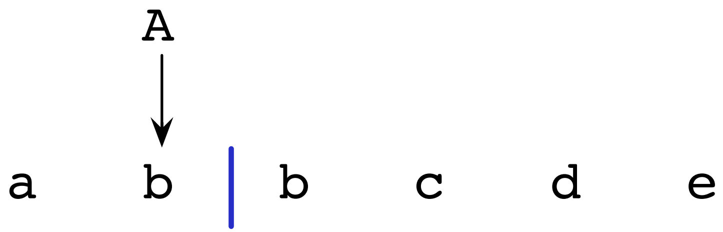
Shift-Reduce Parsing Illustrated

```
1) S → aAcBe
2) A → Ab
3) A → b
4) B → d
```

```
a | b   b   c   d   e
```

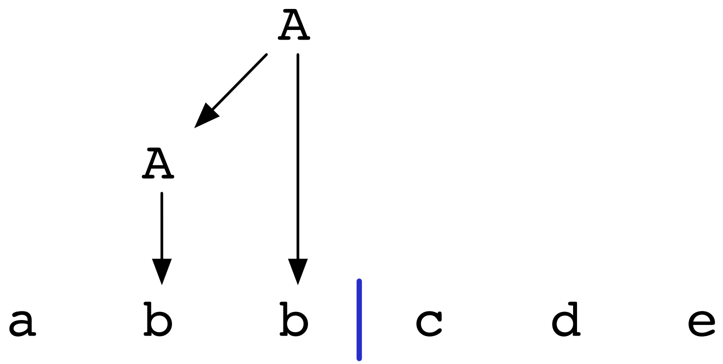
Shift-Reduce Parsing Illustrated

1) S → aAcBe
2) A → Ab
3) A → b
4) B → d



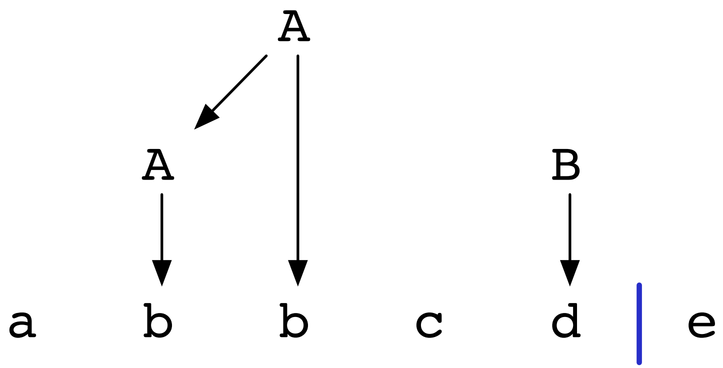
Shift-Reduce Parsing Illustrated

1) S → aAcBe
2) A → Ab
3) A → b
4) B → d



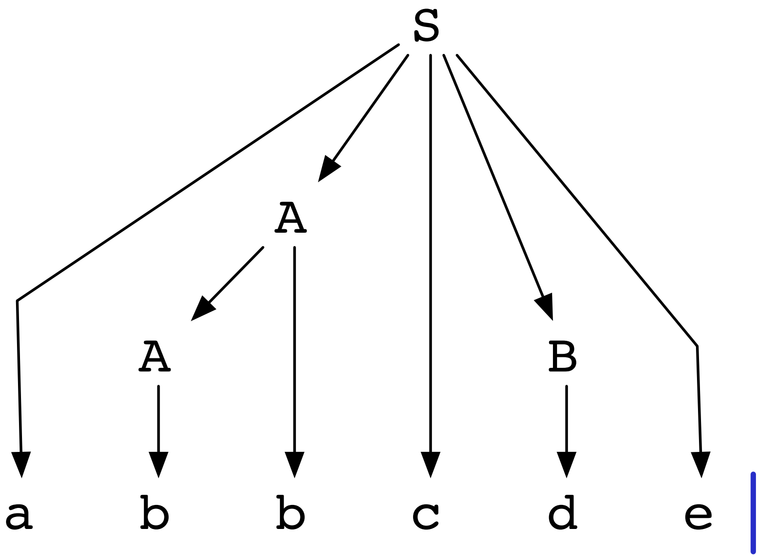
Shift-Reduce Parsing Illustrated

1) S → aAcBe
2) A → Ab
3) A → b
4) B → d



Shift-Reduce Parsing Illustrated

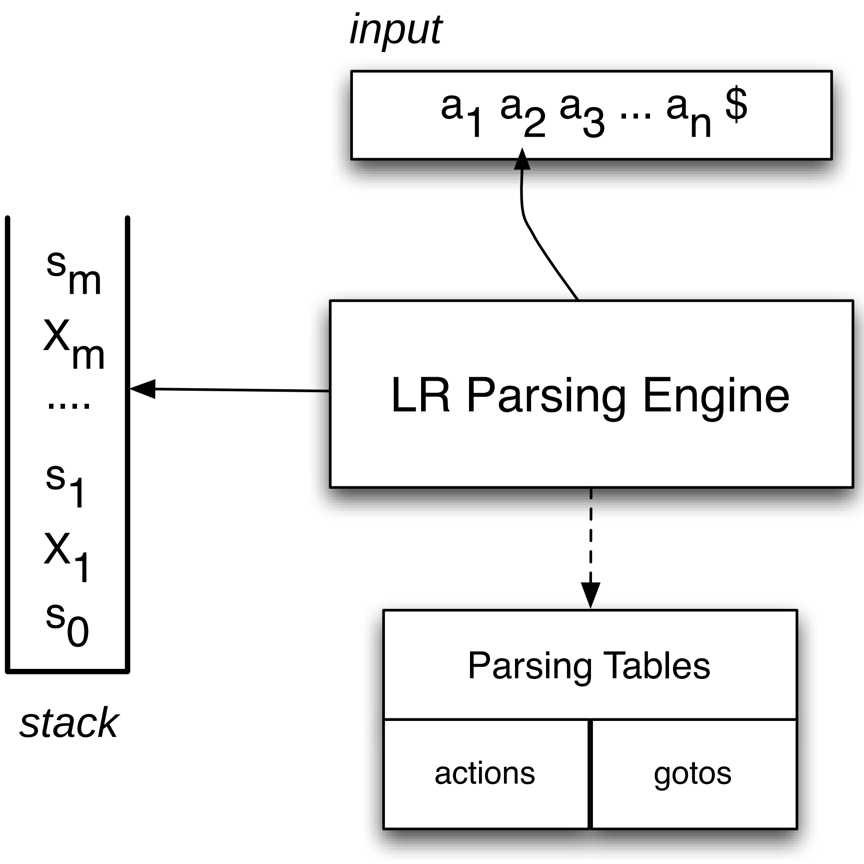
1) S → aAcBe
2) A → Ab
3) A → b
4) B → d



LR parsers

Problem: find the handles (and corresponding productions)

Solution: define a DFA that determines when to shift, when to reduce, when to accept, when to signal error



Advantages of LR Parsers

- *LR parsers can handle virtually all programming language constructs expressible in context free grammars*

- *LR parsing is most general nonbacktracking shift-reduce parsing method, yet is efficiently implementable*

- *Class of grammars parsed by LR parsers is larger than that parsed by predictive parsers*

- *LS parsers can detect syntactic errors as soon as possible, given left to right scan of input*

Building LR Parser Tables

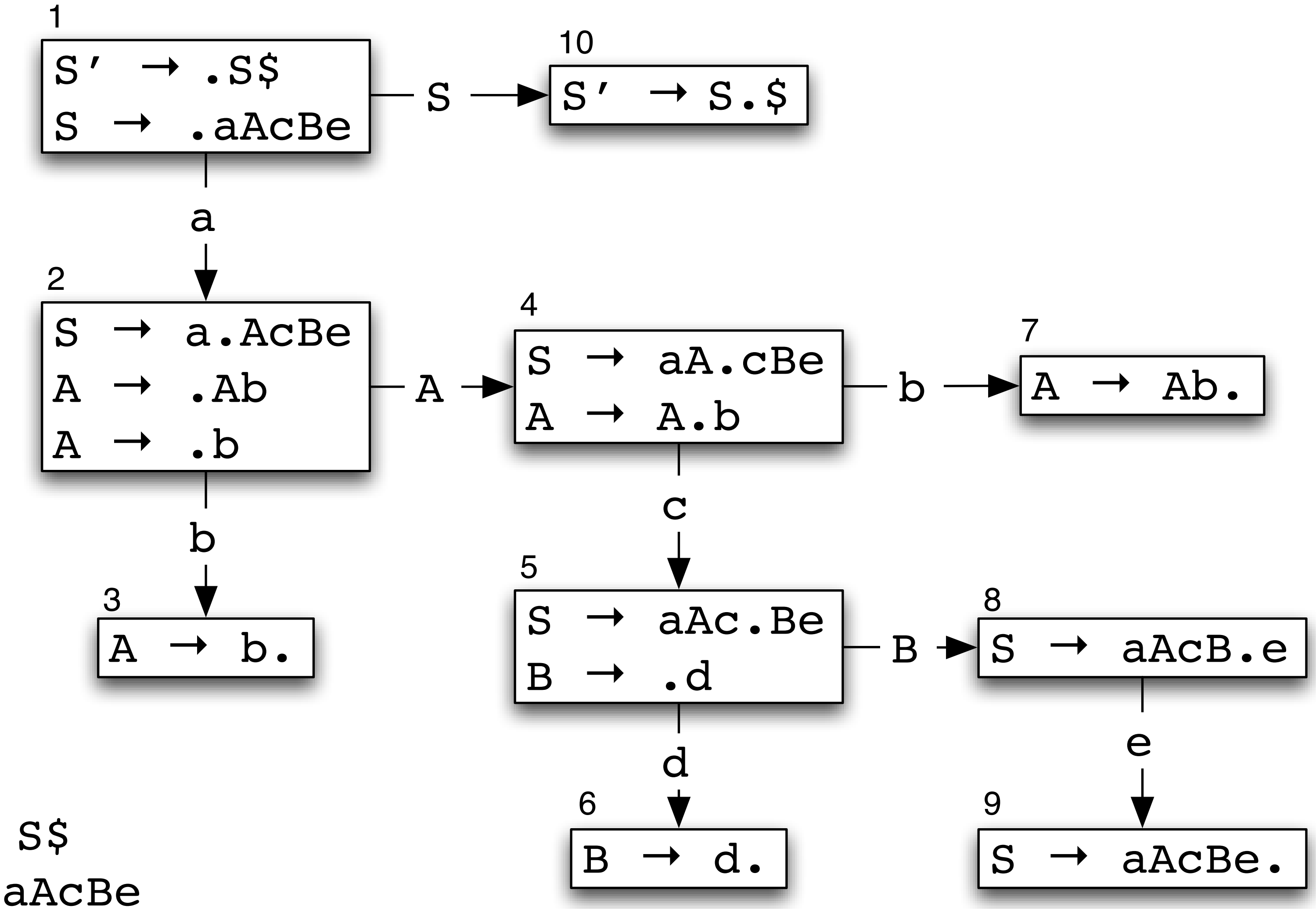
Grammar

```
0) S' → S$
1) S → aAcBe
2) A → Ab
3) A → b
4) B → d
```

Items

```
S → .aAcBe      A → .Ab     B → .d
S → a.AcBe      A → A.b     B → d.
S → aA.cBe      A → Ab.
S → aAc.Be
S → aAcB.e      A → .b      S' → .S$
S → aAcBe.      A → b.      S' → S.$
```

Building LR Parser DFA



Grammar

0) S′ → S$
1) S → aAcBe
2) A → Ab
3) A → b
4) B → d

LR Parser Table

	a	b	c	d	e	$	A	B	S
1	s2								10
2		s3					4		
3	r3	r3	r3	r3	r3	r3			
4		s7	s5						
5				s6				8	
6	r4	r4	r4	r4	r4	r4			
7	r2	r2	r2	r2	r2	r2			
8					s9				
9	r1	r1	r1	r1	r1	r1			
10						!			