CS 4300: Compiler Theory

Chapter 4
Syntax Analysis

Dr. Xuejun Liang

Outlines (Sections)

1. Introduction
2. Context-Free Grammars
3. Writing a Grammar
4. Top-Down Parsing
5. Bottom-Up Parsing
6. Introduction to LR Parsing: Simple LR
7. More Powerful LR Parsers
8. Using Ambiguous Grammars
9. Parser Generators

Quick Review of Last Lecture

- Top-Down Parsing
  - Using FIRST and FOLLOW in a Recursive-Descent Parser
  - Non-Recursive Predictive Parsing: Table-Driven Parsing
    - Constructing an LL(1) Predictive Parsing Table
    - Predictive Parsing (Driver) Program
    - Panic Mode Recovery
    - Phrase-Level Recovery

- Bottom-Up Parsing
  - Shift-Reduce Parsing
  - Handles

Shift-Reduce Parsing

Grammar:

$S \rightarrow \mathbf{a}\ A\ B\ \mathbf{e}$

$A \rightarrow A\ \mathbf{b}\ \mathbf{c} \mid \mathbf{b}$

$B \rightarrow \mathbf{d}$

These match production's right-hand sides

Reducing a sentence:

$\mathbf{a}\ \underline{\mathbf{b}}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ \underline{A\ \mathbf{b}\ \mathbf{c}}\ \mathbf{d}\ \mathbf{e}$

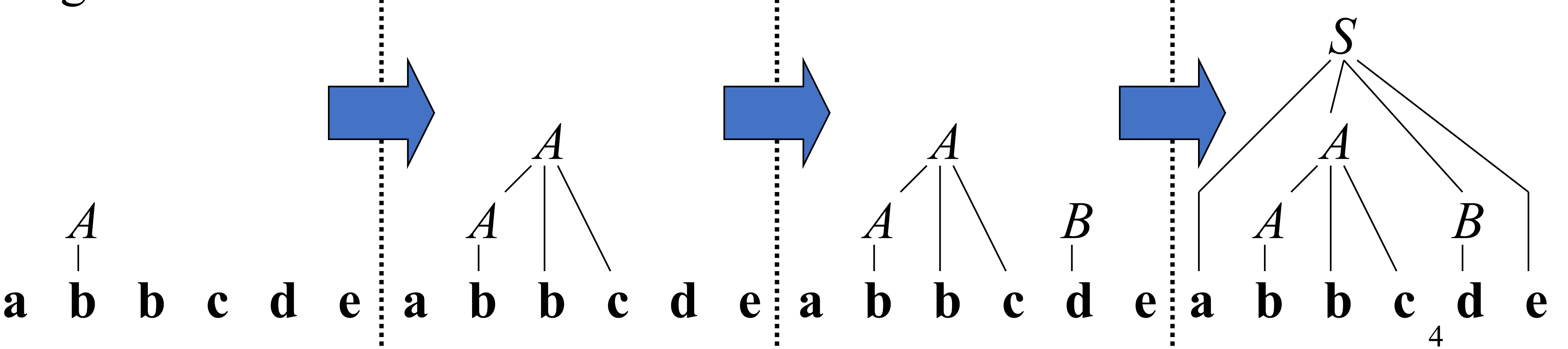
$\mathbf{a}\ A\ \underline{\mathbf{d}}\ \mathbf{e}$

$\underline{\mathbf{a}\ A\ B\ \mathbf{e}}$

S

Shift-reduce corresponds to the reverse of a rightmost derivation:

$$S \Rightarrow_{rm} \mathbf{a}\ A\ B\ \mathbf{e}$$
$$\Rightarrow_{rm} \mathbf{a}\ A\ \mathbf{d}\ \mathbf{e}$$
$$\Rightarrow_{rm} \mathbf{a}\ A\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$$
$$\Rightarrow_{rm} \mathbf{a}\ \mathbf{b}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$$



Stack Implementation of Shift-Reduce Parsing

Grammar:

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow$ **(** E **)**

$E \rightarrow$ **id**

Stack	Input	Action
$	**id+id*id$**	shift
$<u>id</u>	**+id*id$**	reduce $E \rightarrow$ **id**
**$**$E$	**+id*id$**	shift
$**$E$+**	**id*id$**	shift
$**$E$+<u>id</u>**	***id$**	reduce $E \rightarrow$ **id**
**$**$E$**+**$E$	***id$**	shift (or reduce?)
$**$E$**+**$E$***	**id$**	shift
$**$E$**+**$E$*<u>id</u>**	**$**	reduce $E \rightarrow$ **id**
**$**$E$**+**$\underline{E*E}$	**$**	reduce $E \rightarrow E * E$
**$**$\underline{E+E}$	**$**	reduce $E \rightarrow E + E$
**$**$E$	**$**	accept

Found handles to reduce

How to resolve conflicts?

Conflicts

- *Shift-reduce* and *reduce-reduce* conflicts are caused by
  - The limitations of the LR parsing method (even when the grammar is unambiguous)
  - Ambiguity of the grammar

Shift-Reduce Parsing: Shift-Reduce Conflicts

Stack	Input	Action
$…	…**$**	…
$…<u>**if** *E* **then** *S*</u>	**else**…**$**	shift or reduce?

Ambiguous grammar:

S → **if** *E* **then** *S*
  | **if** *E* **then** *S* **else** *S*
  | **other**

Resolve in favor of shift, so **else** matches closest **if**

Shift-Reduce Parsing: Reduce-Reduce Conflicts

Grammar:

$C \rightarrow A\ B$

$A \rightarrow \mathbf{a}$

$B \rightarrow \mathbf{a}$

Stack	Input	Action
$	**aa$**	shift
$<u>a</u>	**a$**	reduce $A \rightarrow \mathbf{a}$ <u>or</u> $B \rightarrow \mathbf{a}$?

Resolve in favor of reducing $A \rightarrow \mathbf{a}$, otherwise we're stuck!

6. LR Parsing: Simple LR

- LR(k) parsing
  - From left to right scanning of the input
  - Rightmost derivation in reverse
  - k lookahead symbols, only consider k=0, or 1
- Why LR Parsers
  - Can recognize virtually all programming language constructs
  - the most general nonbacktracking shift-reduce parsing method
  - Can detect a syntactic error as soon as possible
  - Powerful than LL parsing methods

LR(0) Items of a Grammar

- An *LR*(0) *item* of a grammar G is a production of G with a • at some position of the right-hand side
- Thus, a production

  $$A \to X\ Y\ Z$$

  has four items:

  $$[A \to \bullet\ X\ Y\ Z]$$
  $$[A \to X \bullet\ Y\ Z]$$
  $$[A \to X\ Y \bullet\ Z]$$
  $$[A \to X\ Y\ Z\ \bullet]$$

- Note that production $A \to \varepsilon$ has one item $[A \to \bullet]$

The *closure* Operation for LR(0) Items

1. Start with *closure*(*I*) = *I*
2. If $[A \rightarrow \alpha \bullet B\beta] \in$ *closure*(*I*) then for each production $B \rightarrow \gamma$ in the grammar, add the item $[B \rightarrow \bullet\gamma]$ to *closure*(*I*) if not already in *closure*(*I*)
3. Repeat 2 until no new items can be added

The *closure* Operation Example

$closure(\{[E' \to \bullet E]\}) =$

$\{\ [E' \to \bullet\ E]\ \}$

$\{\ [E' \to \bullet\ E]$
$[E \to \bullet\ E + T]$
$[E \to \bullet\ T]\ \}$

Add $[E \to \bullet\gamma]$

$\{\ [E' \to \bullet\ E]$
$[E \to \bullet\ E + T]$
$[E \to \bullet\ T]$
$[T \to \bullet\ T * F]$
$[T \to \bullet\ F]\ \}$

Add $[T \to \bullet\gamma]$

$\{\ [E' \to \bullet\ E]$
$[E \to \bullet\ E + T]$
$[E \to \bullet\ T]$
$[T \to \bullet\ T * F]$
$[T \to \bullet\ F]$
$[F \to \bullet\ \mathbf{(}\ E\ \mathbf{)}]$
$[F \to \bullet\ \mathbf{id}]\ \}$

Add $[F \to \bullet\gamma]$

Grammar:

$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to \mathbf{(}\ E\ \mathbf{)}$
$F \to \mathbf{id}$

The ***goto*** Operation for LR(0) Items

1. For each item $[A \to \alpha \bullet X \beta] \in I$, add the set of items ***closure***($\{[A \to \alpha X \bullet \beta]\}$) to ***goto***($I$,$X$) if not already there
2. Repeat step 1 until no more items can be added to ***goto***(I,X)

- Intuitively, the ***goto*** function is used to define the transitions in the LR(0) automaton for a grammar.
- The states of the automaton correspond to sets of items, and ***goto***(I, X) specifies the transition from the state for I under input X.

The *goto* Operation Example 1

Suppose

$$I = \{\ [E' \to \bullet\ E]$$
$$[E \to \bullet\ E + T]$$
$$[E \to \bullet\ T]$$
$$[T \to \bullet\ T * F]$$
$$[T \to \bullet\ F]$$
$$[F \to \bullet\ \mathbf{(}\ E\ \mathbf{)}]$$
$$[F \to \bullet\ \mathbf{id}]\ \}$$

Then $goto(I, E)$

$$= closure(\{[E' \to E\ \bullet, E \to E\ \bullet + T]\})$$
$$= \{\ [E' \to E\ \bullet], [E \to E\ \bullet + T]\ \}$$

Grammar:

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to \mathbf{(}\ E\ \mathbf{)}$$
$$F \to \mathbf{id}$$

The *goto* Operation Example 2

Suppose $I = \{ [E' \rightarrow E \bullet], [E \rightarrow E \bullet + T] \}$

Then $goto(I,+) = closure(\{[E \rightarrow E + \bullet\ T]\}) = \{ [E \rightarrow E + \bullet\ T]$

$[T \rightarrow \bullet\ T * F]$

$[T \rightarrow \bullet\ F]$

$[F \rightarrow \bullet\ \mathbf{(}\ E\ \mathbf{)}]$

$[F \rightarrow \bullet\ \mathbf{id}] \}$

Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow \mathbf{(}\ E\ \mathbf{)}$

$F \rightarrow \mathbf{id}$

Constructing the Canonical LR(0) Collection of a Grammar

1. The grammar is augmented with a new start symbol S' and production $S' \to S$
2. Initially, set $C = \{$ ***closure***$(\{[S' \to \bullet S]\})$ $\}$ (this is the start state of the DFA)
3. For each set of items $I \in C$ and each grammar symbol $X \in (N \cup T)$ such that ***goto***$(I,X) \notin C$ and ***goto***$(I,X) \neq \emptyset$, add the set of items ***goto***(I,X) to C
4. Repeat 3 until no more sets can be added to C

LR(0) Automaton for

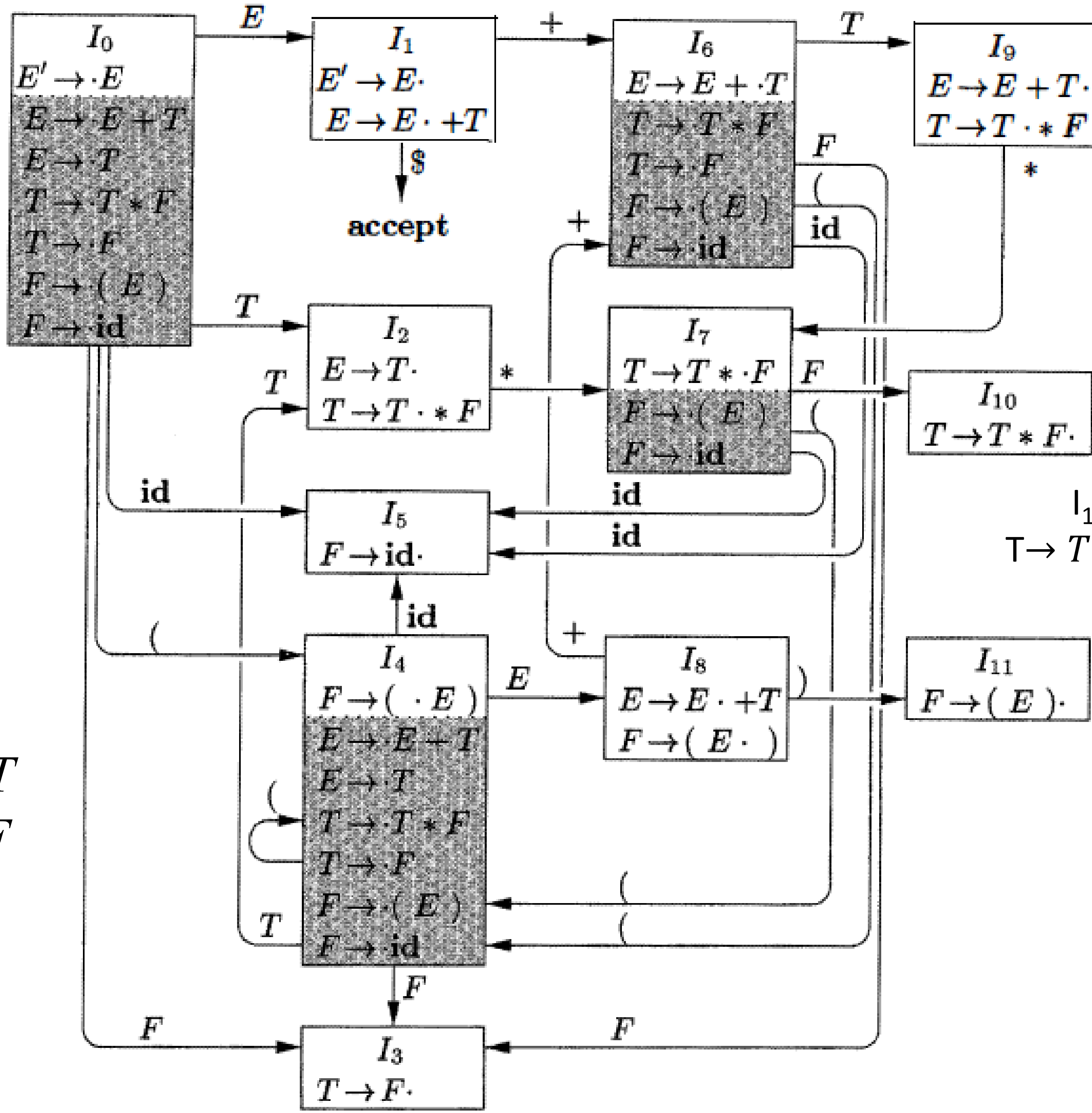
Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E)$

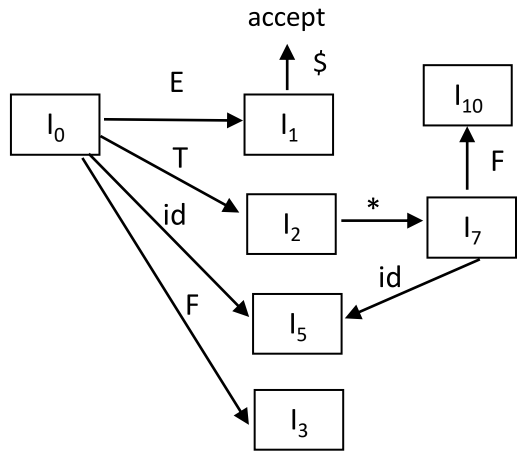
$F \rightarrow \textbf{id}$



I_{10}

$T \rightarrow T * F \cdot$

Use of the LR(0) Automaton



The following Figure shows the actions of a shift-reduce parser on input **id * id**, using the LR(0) automaton shown on previous slide.

LINE	STACK	SYMBOLS	INPUT	ACTION
(1)	0	$	**id** $*$ **id** $	shift to 5
(2)	0 5	$ **id**	$*$ **id** $	reduce by $F \to \mathbf{id}$
(3)	0 3	$ F	$*$ **id** $	reduce by $T \to F$
(4)	0 2	$ T	$*$ **id** $	shift to 7
(5)	0 2 7	$ $T *$	**id** $	shift to 5
(6)	0 2 7 5	$ $T *$ **id**	$	reduce by $F \to \mathbf{id}$
(7)	0 2 7 10	$ $T * F$	$	reduce by $T \to T * F$
(8)	0 2	$ T	$	reduce by $E \to T$
(9)	0 1	$ E	$	accept

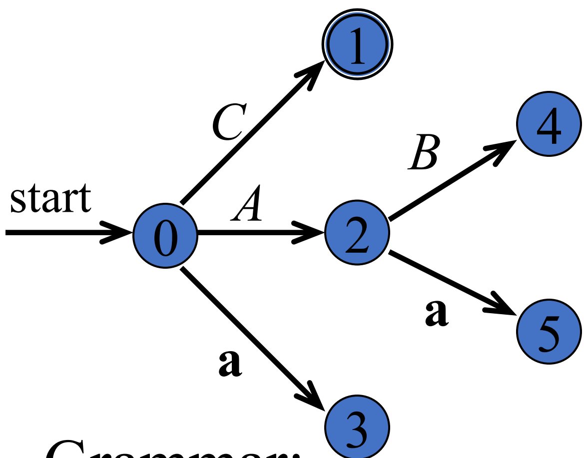
I_5
$F \to id \cdot$

I_3
$T \to F \cdot$

I_2
$E \to T \cdot$

I_{10}
$T \to T * F \cdot$

LR(k) Parsers: Use a DFA for Shift/Reduce Decisions



Use of the LR(0) Automaton

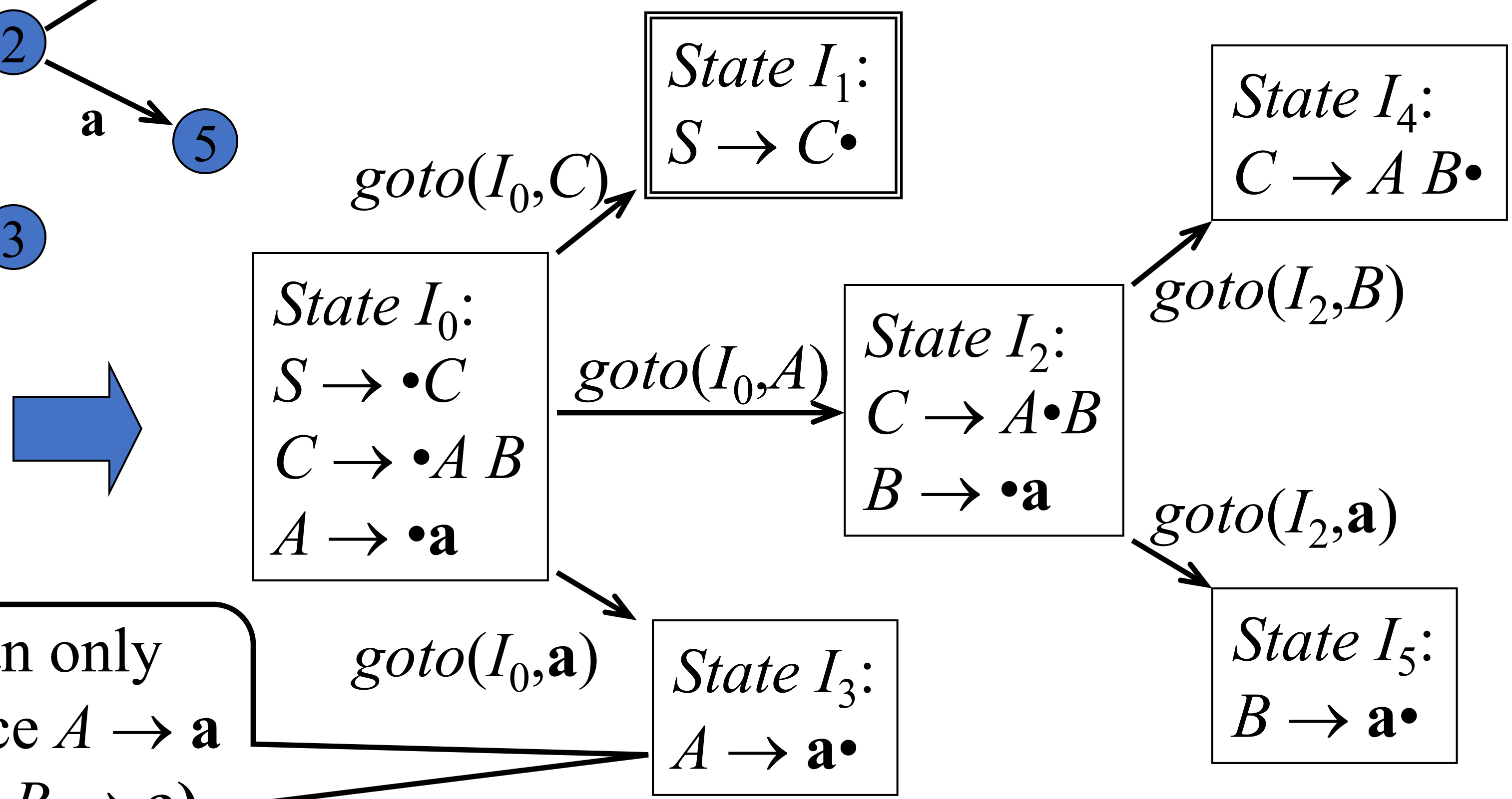
Grammar:

$S \rightarrow C$

$C \rightarrow A\ B$

$A \rightarrow \mathbf{a}$

$B \rightarrow \mathbf{a}$



Can only reduce $A \rightarrow \mathbf{a}$ (not $B \rightarrow \mathbf{a}$)

DFA for Shift/Reduce Decisions

The states of the DFA are used to determine if a handle is on top of the stack

Grammar:

$S \rightarrow C$

$C \rightarrow A\ B$

$A \rightarrow \mathbf{a}$

$B \rightarrow \mathbf{a}$

Stack	Symbols	Input	Action
0	**$**	**aa$**	shift to 3
0 3	**$a**	**a$**	reduce $A \rightarrow \mathbf{a}$
0 2	$A	**a$**	shift to 5
0 2 5	$A**a**	**$**	reduce $B \rightarrow \mathbf{a}$
0 2 4	$AB	**$**	reduce $C \rightarrow AB$
0 1	$C	**$**	accept ($S \rightarrow C$)

State I_0:

$S \rightarrow \bullet C$

$C \rightarrow \bullet A\ B$

$A \rightarrow \bullet\mathbf{a}$

$goto(I_0,\mathbf{a})$ →

State I_3:

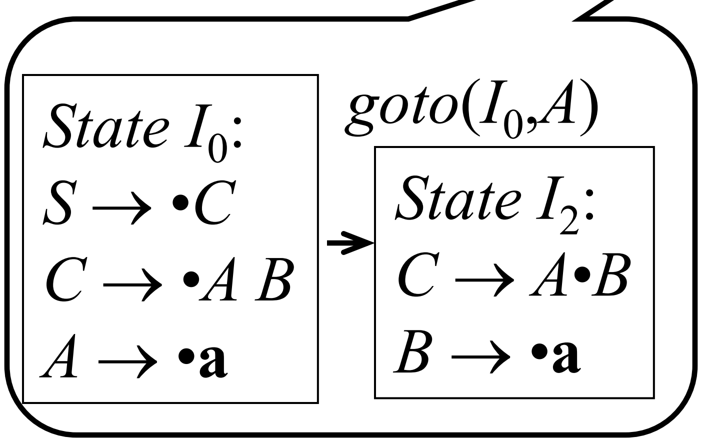
$A \rightarrow \mathbf{a}\bullet$

DFA for Shift/Reduce Decisions

The states of the DFA are used to determine if a handle is on top of the stack

Grammar:
$S \to C$
$C \to A\ B$
$A \to \mathbf{a}$
$B \to \mathbf{a}$

Stack	Symbols	Input	Action
0	$	**aa$**	shift to 3
0 3	$**a**	**a$**	reduce $A \to \mathbf{a}$
0 2	$A	**a$**	shift to 5
0 2 5	$A**a**	$	reduce $B \to \mathbf{a}$
0 2 4	$AB	$	reduce $C \to AB$
0 1	$C	$	accept ($S \to C$)



DFA for Shift/Reduce Decisions

The states of the DFA are used to determine if a handle is on top of the stack

Grammar:

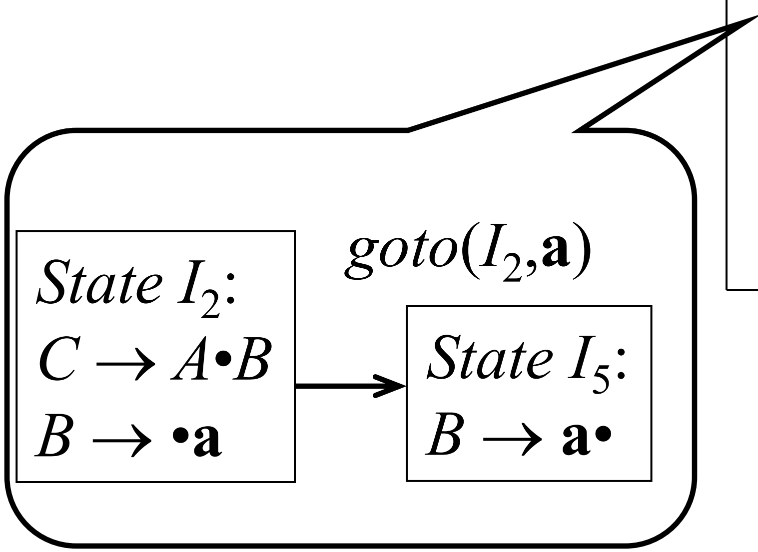
$S \rightarrow C$

$C \rightarrow A\ B$

$A \rightarrow \mathbf{a}$

$B \rightarrow \mathbf{a}$

Stack	Symbols	Input	Action
0	**$**	**aa$**	shift to 3
0 3	**$a**	**a$**	reduce $A \rightarrow \mathbf{a}$
0 2	$A	**a$**	shift to 5
0 2 5	$A**a**	**$**	reduce $B \rightarrow \mathbf{a}$
0 2 4	$AB	**$**	reduce $C \rightarrow AB$
0 1	$C	**$**	accept ($S \rightarrow C$)



DFA for Shift/Reduce Decisions

The states of the DFA are used to determine if a handle is on top of the stack

Grammar:

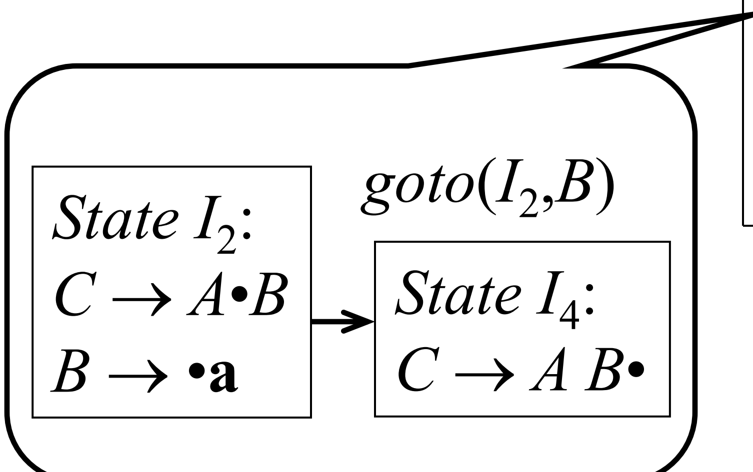
$S \to C$

$C \to A\ B$

$A \to \mathbf{a}$

$B \to \mathbf{a}$

Stack	Symbols	Input	Action
0	**$**	**aa$**	shift to 3
0 3	**$a**	**a$**	reduce $A \to \mathbf{a}$
0 2	$A	**a$**	shift to 5
0 2 5	$A**a**	**$**	reduce $B \to \mathbf{a}$
0 2 4	$AB	**$**	reduce $C \to AB$
0 1	$C	**$**	accept ($S \to C$)



DFA for Shift/Reduce Decisions

The states of the DFA are used to determine if a handle is on top of the stack

Grammar:

$S \to C$

$C \to A\ B$

$A \to \mathbf{a}$

$B \to \mathbf{a}$

Stack	Symbols	Input	Action
0	**$**	**aa$**	shift to 3
0 3	**$a**	**a$**	reduce $A \to \mathbf{a}$
0 2	$A	**a$**	shift to 5
0 2 5	$A**a**	**$**	reduce $B \to \mathbf{a}$
0 2 4	$AB	**$**	reduce $C \to AB$
0 1	$C	**$**	accept ($S \to C$)

State I_0:

$S \to \bullet C$

$C \to \bullet A\ B$

$A \to \bullet\mathbf{a}$

$goto(I_0, C)$ →

State I_1:

$S \to C\bullet$

DFA for Shift/Reduce Decisions

The states of the DFA are used to determine if a handle is on top of the stack

Grammar:

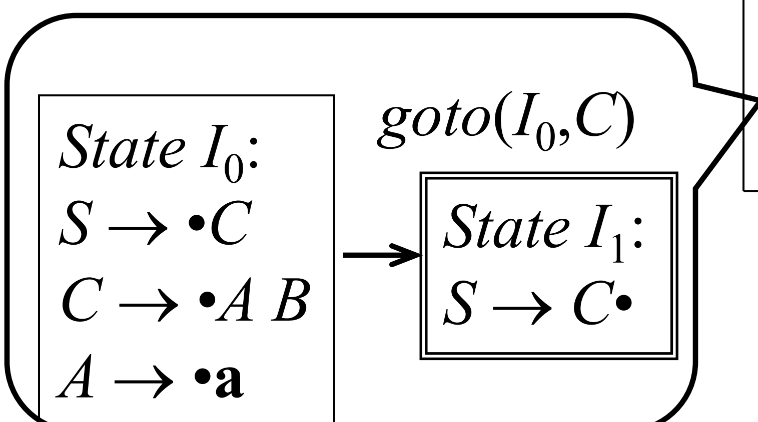
$S \rightarrow C$

$C \rightarrow A\ B$

$A \rightarrow \mathbf{a}$

$B \rightarrow \mathbf{a}$

Stack	Symbols	Input	Action
0	**$**	**aa$**	shift to 3
0 3	**$a**	**a$**	reduce $A \rightarrow \mathbf{a}$
0 2	$A	**a$**	shift to 5
0 2 5	$A**a**	**$**	reduce $B \rightarrow \mathbf{a}$
0 2 4	$AB	**$**	reduce $C \rightarrow AB$
0 1	$C	**$**	accept ($S \rightarrow C$)



Model of an LR Parser