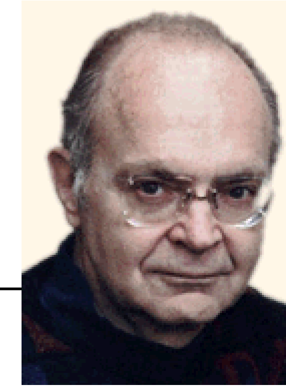
Principles of Compiler Construction

Prof. Wen-jun LI
School of Computer Science and Engineering
lnslwj@mail.sysu.edu.cn

Lecture 7. LR Parsing

1. Introduction to LR Parsing
2. Motivation of LR Parsing
3. Simple LR Parsing
   - LR(0) and SLR(1)
4. More Powerful LR Parsing
   - LR(1) and LALR(1)
5. Ambiguity in LR Parsing
6. Error Recovery in LR Parsing

1. Introduction to LR Parsing



- Proposed by
  - D. Knuth (Stanford U.). **On the Translation of Languages from Left to Right**. Information and Control, 8(6), 1965, pp.607-639
    - Prof. **高德纳**: The Art of Computer Programming, T_EX, Literal Programming, LR Parsing, Attribute Grammar, etc.
- Pros
  - Recognize almost all practical CFGs (more powerful than LL parsing).
  - High efficiency.
  - Detect errors as soon as possible.
- Cons
  - A large size of parsing table.
  - Hard to construct the parsing table manually.

2. Motivation of LR Parsing

- Critical: shift-reduce decision making
  - Maintain states to keep track of where we are in the parsing procedure.
- States are represented by items
  - $A \rightarrow X\ Y\ Z$ yields four items (LR(0) items)
    - $A \rightarrow \bullet X\ Y\ Z$
    - $A \rightarrow X \bullet Y\ Z$
    - $A \rightarrow X\ Y \bullet Z$
    - $A \rightarrow X\ Y\ Z \bullet$
  - $A \rightarrow \varepsilon$ yields only one item: $A \rightarrow \bullet$

Items

- Different forms of items
  - $A \rightarrow X \bullet a\ Z$
    - A "shift" item
    - Indicates the shift of symbol **a**.
  - $A \rightarrow X \bullet B\ Z$
    - A "reduce-expected" item.
    - Indicates the expectation of reducing a B from the remaining input string.
  - $A \rightarrow X\ Y\ Z \bullet$
    - A "reduce" item
    - Indicates a handle has appeared on top of stack.
  - $S \rightarrow X\ Y\ Z \bullet$
    - An "accept" item (special form of reduce items)
    - Indicates the state of accepting the input.

Augmented Grammar

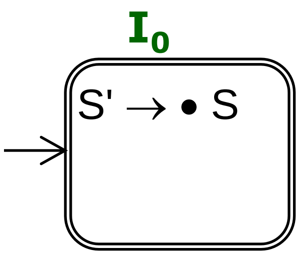
- How and why of augmented grammars?
  - Add a new start symbol S'
    - S' $\rightarrow$ S
    - S $\rightarrow$ ...
  - Intent of augmenting a grammar
    - The start symbol will never appear in the body of any productions.
    - Then the accepting state (items) is unique:

      S' $\rightarrow$ S •

A Motivating Example

- Consider the following grammar

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow \mathbf{a}\,A \mid \mathbf{b}\,B \\
A &\rightarrow \mathbf{c}\,A \mid \mathbf{d} \\
B &\rightarrow \mathbf{c}\,B \mid \mathbf{d}
\end{aligned}
$$



Initial state

I_0

$S' \rightarrow \bullet\ S$

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

$S \rightarrow \bullet\ \mathbf{b}\ B$

Equivalent closure

Kernel items

I_0

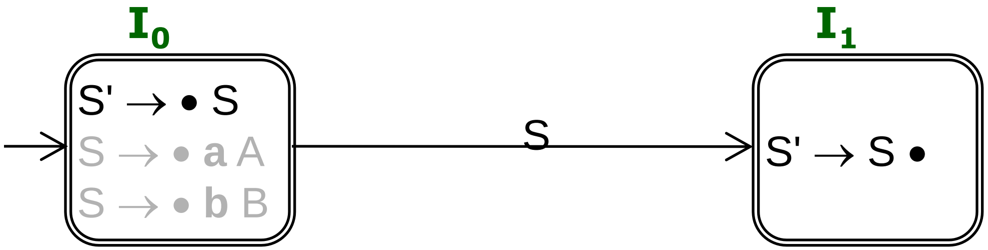
$S' \rightarrow \bullet\ S$

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

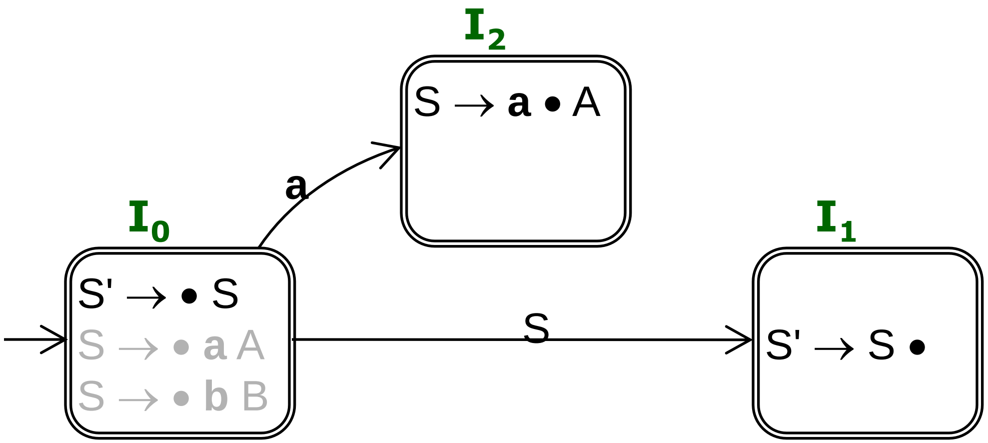
$S \rightarrow \bullet\ \mathbf{b}\ B$

Nonkernel items or closure items

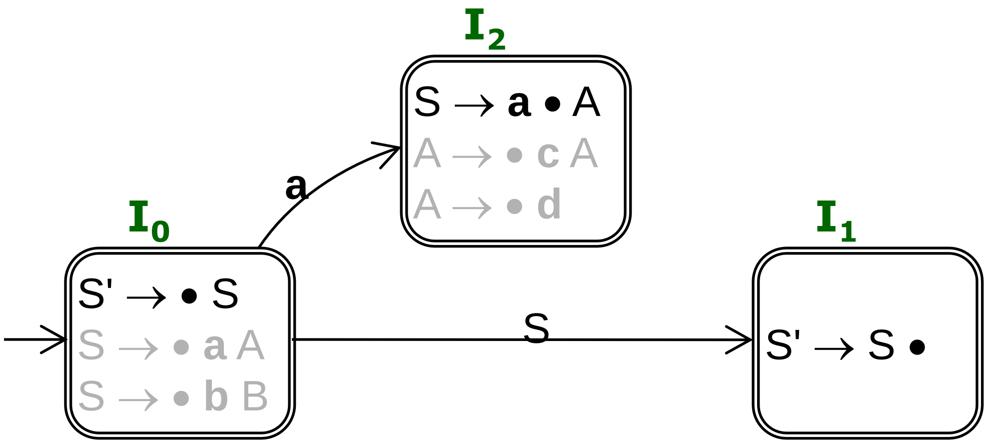
Equivalent closure



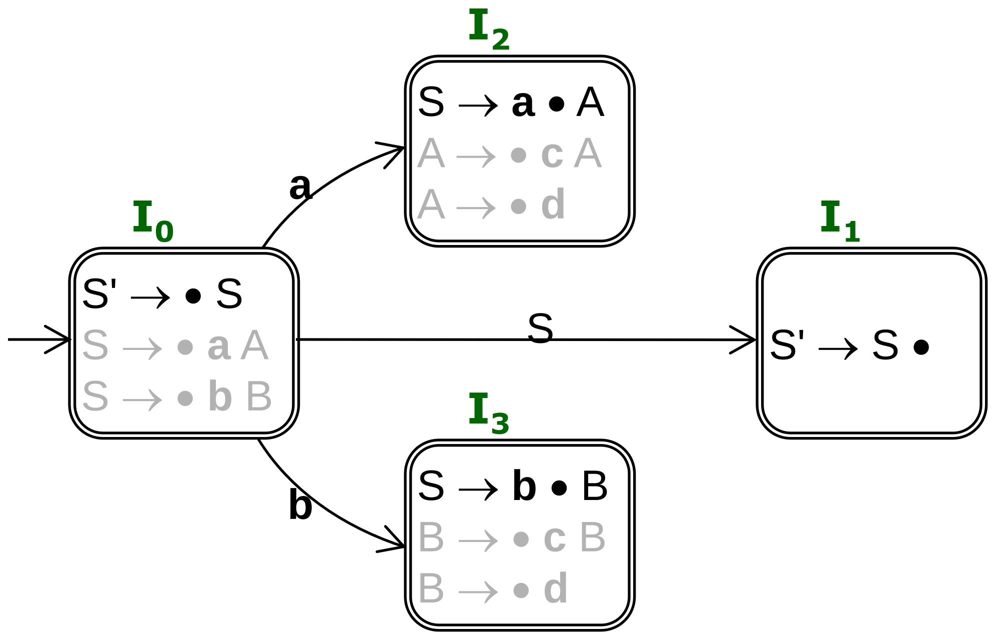
If the remaining string can be reduced to S (expected!)



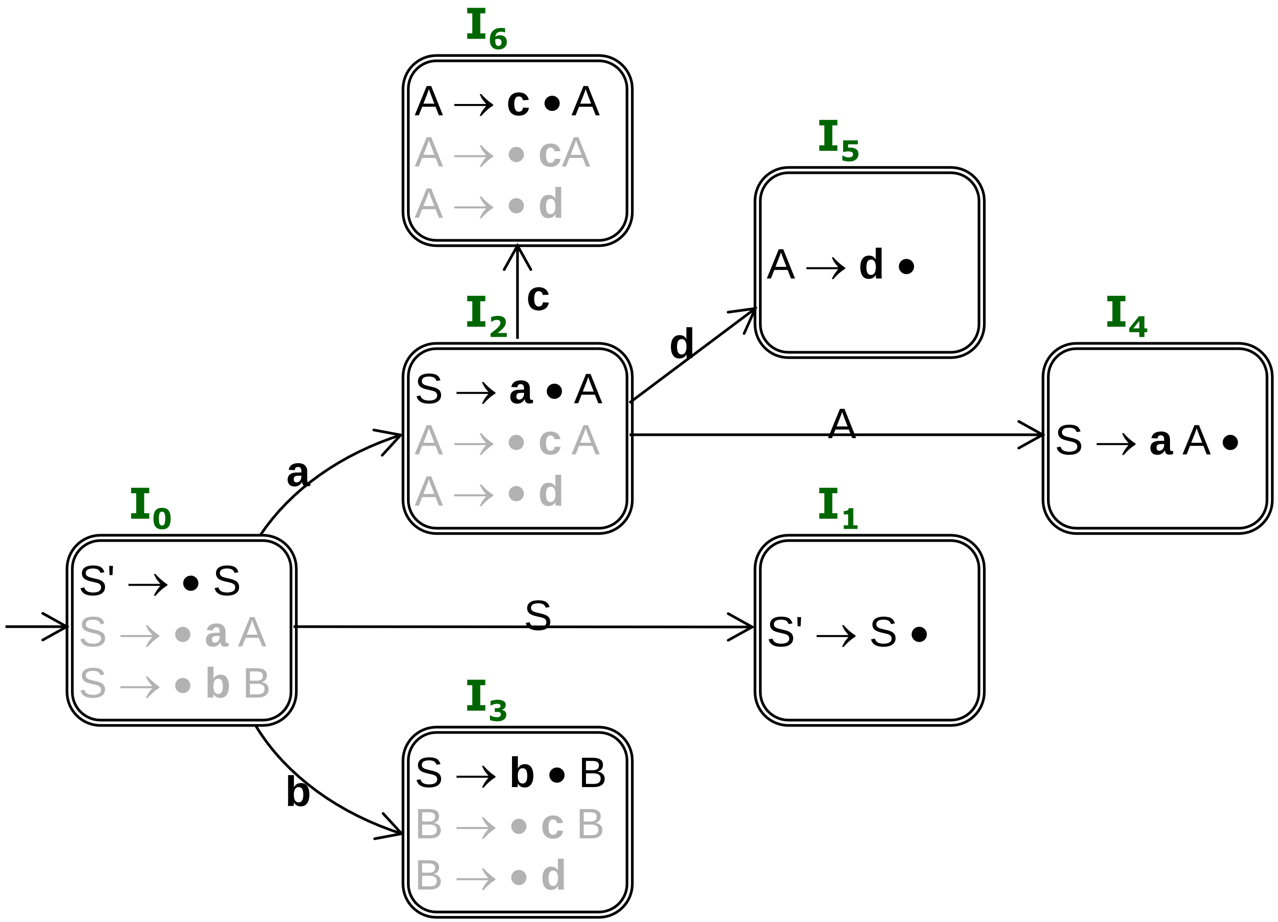
If the first symbol of remaining string is **a** (shift!)



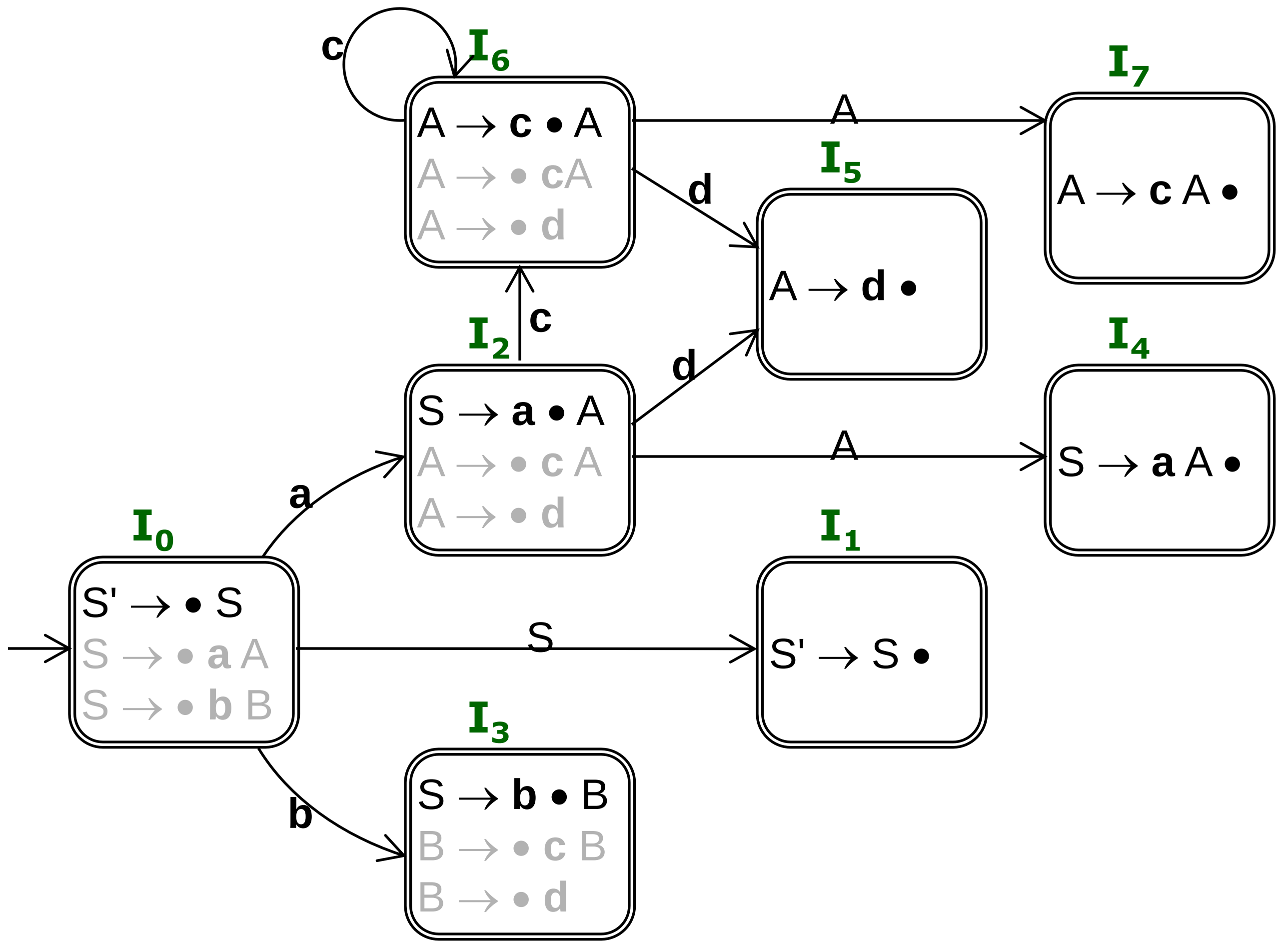
Equivalent closure in state I_2



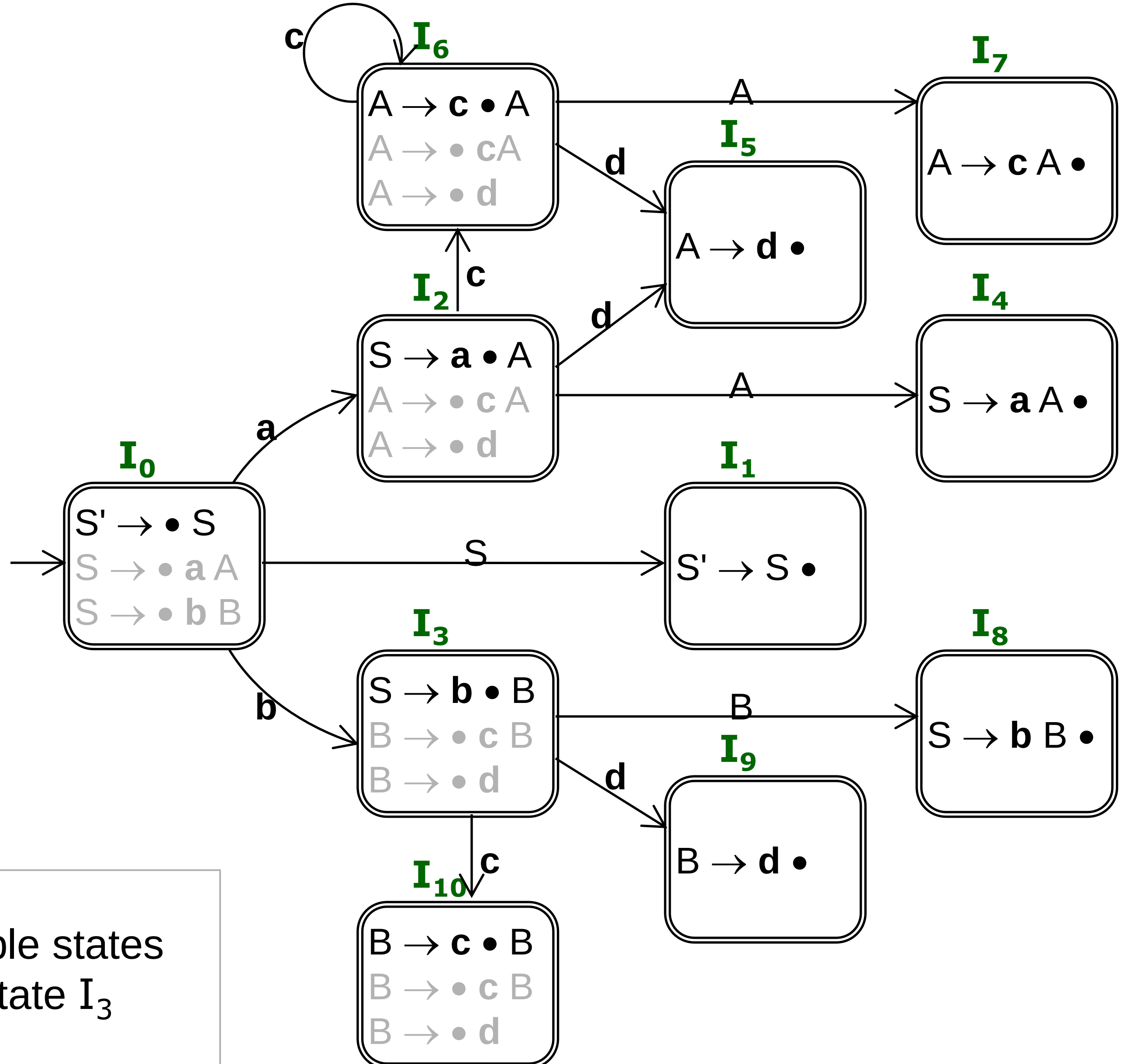
If the first symbol of remaining string is **b** (shift!)



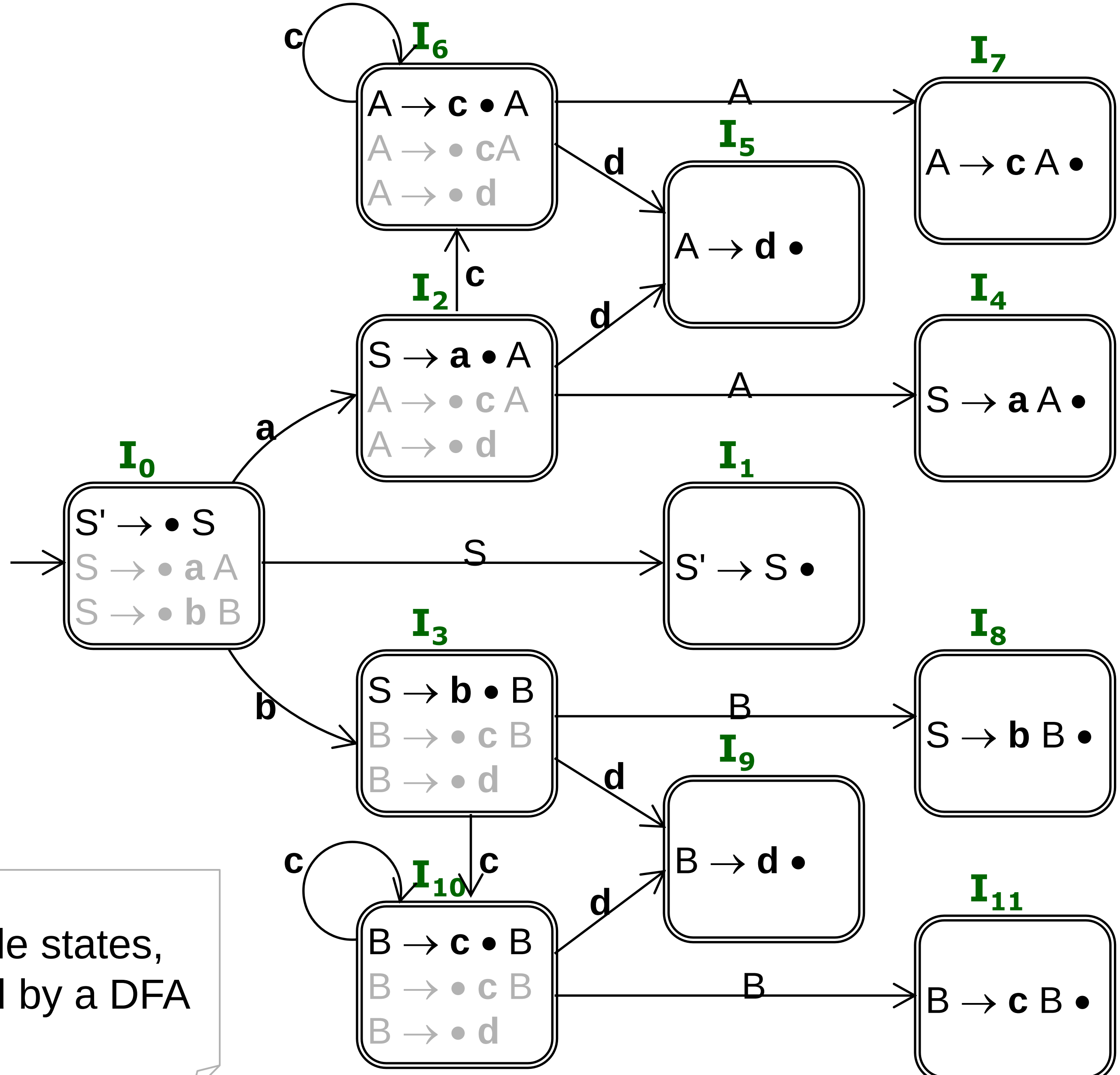
All possible states from state I_2



All possible states from state I_6



All possible states from state I_3



All possible states, recognized by a DFA

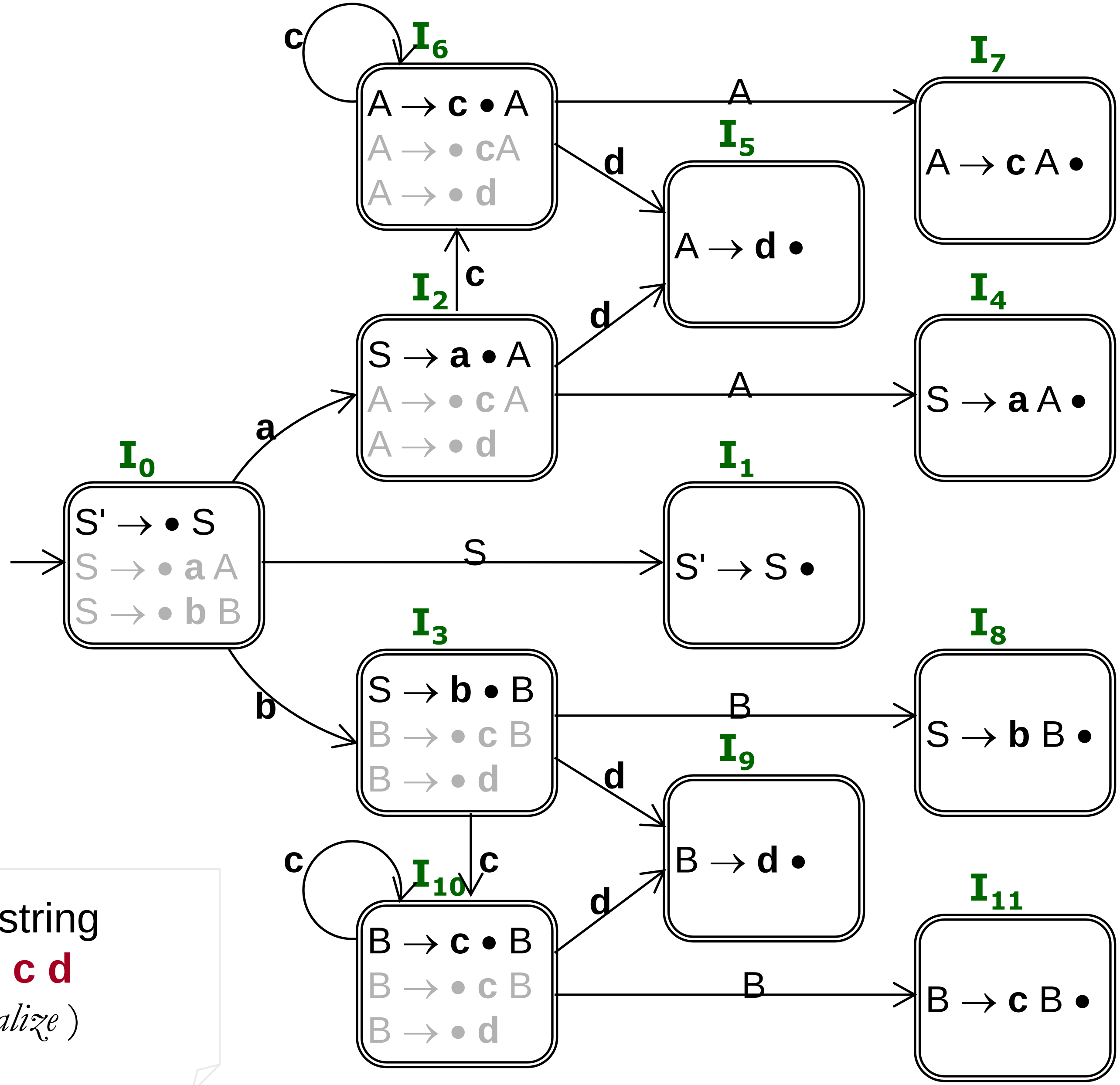
Working with the DFA

- Consider the following sentence:

  $\mathbf{a\ c\ c\ d}$

  We have the right-most derivation:

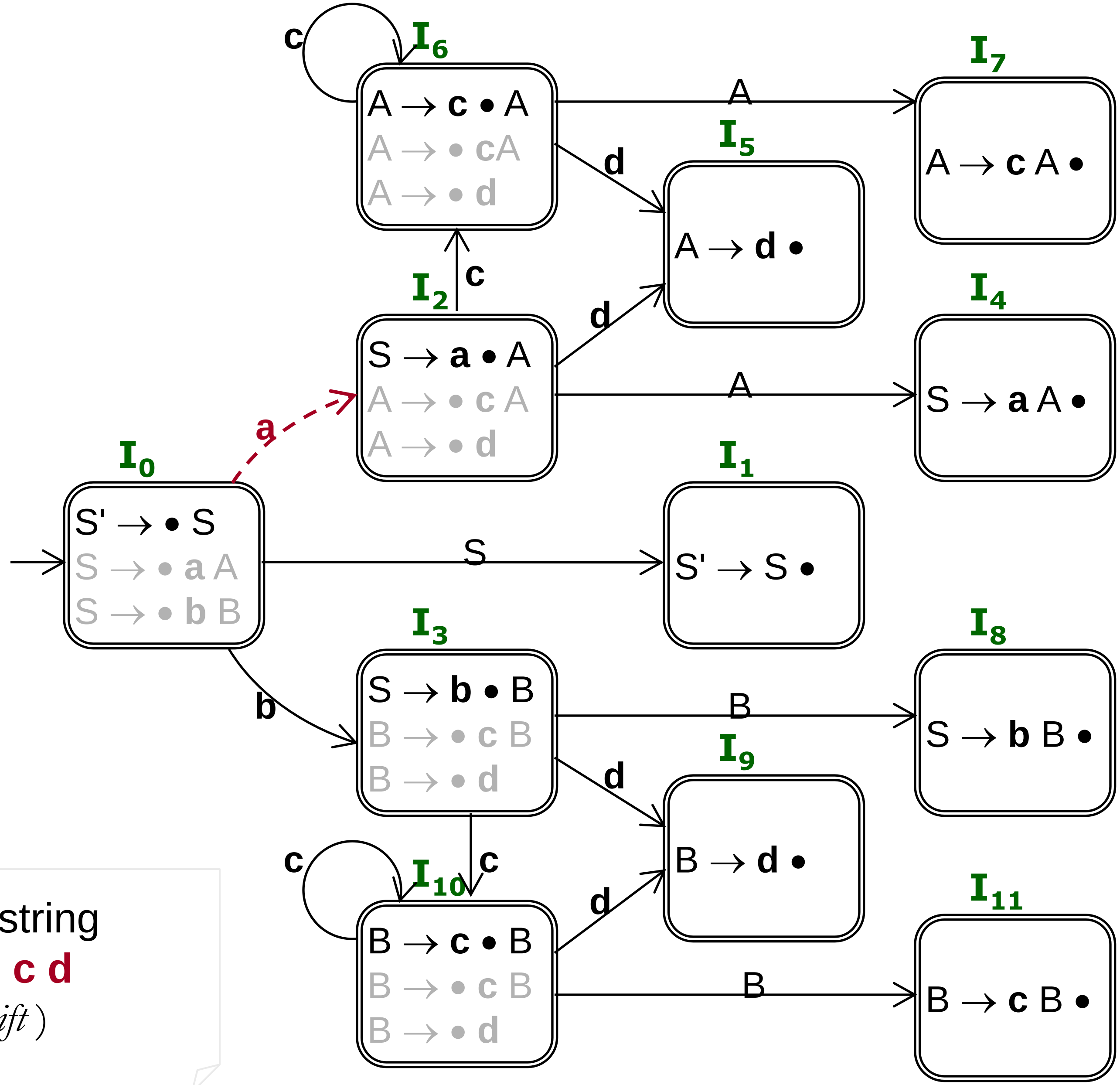
  $S' \Rightarrow S \Rightarrow \mathbf{a}\,A \Rightarrow \mathbf{a\ c}\,A \Rightarrow \mathbf{a\ c\ c}\,A \Rightarrow \mathbf{a\ c\ c\ d}$



Input string

| a c c d

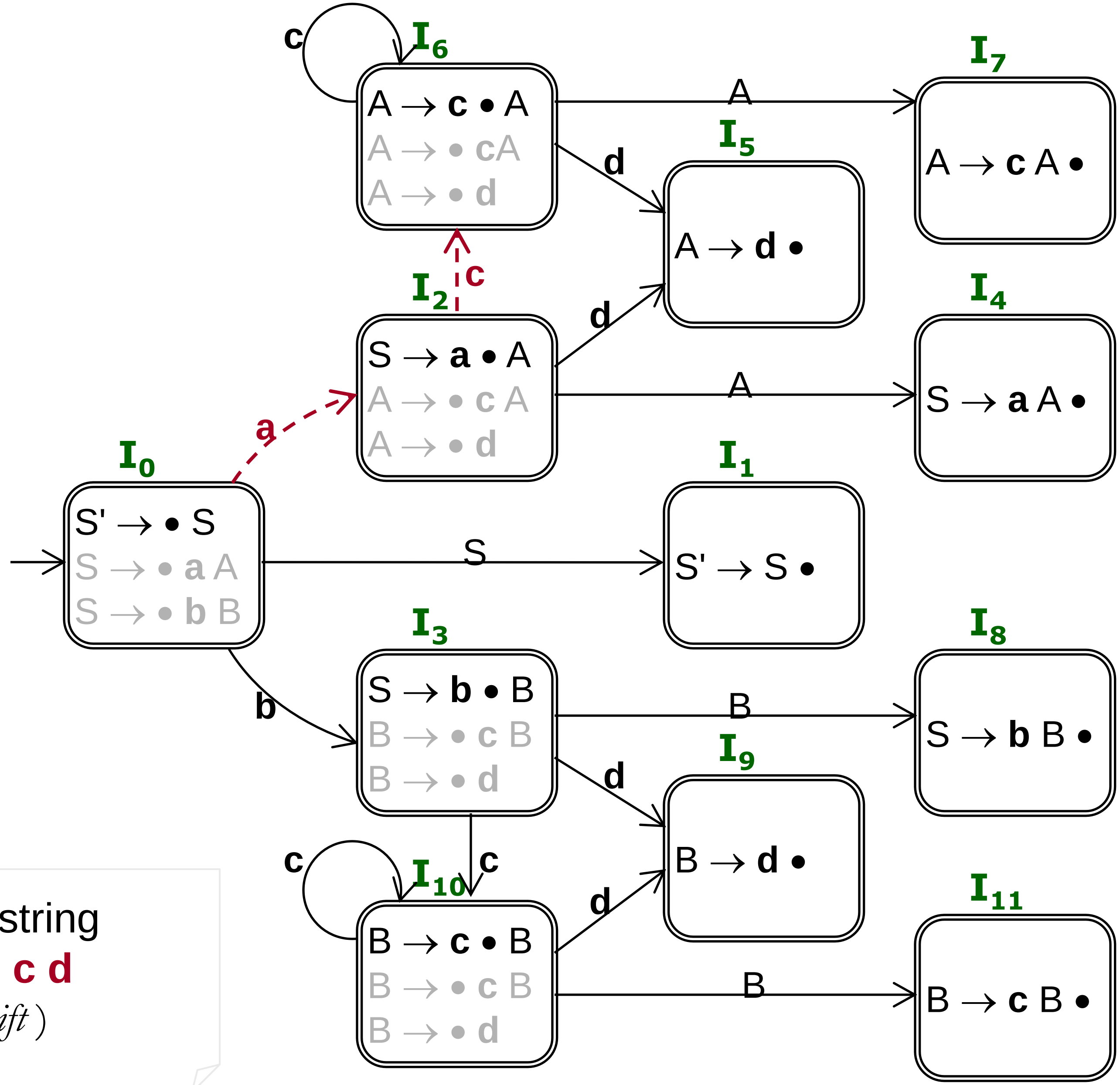
(*initialize*)



Input string

a | c c d

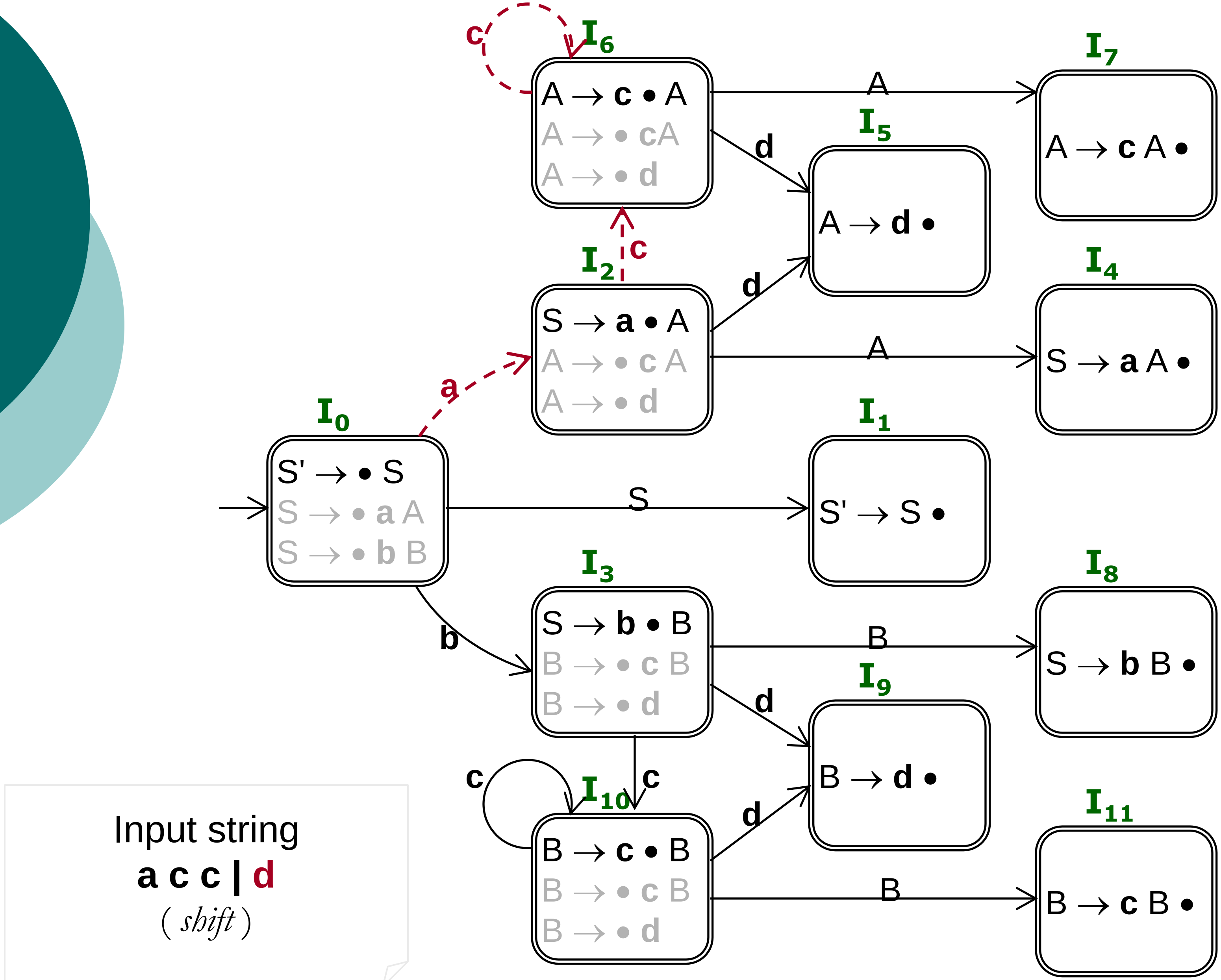
(*shift*)

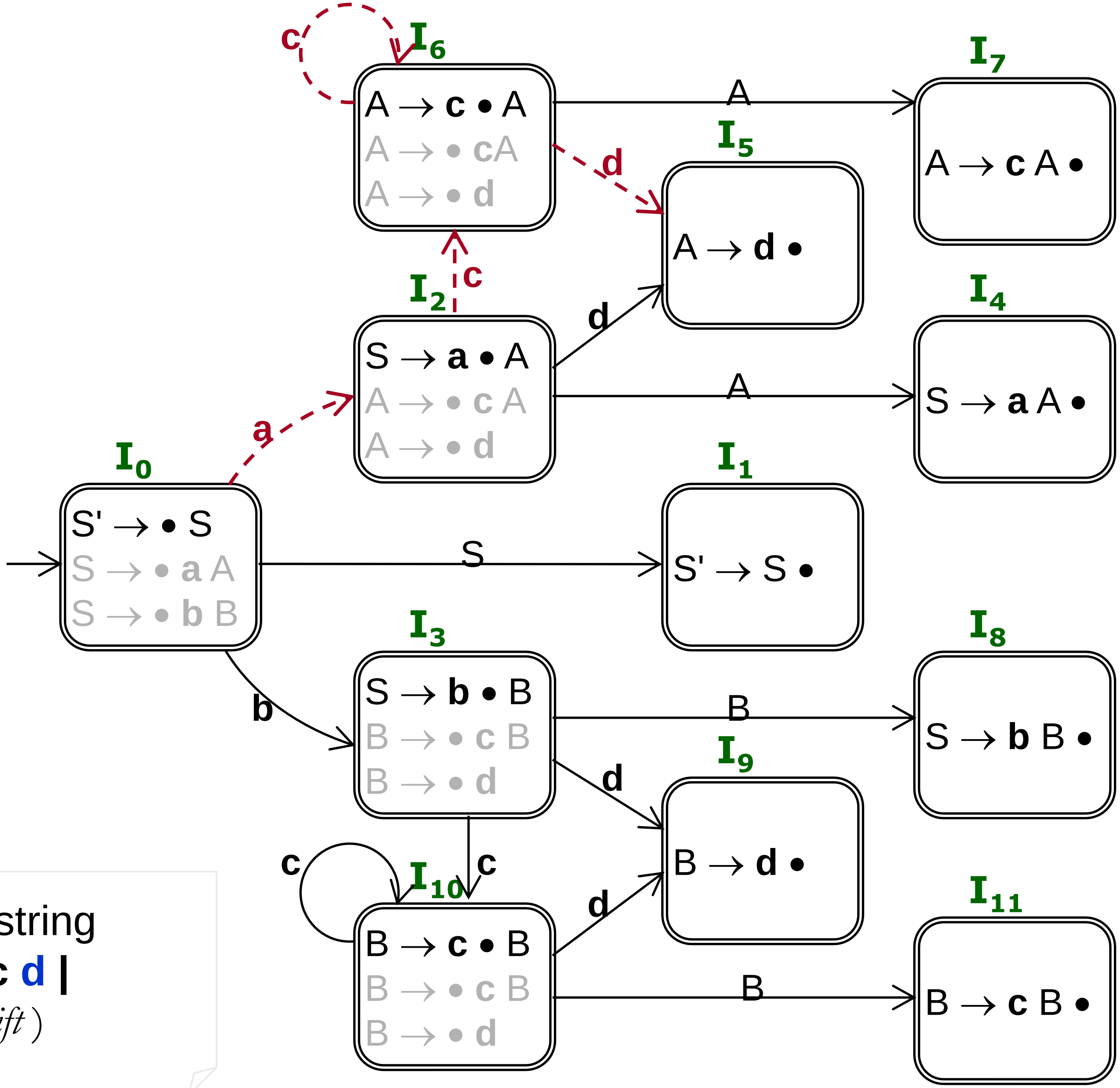


Input string

a c | c d

(shift)


I0
S' → • S
S → • a A
S → • b B
I1
S' → S •
I2
S → a • A
A → • c A
A → • d
I3
S → b • B
B → • c B
B → • d
I4
S → a A •
I5
A → d •
I6
A → c • A
A → • cA
A → • d
I7
A → c A •
I8
S → b B •
I9
B → d •
I10
B → c • B
B → • c B
B → • d
I11
B → c B •
a
b
c
d
S
A
B
Input string
a c c | d
(shift)

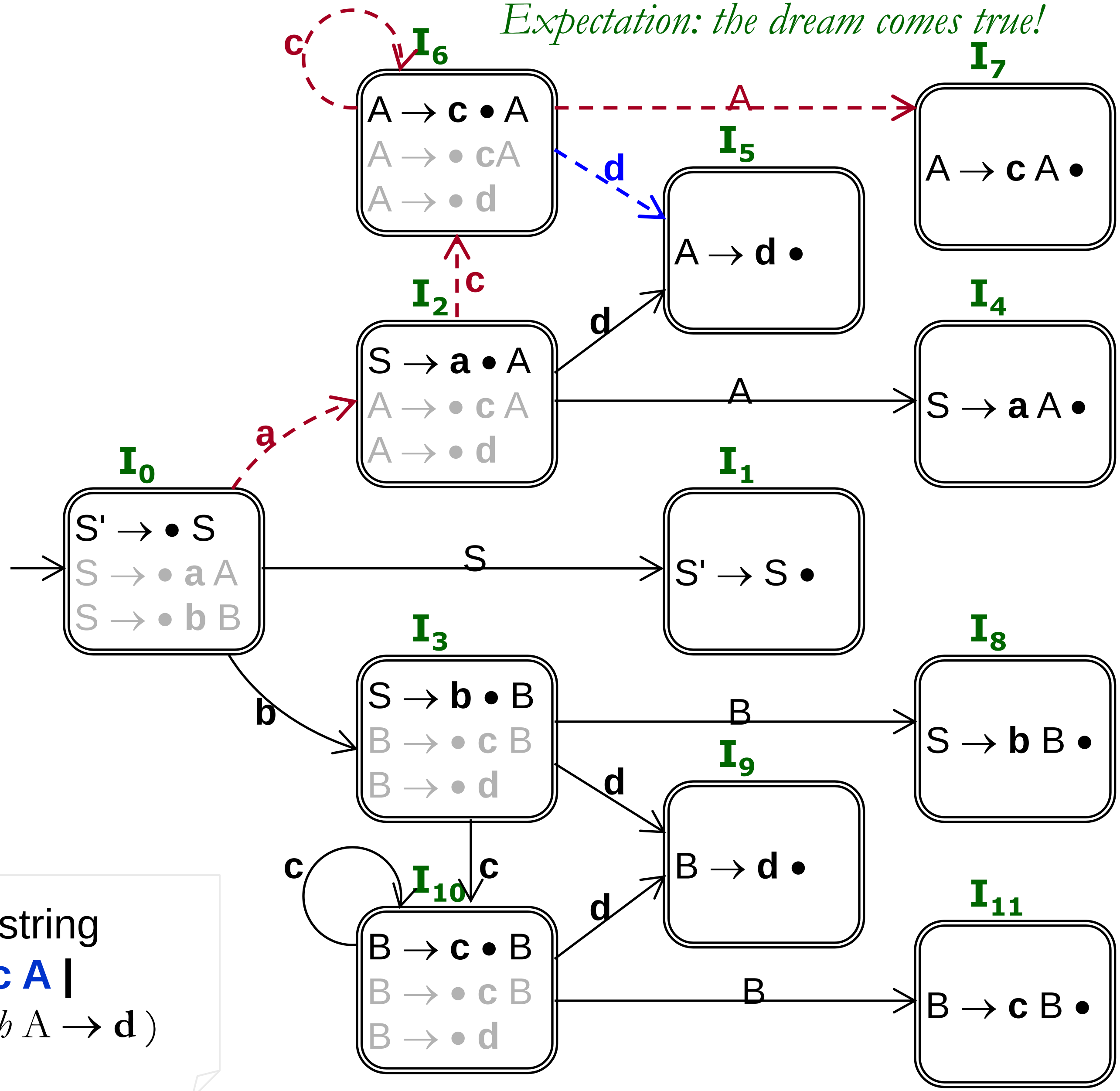


Input string

a c c d |

(shift)

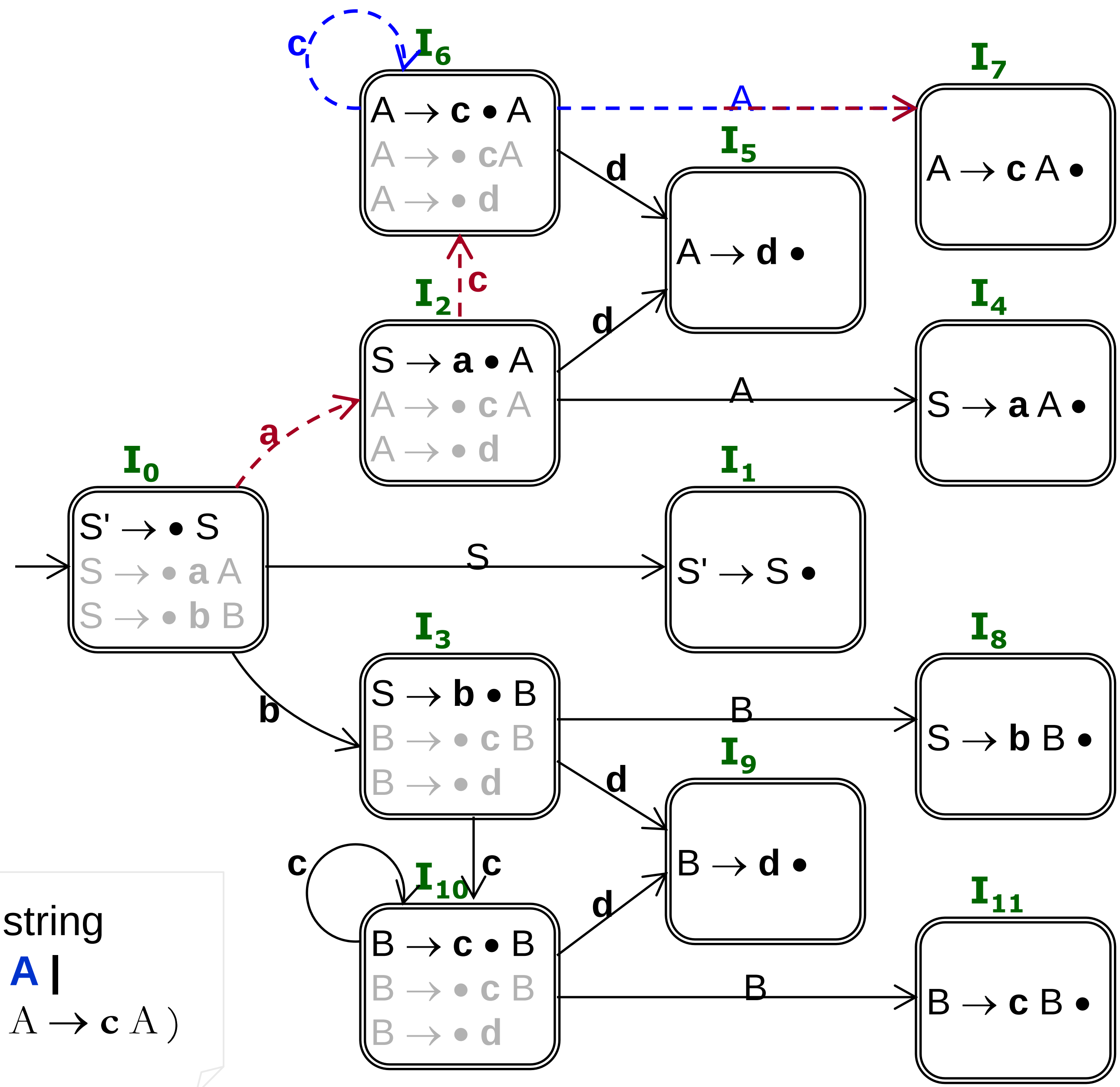
Expectation: the dream comes true!



Input string

a c c A |

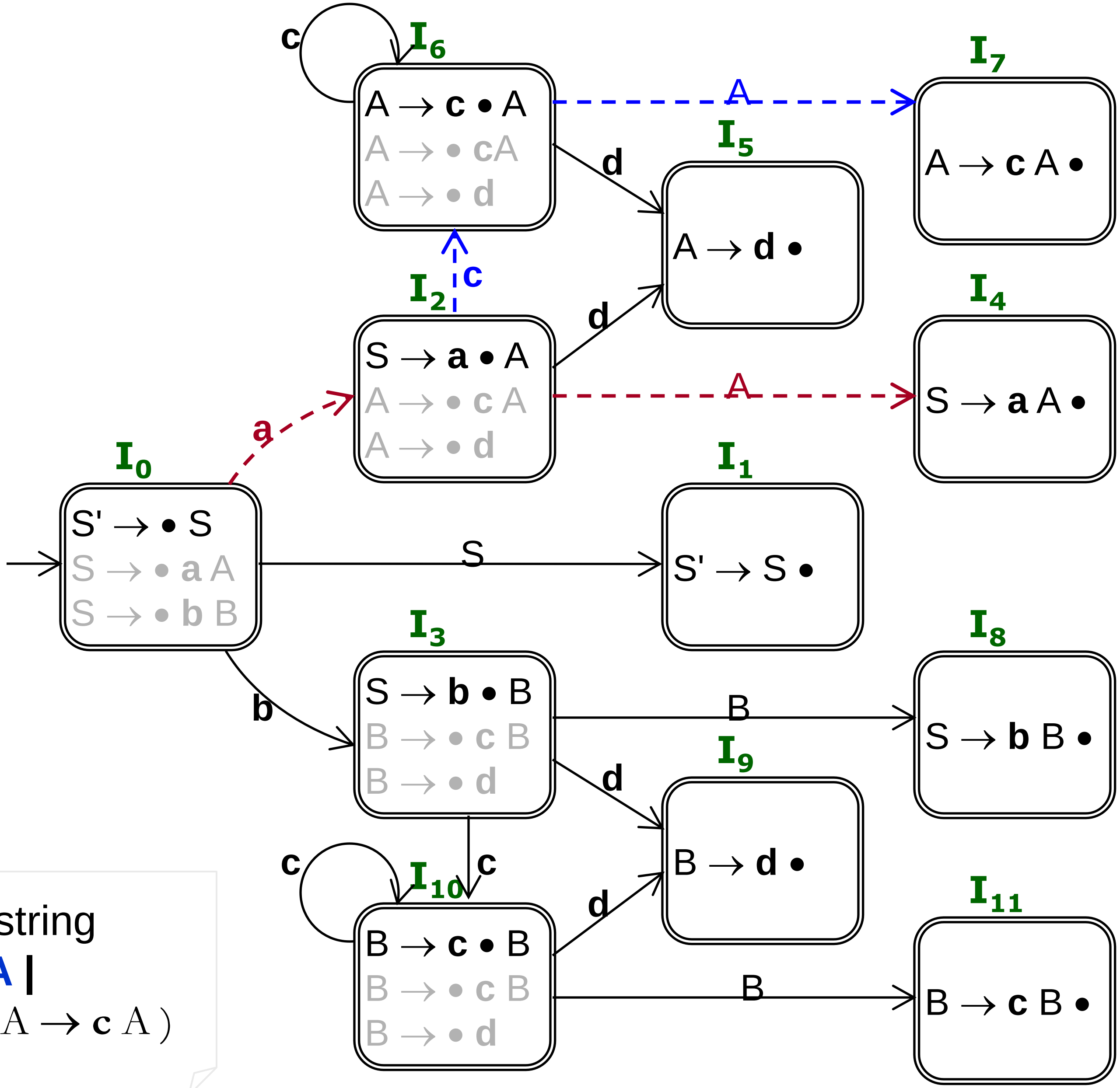
(*reduce with* $A \to \mathbf{d}$)



Input string

a **c** **A** **|**

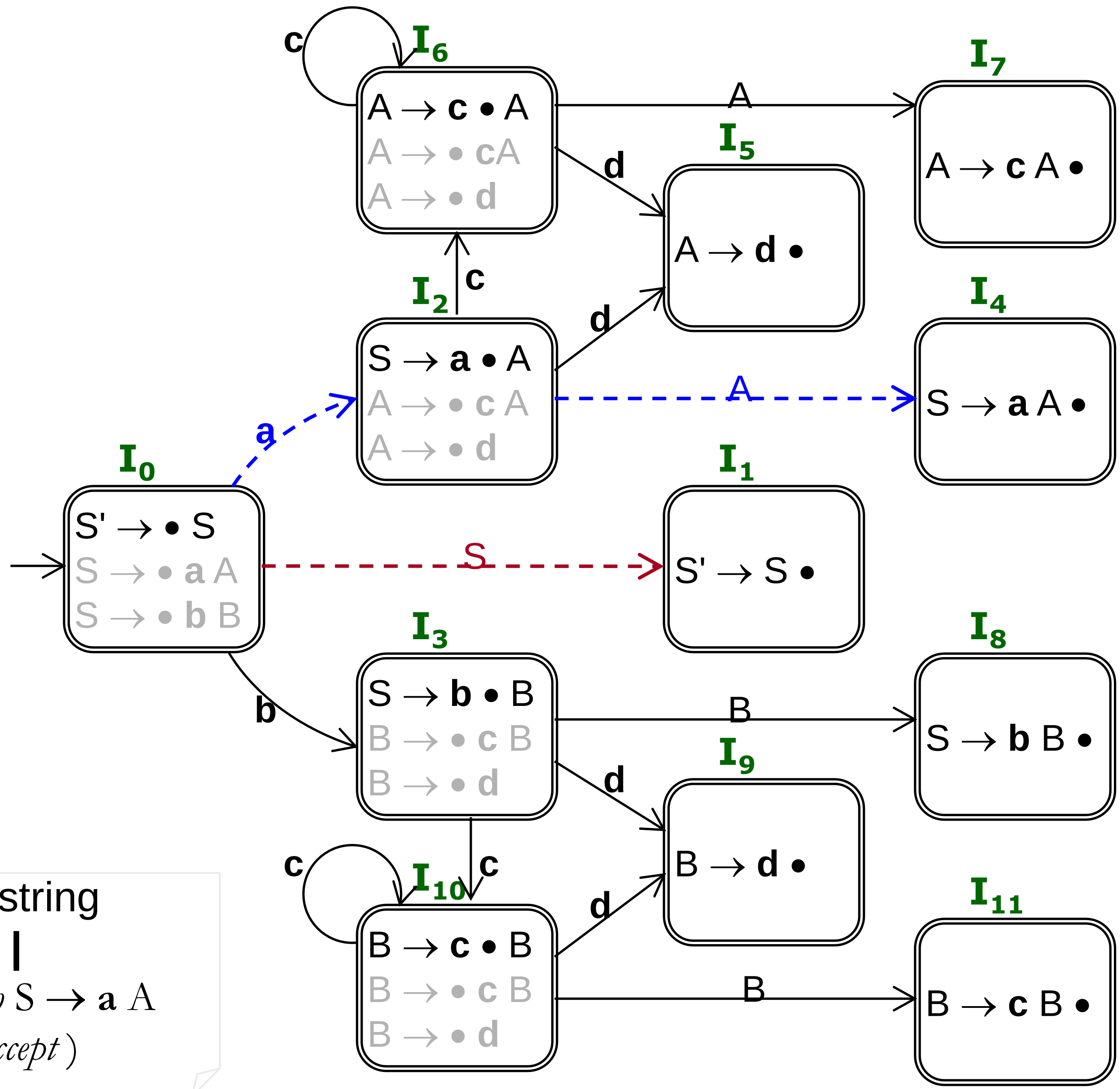
(*reduce with* A → **c** A)



Input string

a A |

(*reduce with* A → **c** A)



Input string

S |

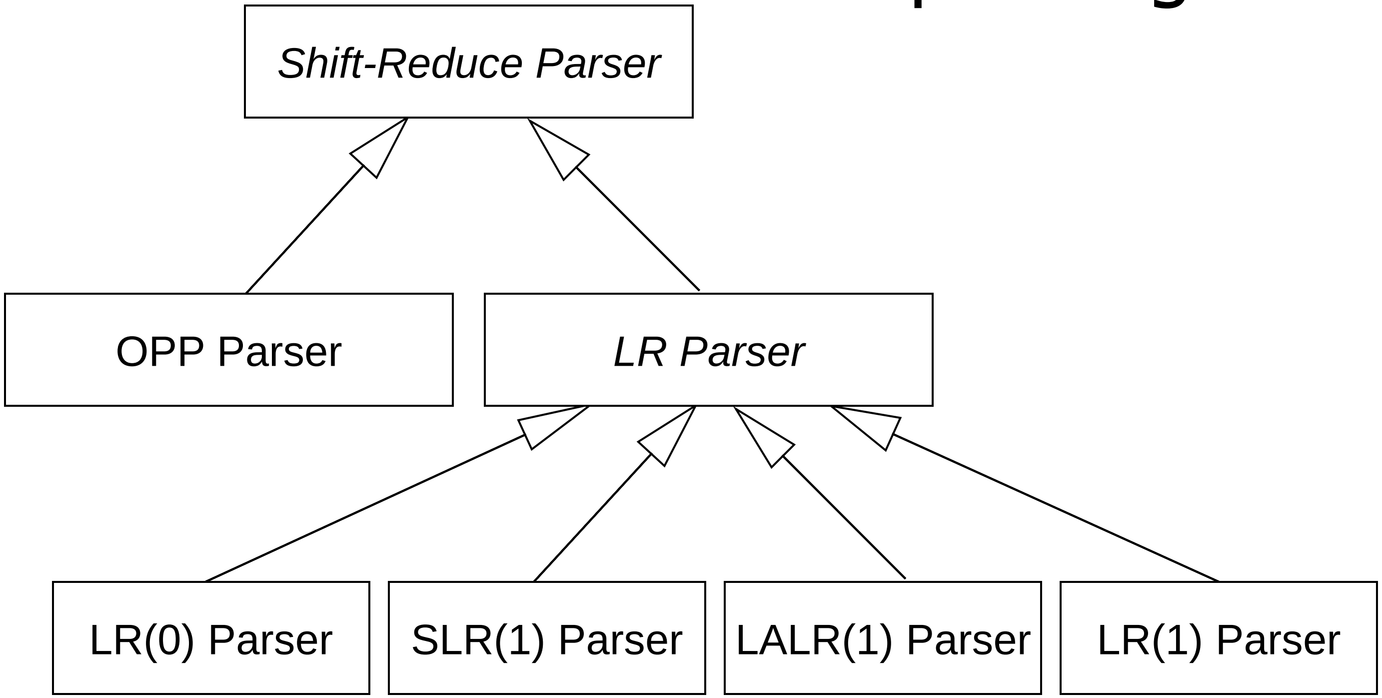
(*reduce with* S → **a** A *and accept*)

Parsing Decisions

Step	Symbol	State	Input	Reference	Action	Output
1	**$**	0	**a c c d $**	S → • **a** A	shift	
2	**$ a**	0 2	**c c d $**	A → • **c** A	shift	
3	**$ a c**	0 2 6	**c d $**	A → • **c** A	shift	
4	**$ a c c**	0 2 6 6	**d $**	A → • **d**	shift	
5	**$ a c c d**	0 2 6 6 5	**$**	A → **d** •	reduce	A → **d**
6	**$ a c c** A	0 2 6 6 7	**$**	A → **c** A •	reduce	A → **c** A
7	**$ a c** A	0 2 6 7	**$**	A → **c** A •	reduce	A → **c** A
8	**$ a** A	0 2 4	**$**	S → **a** A •	reduce	S → **a** A
9	**$** S	0 1	**$**	S' → S •	accept	

Review

- Implementations of the abstract model for shift-reduce parsing

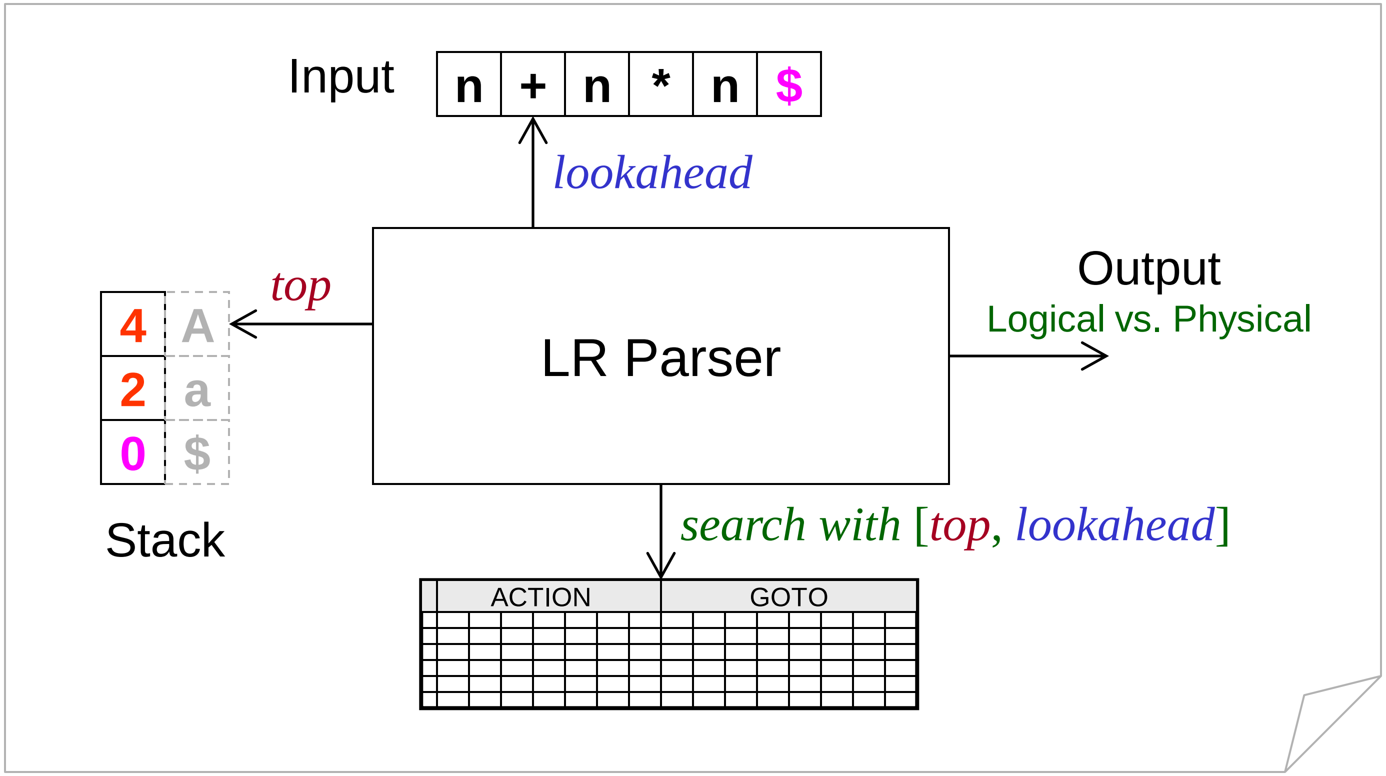


LR Parsers

- Concrete implementations
  - Parsing table
    - ACTION
    - GOTO
  - Explicit stack
    - States
    - Grammar symbols (optional)
  - Reducable substring
    - Handles

A Concrete Model for LR Parser

- LR Parser



What Language the DFA recognizes?

- Viable prefixes of the grammar
  - All strings of grammar symbols that can appear on the stack.
  - *A* ***viable prefix*** *is a prefix of a right-sentential form, that does not continue past the right end of the right-most handle of that sentential form.* (DBv2, pp.256, seg.4, line.2-4)

Learn how to build up a formulation!

Properties of Viable Prefixes

- A viable prefix must be a prefix of a right-sentential form.
  - $S \Rightarrow^*_{rm} \alpha\ \omega$, where α is the content of the stack and ω contains no nonterminals.
- Not all prefixes of a right-sentential form are viable prefixes.
  - $E \Rightarrow^*_{rm} F\ \mathbf{*}\ \mathbf{n} \Rightarrow_{rm} \mathbf{(}\ E\ \mathbf{)}\ \mathbf{*}\ \mathbf{n}$,
  - where **(**, **(E**, **(E)** are viable prefixes,
  - but **(E) *** is not, since the parser will perform reduction once the handle appears.

Definition: Valid Item

- The definition of a valid item
  - Item $A \to \beta_1 \bullet \beta_2$ is valid for a viable prefix $\alpha\beta_1$, if there exists a derivation

    $$S' \Rightarrow^*_{rm} \alpha\ A\ \omega \Rightarrow_{rm} \alpha\ \beta_1\ \beta_2\ \omega$$

    where ω contains terminals only.
  - Hints provided by valid items
    - If $\beta_2 \neq \varepsilon$, the parser should do shift() since a handle has not appear on top of parsing stack.
    - If $\beta_2 = \varepsilon$, the parser should do reduce().

Theorem 1 on Valid Items

- Foundation of closure(), which is used to construct the states of DFA.
  - If $A \to \beta_1 \bullet B\ \beta_2$ is a valid item for $\alpha\beta_1$, and $B \to \gamma$ is a production, then $B \to \bullet\ \gamma$ is also a valid item for $\alpha\beta_1$.
  - [Proof]
    - $S' \Rightarrow^*_{rm} \alpha\ A\ \omega \Rightarrow_{rm} \alpha\ \beta_1\ B\ \beta_2\ \omega$ (definition)
    - $\Rightarrow^*_{rm} \alpha\ \beta_1\ B\ \delta\ \omega$ (suppose $\beta_2\ \omega \Rightarrow^*_{rm} \delta\ \omega$)
    - $\Rightarrow_{rm} \alpha\ \beta_1\ \gamma\ \delta\ \omega$ (we have $B \to \gamma$)
    - $B \to \bullet\ \gamma$ is a valid item for $\alpha\beta_1$ (definition)

Theorem 2 on Valid Items

- Foundation of goto(), which is used to construct the transitions of DFA.
  - If $A \rightarrow \beta_1 \bullet X \beta_2$ is a valid item for $\alpha\beta_1$, then $A \rightarrow \beta_1 X \bullet \beta_2$ is a valid item for $\alpha\beta_1 X$.
  - [Proof]
    - $S' \Rightarrow^*_{rm} \alpha A \omega \Rightarrow_{rm} \alpha \beta_1 X \beta_2 \omega$ (definition)
    - $S' \Rightarrow^*_{rm} \alpha A \omega \Rightarrow_{rm} \alpha \beta_1 X \beta_2 \omega$ (definition)
    - $A \rightarrow \beta_1 X \bullet \beta_2$ is valid for $\alpha\beta_1 X$ (definition)

3. Simple LR Parsing

- Steps of parsing table construction
  - Augment the grammar.
    - To ensure a unique accepting state.
  - Draw the DFA recognizing all viable prefixes of the grammar.
  - Calculate FIRST() and FOLLOW() sets of all nonterminal symbols.
    - Or on-demand calculating while filling the table.
  - Fill the LR parsing table (ACTION and GOTO).
    - Using lookahead to decide reductions.

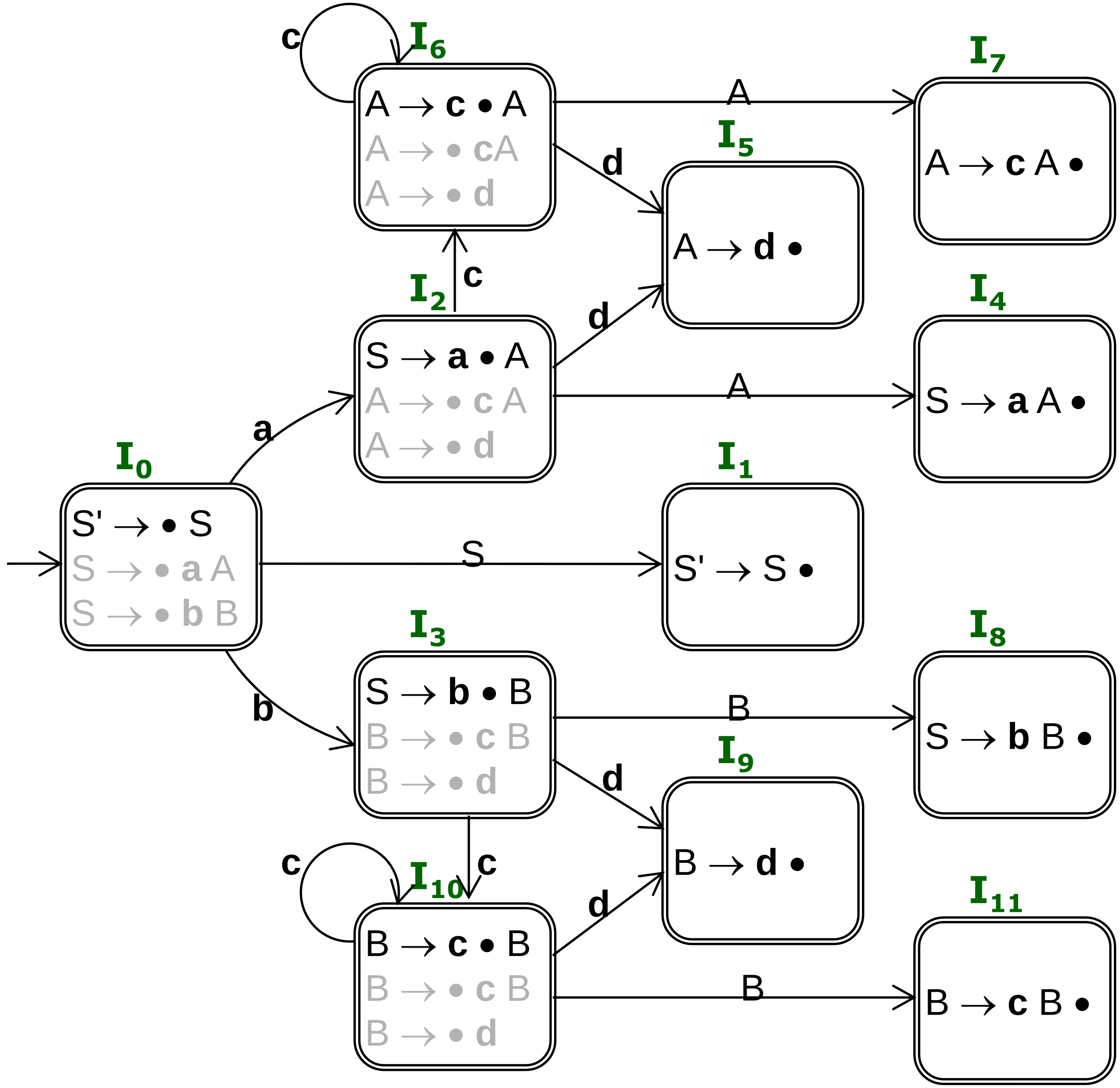
Augmented Grammar

- Given the previous example augmented and numbered:

(0)	S'	$\rightarrow$	S
(1)	S	$\rightarrow$	**a** A
(2)	S	$\rightarrow$	**b** B
(3)	A	$\rightarrow$	**c** A
(4)	A	$\rightarrow$	**d**
(5)	B	$\rightarrow$	**c** B
(6)	B	$\rightarrow$	**d**

FIRST() and FOLLOW() Sets

- We have
  - FIRST(S) = { **a**, **b** }
  - FIRST(A) = { **c**, **d** }
  - FIRST(B) = { **c**, **d** }
  - FOLLOW(S) = { **$** }
  - FOLLOW(A) = { **$** }
  - FOLLOW(B) = { **$** }



Recall: DFA recognizing all viable prefixes

LR(0) Parsing Table

State	ACTION					GOTO		
	a	**b**	**c**	**d**	**$**	S	A	B
0	s2	s3				1		
1					acc			
2			s6	s5			4	
3			s10	s9				8
4	r1	r1	r1	r1	r1			
5	r4	r4	r4	r4	r4			
6			s6	s5			7	
7	r3	r3	r3	r3	r3			
8	r2	r2	r2	r2	r2			
9	r6	r6	r6	r6	r6			
10			s10	s9				11
11	r5	r5	r5	r5	r5			

*LR(0) uses **0** lookahead to decide reducing*

Overlap of Two Tables

State	ACTION					GOTO		
	a	**b**	**c**	**d**	**$**	S	A	B
0	s2	s3				1		
1					acc			
2			s6	s5			4	
3			s10	s9				8
4	r1	r1	r1	r1	r1			
5	r4	r4	r4	r4	r4			
6			s6	s5			7	
7	r3	r3	r3	r3	r3			
8	r2	r2	r2	r2	r2			
9	r6	r6	r6	r6	r6			
10			s10	s9				11
11	r5	r5	r5	r5	r5			

Overlap of Two Tables (cont')

State	ACTION					GOTO		
	a	b	c	d	$	S	A	B
0	s2	s3				1		
1					acc			
2			s6	s5			4	
3			s10	s9				8
4	r1	r1	r1	r1	r1			
5	r4	r4	r4	r4	r4			
6			s6	s5			7	
7	r3	r3	r3	r3	r3			
8	r2	r2	r2	r2	r2			
9	r6	r6	r6	r6	r6			
10			s10	s9				11
11	r5	r5	r5	r5	r5			

Decisions Based on Parsing Table

Step	Symbol	State	Input	Reference	Action	Output
1	$	0	**a c c d $**	a[0, **a**] = s2	shift	
2	$ a	0 2	**c c d $**	a[2, **c**] = s6	shift	
3	$ a c	0 2 6	**c d $**	a[6, **c**] = s6	shift	
4	$ a c c	0 2 6 6	**d $**	a[6, **d**] = s5	shift	
5	$ a c c d	0 2 6 6 5	**$**	a[5, **$**] = r4 g[6, A] = 7	reduce	A → **d**
6	$ a c c A	0 2 6 6 7	**$**	a[7, **$**] = r3 g[6, A] = 7	reduce	A → **c** A
7	$ a c A	0 2 6 7	**$**	a[7, **$**] = r3 g[2, A] = 4	reduce	A → **c** A
8	$ a A	0 2 4	**$**	a[4, **$**] = r1 g[0, S] = 1	reduce	S → **a** A
9	$ S	0 1	**$**	a[1, **$**] = acc	accept	

Simple LR (SLR) Parsing Table

State	ACTION					GOTO		
	a	**b**	**c**	**d**	**$**	S	A	B
0	s2	s3				1		
1					acc			
2			s6	s5			4	
3			s10	s9				8
4					r1			
5					r4			
6			s6	s5			7	
7					r3			
8					r2			
9					r6			
10			s10	s9				11
11					r5			

SLR(1) *simply uses* **1** *lookahead to decide reducing*

(Lookaheads must be in the FOLLOW *set of the reduced symbol)*

Difference between LR(0) and SLR(1)

- Parsing with the LR(0) parsing table

Step	Symbol	State	Input	Reference	Action	Output
1	$	0	**a d d $**	a[0, **a**] = s2	shift	
2	$ a	0 2	**d d $**	a[2, **d**] = s5	shift	
3	**$ a d**	0 2 5	**d $**	a[5, **d**] = r4, g[2, A] = 4	reduce	A → **d**
4	**$ a** A	0 2 4	**d $**	a[4, **d**] = r1, g[0, S] = 1	reduce	S → **a** A
5	**$** S	0 1	**d $**	a[1, **d**] = empty	error	

- Parsing with the SLR(1) parsing table

Step	Symbol	State	Input	Reference	Action	Output
1	$	0	**a d d $**	a[0, **a**] = s2	shift	
2	$ a	0 2	**d d $**	a[2, **d**] = s5	shift	
3	**$ a d**	0 2 5	**d $**	a[5, **d**] = empty	error	

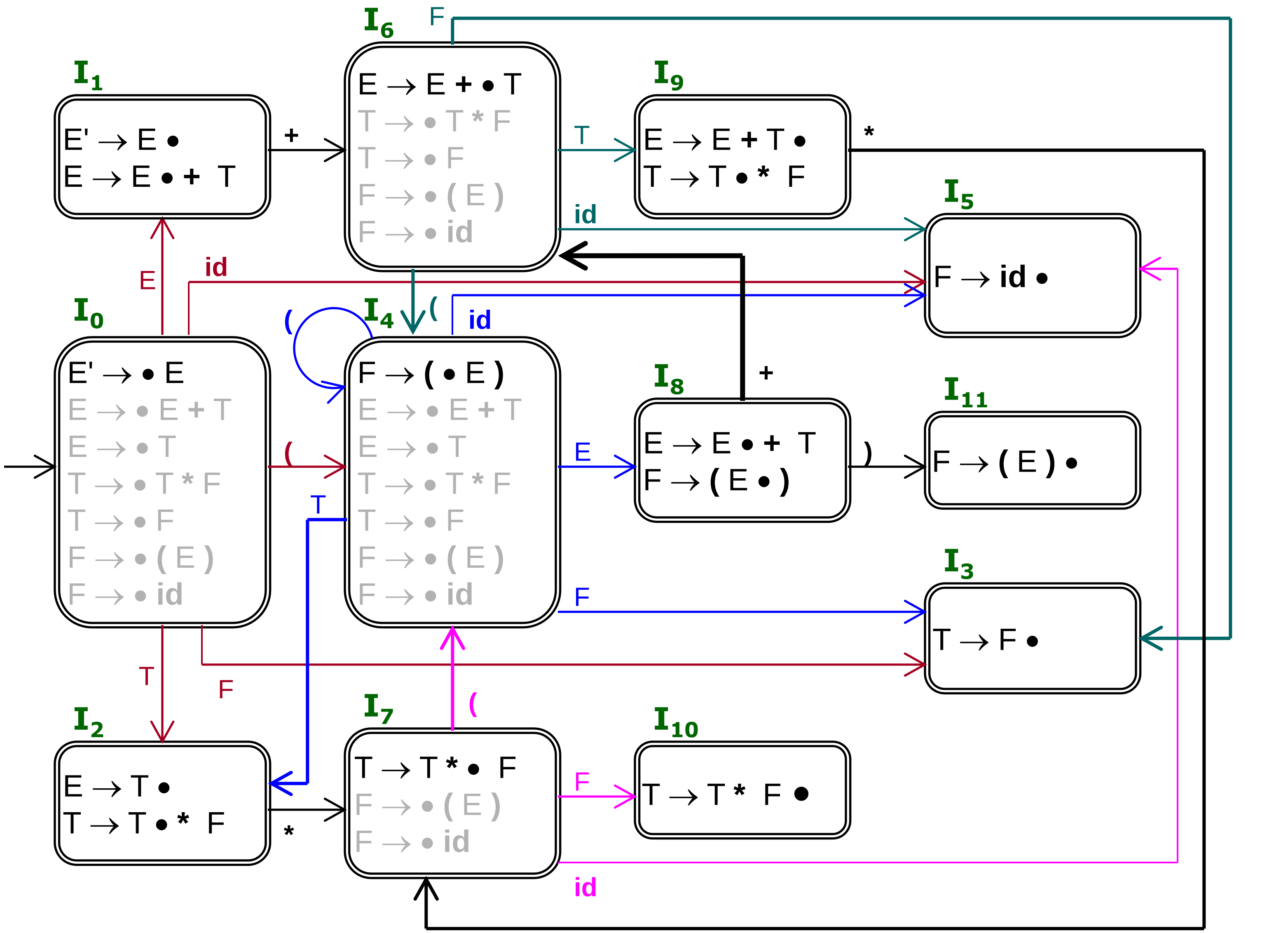
A More Complicated Example

- Consider the **unambiguous** grammar for expressions

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid \textbf{id}$$

- Augmented grammar (numbered)

$$
\begin{array}{llll}
(0) & E' & \rightarrow & E \\
(1) & E & \rightarrow & E + T \\
(2) & E & \rightarrow & T \\
(3) & T & \rightarrow & T * F \\
(4) & T & \rightarrow & F \\
(5) & F & \rightarrow & (E) \\
(6) & F & \rightarrow & \textbf{id}
\end{array}
$$



DFA recognizing all viable prefixes

FIRST() and FOLLOW() Sets

- From the previous lectures, it is easy to calculate the following sets:
  - FIRST(E) = { **(**, **id** }
  - FIRST(T) = { **(**, **id** }
  - FIRST(F) = { **(**, **id** }
  - FOLLOW(E) = { **+**, **)**, **$** }
  - FOLLOW(T) = { **+**, *****, **)**, **$** }
  - FOLLOW(F) = { **+**, *****, **)**, **$** }

SLR(1) Parsing Table

State	ACTION						GOTO		
	id	**+**	*****	**(**	**)**	**$**	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

This Is Not an LR(0) Grammar!

State	ACTION						GOTO		
	id	**+**	*****	**(**	**)**	**$**	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2	r2	r2	s7/r2	r2	r2	r2			
3	r4	r4	r4	r4	r4	r4			
4	s5			s4			8	2	3
5	r6	r6	r6	r6	r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9	r1	r1	s7/r1	r1	r1	r1			
10	r3	r3	r3	r3	r3	r3			
11	r5	r5	r5	r5	r5	r5			

4. More Powerful LR Parsing

- A Grammar That Is Not SLR(1)
- LR(1) Parsing Table
- LALR(1) Parsing Table
- Conflicts in LALR(1) Parsing

An Unambiguous But Not SLR(1) Grammar

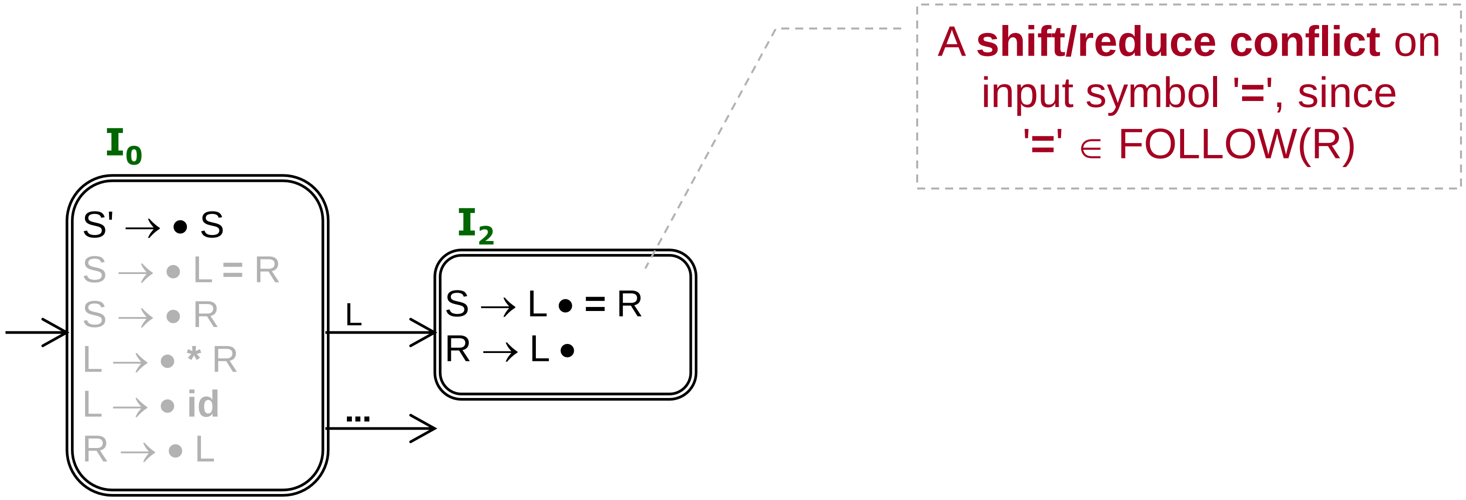
- Consider the unambiguous grammar, which is not SLR(1):

$$S \rightarrow L = R \mid R$$

$$L \rightarrow \mathbf{*}\, R \mid \mathbf{id}$$

$$R \rightarrow L$$

Conflicts in Items



Part of the DFA recognizing all viable prefixes

Motivation of LR(1) Parsing

- How to make use of the lookahead?
  - LR(0): does not use the lookahead.
    - All columns are filled with the same reduction.
  - SLR(1): simply make use of the lookahead.

    - Only columns in the FOLLOW set are filled.
    - More accurate than LR(0), thus less conflicts.
  - More powerful LR parsing – LR(1)

    - Only columns (symbols) that can follow the viable prefixes are filled.
    - More accurate than SLR(1).
  - Trade-off – LALR(1)

    - More efficient but less powerful than LR(1).
    - More powerful than SLR(1).

Definition: Valid LR(1) Item

- The definition of a valid item
  - Item $[A \rightarrow \beta_1 \bullet \beta_2, a]$ is valid for a viable prefix $\alpha\beta_1$, if there exists a derivation
    $$S' \Rightarrow^*_{rm} \alpha\, A\, \omega \Rightarrow_{rm} \alpha\, \beta_1\, \beta_2\, \omega$$
    where ω starts with a, or $\omega = \varepsilon \wedge a = \$$.

Theorem 1 on Valid LR(1) Items

- Foundation of closure(), which is used to construct the states of DFA.
  - If $[A \to \beta_1 \bullet B\ \beta_2,\ a]$ is a valid item for $\alpha\beta_1$, and $B \to \gamma$ is a production, then $[B \to \bullet\ \gamma,\ b]$ is also a valid item for $\alpha\beta_1$, where $b \in \mathrm{FIRST}(\beta_2 a)$.
  - [Proof]
    - $S' \Rightarrow^*_{rm} \alpha\ A\ a\ \omega \Rightarrow_{rm} \alpha\ \beta_1\ B\ \beta_2\ a\ \omega$ (definition)
    - $\Rightarrow^*_{rm} \alpha\ \beta_1\ B\ b\ \delta$ (suppose $\beta_2\ a\ \omega \Rightarrow^*_{rm} b\ \delta$)
    - $\Rightarrow_{rm} \alpha\ \beta_1\ \gamma\ b\ \delta$ (we have $B \to \gamma$)
    - $[B \to \bullet\ \gamma\ ,\ b]$ is valid for $\alpha\beta_1$ (definition)

Theorem 2 on Valid LR(1) Items

- Foundation of goto(), which is used to construct the transitions of DFA.
  - If $[A \to \beta_1 \bullet X\ \beta_2, a]$ is a valid item for $\alpha\beta_1$, then $[A \to \beta_1\ X \bullet \beta_2\ , a]$ is a valid item for $\alpha\beta_1 X$.
  - [Proof]
    - $S' \Rightarrow^*_{rm} \alpha\ A\ a\ \omega \Rightarrow_{rm} \alpha\ \beta_1\ X\ \beta_2\ a\ \omega$ (definition)
    - $S' \Rightarrow^*_{rm} \alpha\ A\ a\ \omega \Rightarrow_{rm} \alpha\ \beta_1\ X\ \beta_2\ a\ \omega$ (definition)
    - $[A \to \beta_1\ X \bullet \beta_2\ , a]$ is valid for $\alpha\beta_1 X$ (definition)

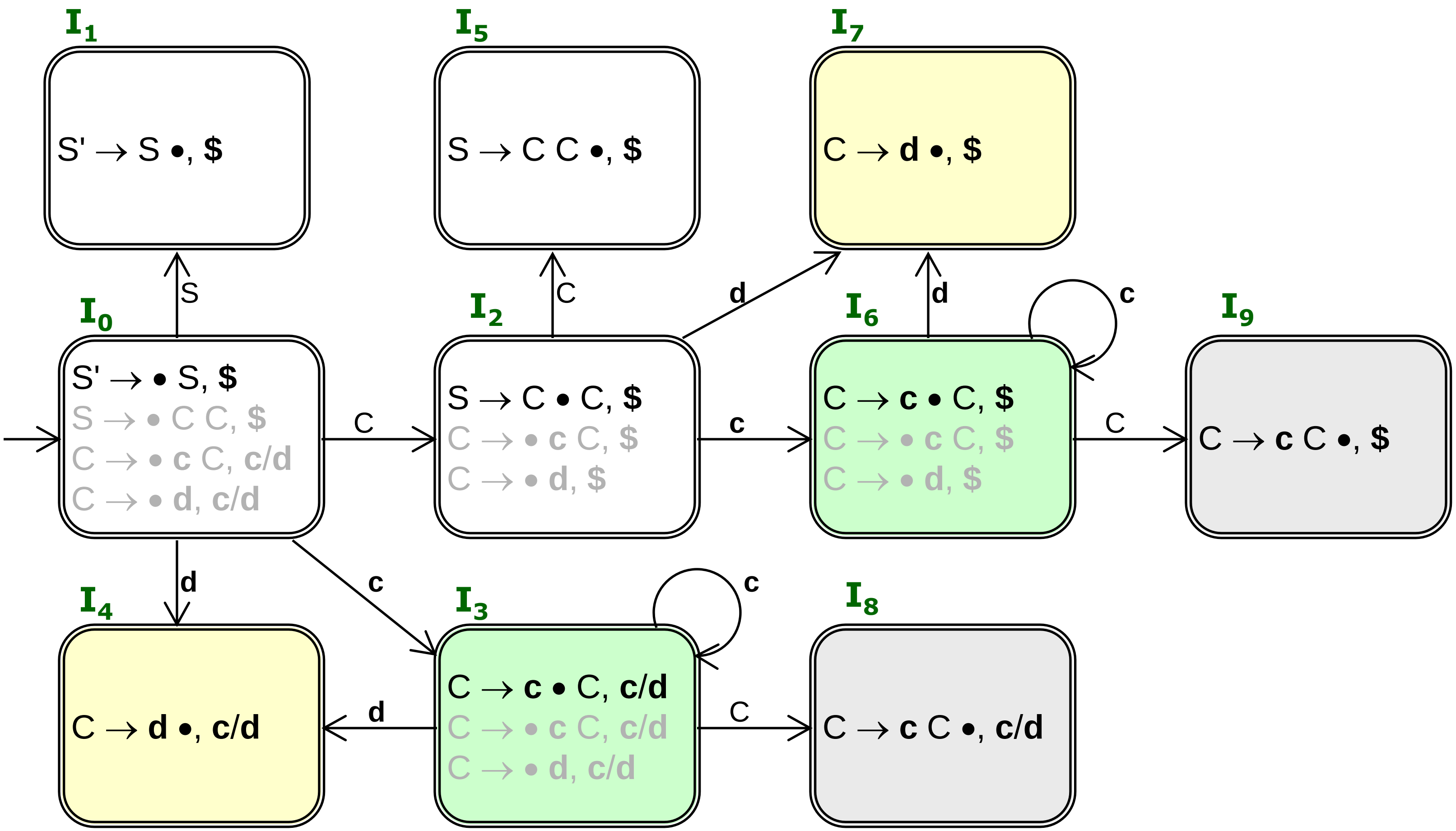
An Example

- Consider the following grammar

$$
\begin{array}{llll}
(0) & S' & \rightarrow & S \\
(1) & S & \rightarrow & C\ C \\
(2) & C & \rightarrow & \mathbf{c}\ C \\
(3) & C & \rightarrow & \mathbf{d}
\end{array}
$$

FIRST() and FOLLOW() Sets

- It is easy to calculate the following sets:
  - FIRST(S) = FIRST(C) = { **c**, **d** }
  - FOLLOW(S) = { **$** }
  - FOLLOW(C) = { **c**, **d**, **$** }



DFA recognizing all viable prefixes

LR(1) Parsing Table

State	ACTION			GOTO	
	c	**d**	**$**	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

LALR(1) Parsing Table

- Lookahead-LR parsing in practice
  - Number of states, such as C or Pascal
    - LR(0) = SLR(1): several hundreds.
    - LR(1): several thousands (10 times).
  - Strategy for LALR(1): merge the states with the same **core**.
    - E.g. I_3 and I_6, I_4 and I_7, I_8 and I_9
    - Lookaheads of the same item are merged.
    - GOTO() depends only on the core.

LALR(1) vs. LR(1)

- The merge will never produce **new** shift-reduce conflicts
  - Suppose in the merged state
    - [A $\rightarrow \alpha \bullet$, a] calls for a reduction
    - [B $\rightarrow \beta \bullet$ a γ, b] calls for a shift
  - Since the original states have the same core, there must be some state have
    - [A $\rightarrow \alpha \bullet$, a] calls for a reduction
    - [B $\rightarrow \beta \bullet$ a γ, c] calls for a shift (for some c)
  - Then the original state already has conflicts.

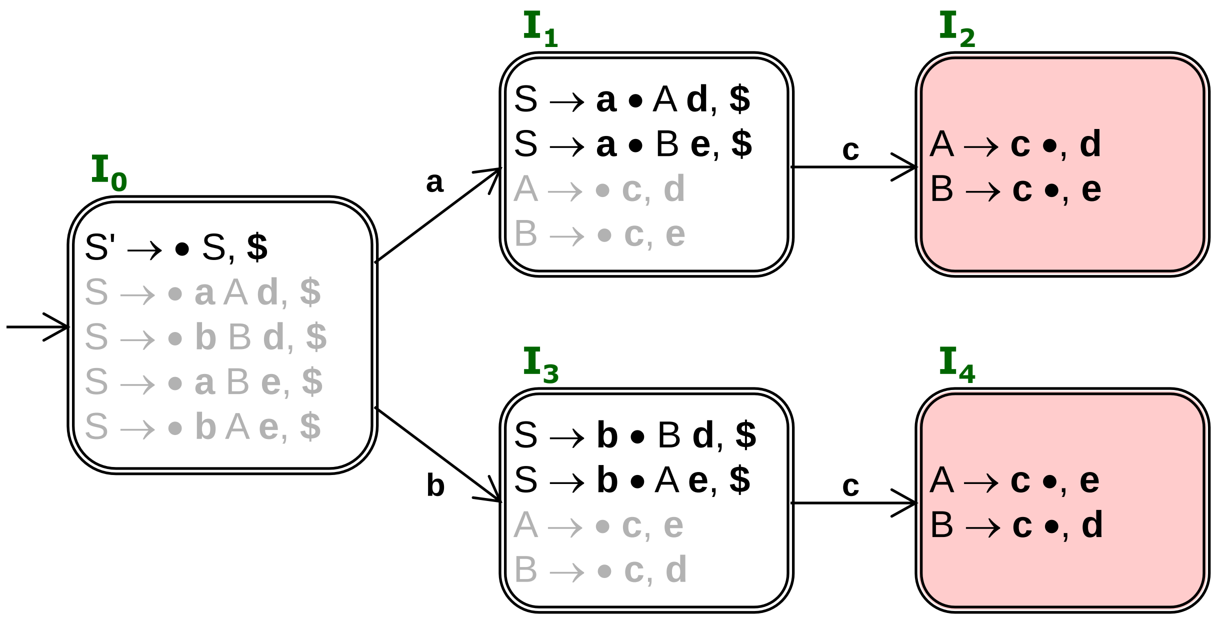
LALR(1) vs. LR(1) (cont')

- But the merge will produce **new** reduce-reduce conflicts
  - For example

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow \mathbf{a}\, A\, \mathbf{d} \mid \mathbf{b}\, B\, \mathbf{d} \mid \mathbf{a}\, B\, \mathbf{e} \mid \mathbf{b}\, A\, \mathbf{e} \\
A &\rightarrow \mathbf{c} \\
B &\rightarrow \mathbf{c}
\end{aligned}
$$

  - {[A → **c** •, **d**], [B → **c** •, **e**]} is valid for viable prefix ac, {[A → **c** •, **e**], [B → **c** •, **d**]} is valid for viable prefix bc. But the merge has conflicts:
    - {[A → **c** •, **d**/**e**], [B → **c** •, **d**/**e**]}

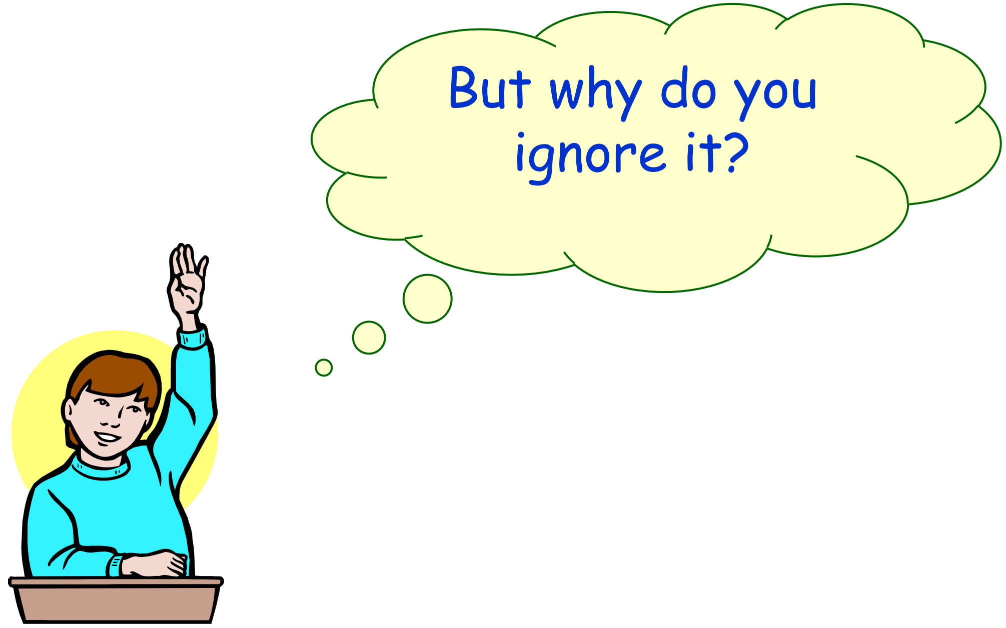
Example: New Conflicts in LALR(1)



Part of the DFA recognizing all viable prefixes

More Efficient Algorithm to Construct LALR(1) Parsing Tables

- Just feel free to ignore it.



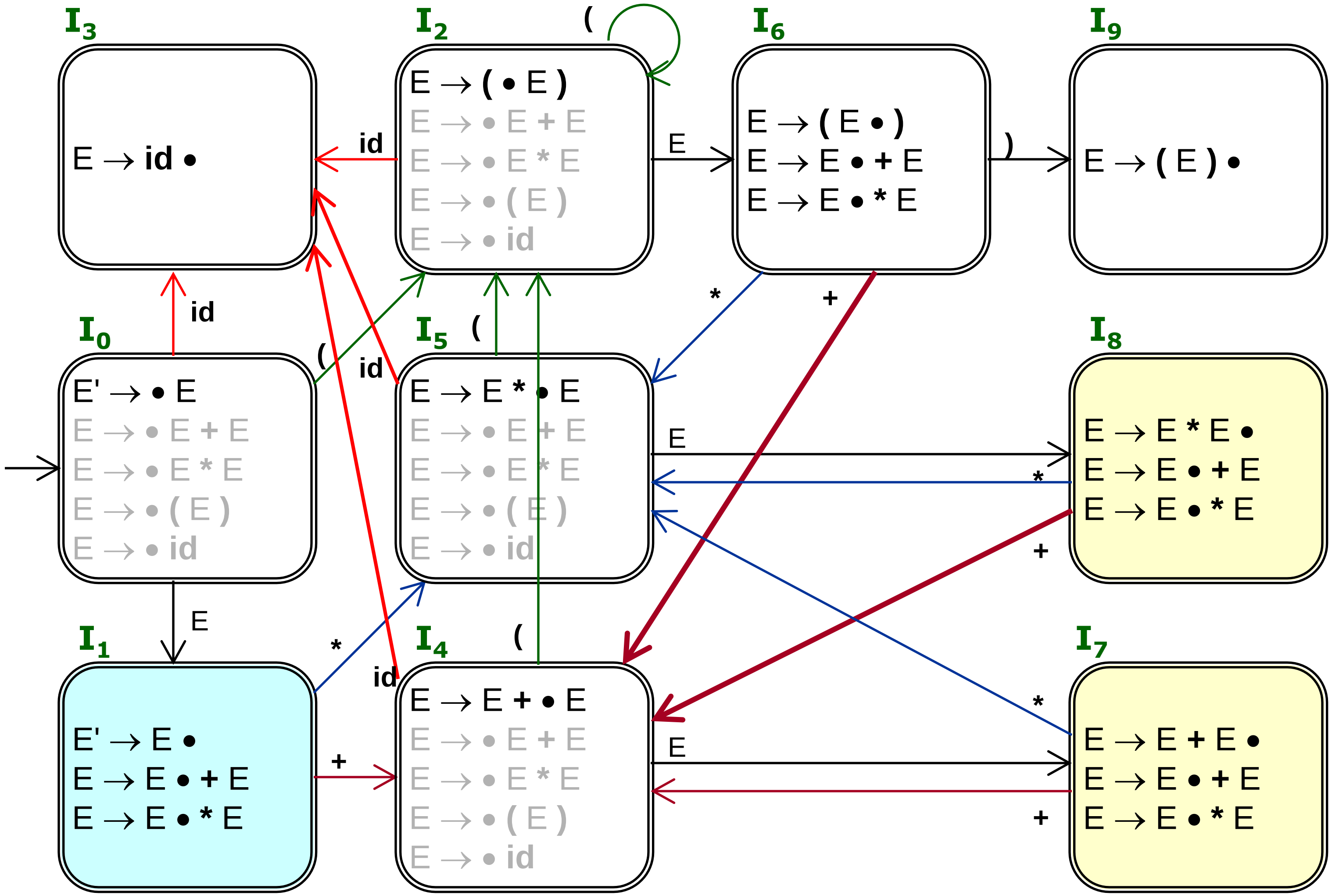
5. Ambiguity in LR Parsing

- Trade-off and consequence for ambiguity in LR parsing
  - Resolving ambiguities at the grammar level
    - Pros and cons?
  - Resolving ambiguities at the parsing table level
    - Pros and cons?
  - Resolving ambiguities at the source code level
    - Pros and cons?

Ambiguous Expression Grammar

- Given the ambiguous grammar

$$
\begin{array}{lllll}
(0) & E' & \rightarrow & E \\
(1) & E & \rightarrow & E \mathbf{+} E \\
(2) & E & \rightarrow & E \mathbf{*} E \\
(3) & E & \rightarrow & \mathbf{(}\ E\ \mathbf{)} \\
(4) & E & \rightarrow & \mathbf{id}
\end{array}
$$



DFA recognizing all viable prefixes

shift-reduce conflict

SLR(1) Parsing Table

State	ACTION						GOTO
	id	**+**	*****	**(**	**)**	**$**	E
0	s3			s2			1
1		s4	s5			acc	
2	s3			s2			6
3		r4	r4		r4	r4	
4	s3			s2			7
5	s3			s2			8
6		s4	s5		s9		
7		r1/s4	r1/s5		r1	r1	
8		r2/s4	r2/s5		r2	r2	
9		r3	r3		r3	r3	

Resolve ambiguities
at the parsing table level

Dangling-**else** Grammar

- Given the ambiguous grammar

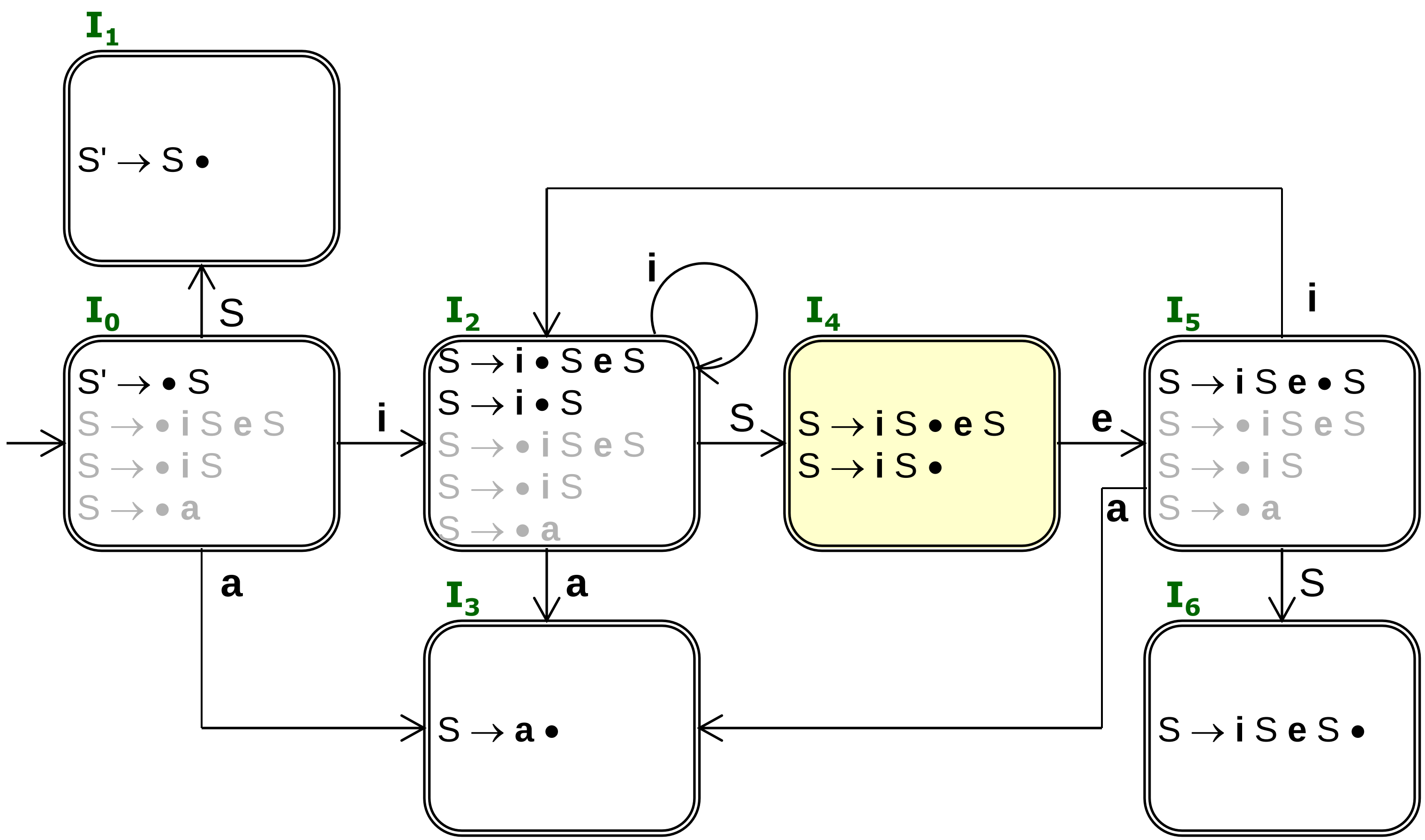
(0) $S' \rightarrow S$

(1) $S \rightarrow \mathbf{i}\ S\ \mathbf{e}\ S$

(2) $S \rightarrow \mathbf{i}\ S$

(3) $S \rightarrow \mathbf{a}$

DFA Recognizing All Viable Prefixes



shift-reduce conflict

SLR(1) Parsing Table

State	ACTION				GOTO
	i	**e**	**a**	**$**	S
0	s2		s3		1
1				acc	
2	s2		s3		4
3		r3		r3	
4		r2/s5		r2	
5	s2		s3		6
6		r1		r1	

Resolve ambiguities
at the parsing table level

Parsing a Sentence

Step	Symbol	State	Input	Reference	Action	Output
1	\$	0	**i i a e a \$**	a[0, **i**] = s2	shift	
2	\$ i	0 2	**i a e a \$**	a[2, **i**] = s2	shift	
3	\$ i i	0 2 2	**a e a \$**	a[2, **a**] = s3	shift	
4	\$ i i a	0 2 2 3	**e a \$**	a[3, **e**] = r3 g[2, S] = 4	reduce	S → **a**
5	\$ i i S	0 2 2 4	**e a \$**	a[4, **e**] = s5	shift	
6	\$ i i S e	0 2 2 4 5	**a \$**	a[5, **a**] = s3	shift	
7	\$ i i S e a	0 2 2 4 5 3	**\$**	a[3, **\$**] = r3 g[5, S] = 6	reduce	S → **a**
8	\$ i i S e S	0 2 2 4 5 6	**\$**	a[6, **\$**] = r1 g[2, S] = 4	reduce	S → **i** S **e** S
9	\$ i S	0 2 4	**\$**	a[4, **\$**] = r2 g[0, S] = 1	reduce	S → **i** S
10	\$ S	0 1	**\$**	a[1, **\$**] = acc	accept	

6. Error Recovery in LR Parsing

- For errors in an input sentence
  - All errors will be found by all LR parsers
    - If we can construct a parsing table without conflicts.
  - More reductions before reporting an error
    - LR(0) $\geq$ SLR(1) $\geq$ LALR(1) $\geq$ LR(1)
    - But all of them will never shift an erroneous input symbol onto the stack.

Review: Error Recovery Techniques

- Panic-Mode Error Recovery
  - Skip inputs until synchronizing token found.
- Phrase-Level Error Recovery
  - Assign each empty entry a specific error routine.
- Error-Productions
  - Suitable for common errors but not all errors.
- Global-Correction
  - Globally analyze the input to find the error.
  - Expensive and not in practice.

Panic-Mode Error Recovery

- Pop until a state **s** with a GOTO() on a particular nonterminal **A** is found.
  - Usually **A** represents major constructs, e.g. *stmt*, *expr*, or *block*.
  - It indicates that the construct A has errors.
- Push GOTO(**s**, **A**).
- 0 or more lookaheads are then discarded, until a symbol that can follow **A**.

It is easier to understand if we look from the viewpoint of DFA

Phrase-Level Error Recovery

- Examine each empty entry, and assign it a pointer to a specific error-handling routine.
- Ad hoc: depending on the usage of the language.

The Previous Parsing Table

State	ACTION						GOTO
	id	**+**	*****	**(**	**)**	**$**	E
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

Postpone error detection until one or more reductions are made

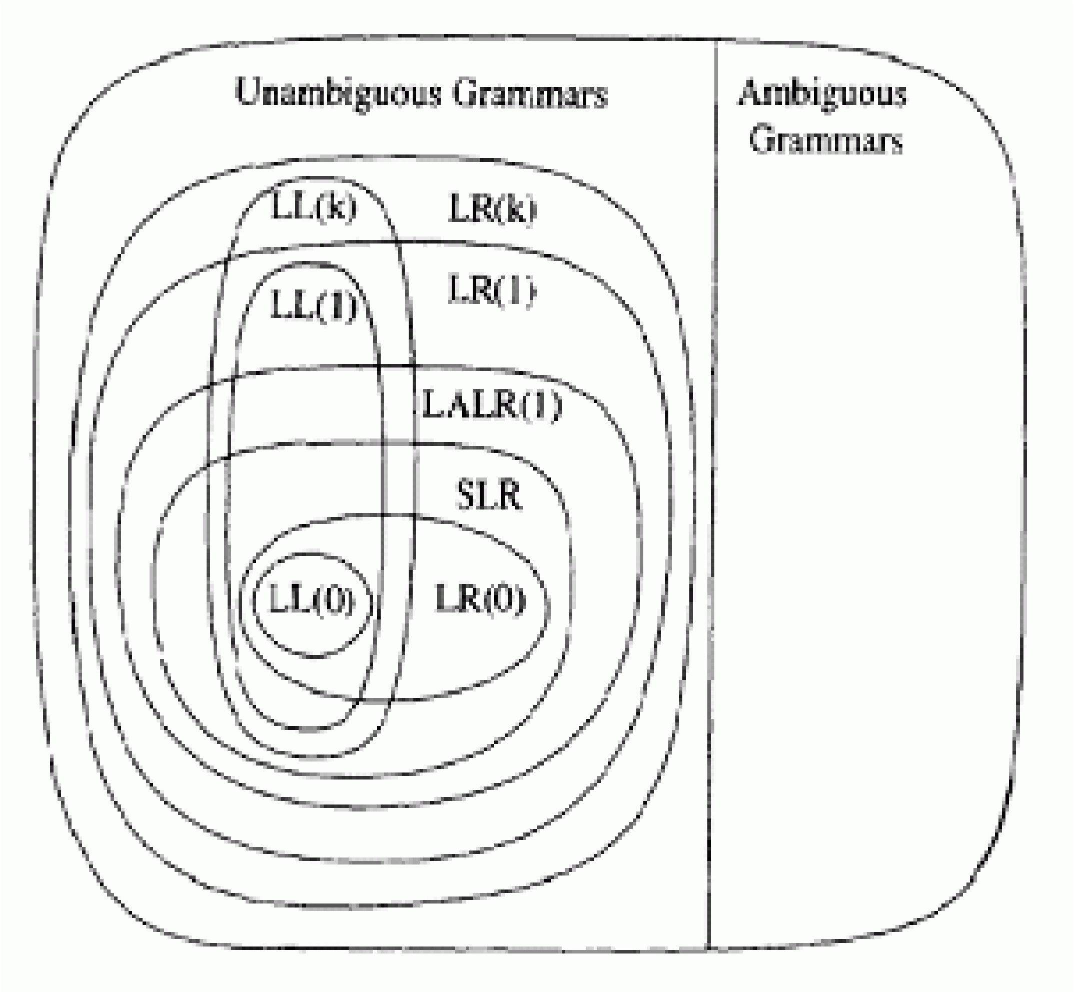
Error-Handling Routines

- e1: an operand ('id' or '(') is expected.
  - `push state 3;  // add a symbol 'id'`
- e2: unbalanced right parenthesis.
  - `drop one lookahead;  // remove ')'`
- e3: an operator is expected.
  - `push state 4;  // add a symbol '+'`
- e4: a right parenthesis is expected.
  - `push state 9;  // add a symbol ')'`

Parsing an Erroneous Input

Step	Symbol	State	Input	Reference	Action	Output
1	$	0	**id +) $**	a[0, **id**] = s3	shift	
2	$ id	0 3	**+) $**	a[3, **+**] = r4 g[0, E] = 1	reduce	E → **id**
3	$ E	0 1	**+) $**	a[1, **+**] = s4	shift	
4	$ E +	0 1 4	**) $**	a[4, **)**] = e2	drop	Unbalanced ')'
5	$ E +	0 1 4	**$**	a[4, **$**] = e1	push 3	Operand expected
6	$ E + **id**	0 1 4 3	**$**	a[3, **$**] = r4 g[4, E] = 7	reduce	E → **id**
7	$ E + E	0 1 4 7	**$**	a[7, **$**] = r1 g[0, E] = 1	reduce	E → E **+** E
8	$ E	0 1	**$**	a[1, **$**] = acc	end	

Conclusions:
Context-Free Grammar Classification



Exercise 7.1

- Consider the grammar

$$S \rightarrow \mathbf{(}\, S\, R \mid \mathbf{a}$$

$$R \rightarrow \mathbf{,}\, S\, R \mid \mathbf{)}$$

  - Try to construct an SLR(1) parsing table for the grammar, and see if there are conflicts in the parsing table.

Exercise 7.2

- Consider the grammar

$$S \rightarrow S\,\mathbf{a}\,\mathbf{b} \mid \mathbf{a}\,R$$

$$R \rightarrow S \mid \mathbf{a}$$

Is the grammar an SLR(1) grammar? and why?

Exercise 7.3

- Consider the grammar

$$S \rightarrow A$$
$$A \rightarrow B\,A \mid \varepsilon$$
$$B \rightarrow \mathbf{a}\,B \mid \mathbf{b}$$

  - Prove that the grammar is an LR(1) grammar.
  - Construct an LR(1) parsing table for the grammar.
  - Show the detailed parsing procedure for the sentence **abab**, following the style in slides of this lecture.

Exercise 7.4*

- (DBv2, pp.278, ex.4.7.4) Show that the grammar

  $$S \rightarrow A\,\mathbf{a} \mid \mathbf{b}\,A\,\mathbf{c} \mid \mathbf{d}\,\mathbf{c} \mid \mathbf{b}\,\mathbf{d}\,\mathbf{a}$$

  $$A \rightarrow \mathbf{d}$$

  is LALR(1) but not SLR(1).

Further Reading

- Dragon Book, 2nd Edition (DBv2)
  - Comprehensive Reading:
    - Section 4.6 on SLR(1) parsing.
    - Section 4.7 on LR(1) and LALR(1) parsing.
    - Section 4.8.1-4.8.2 on ambiguities in LR parsing.
  - Skip Reading:
    - Section 4.8.3 on error recovery in LR parsing.

Enjoy the Course!