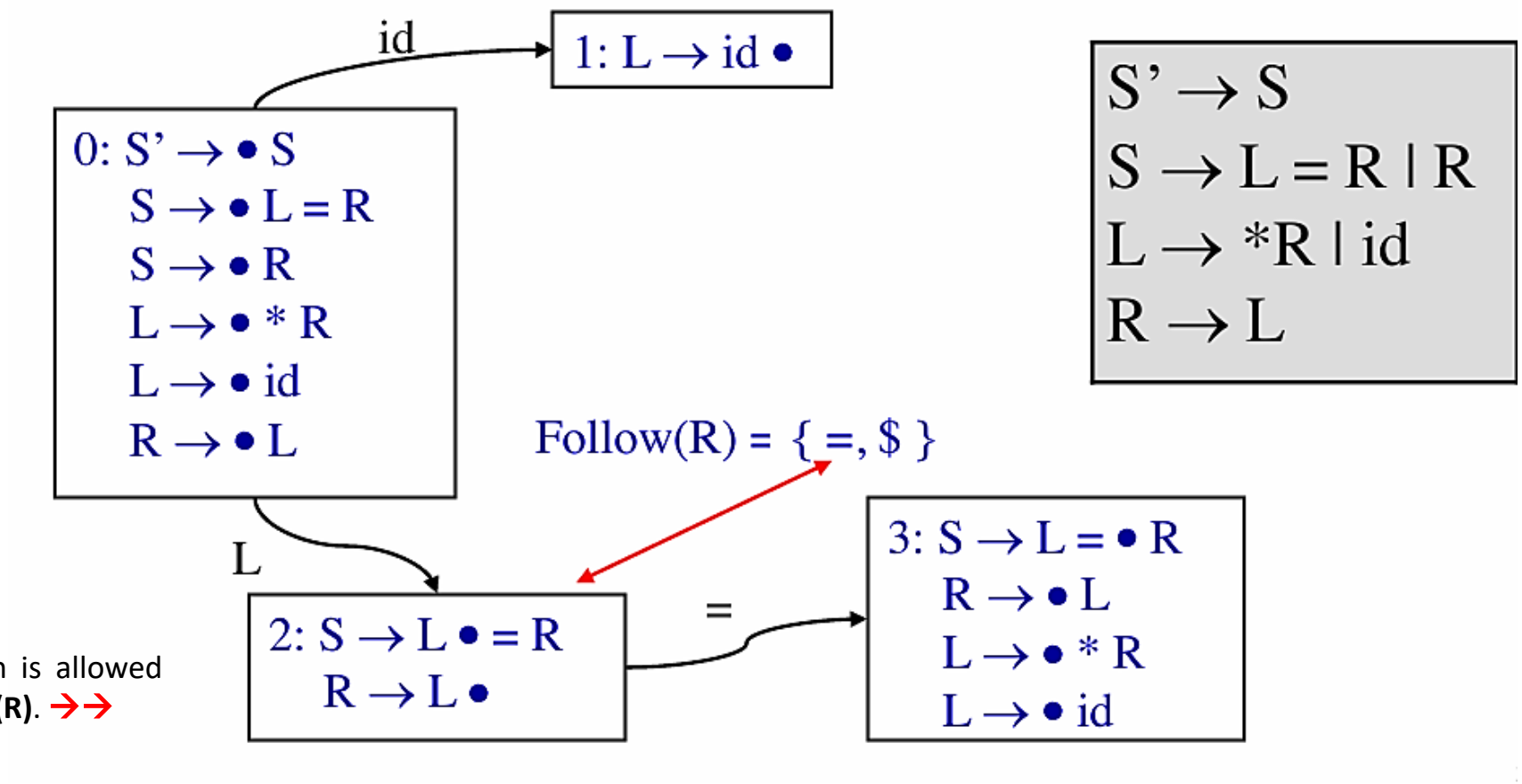
IT 302 Compiler Design

Canonical LR Parsing

SLR limitation: lack of context



In SLR parsing, Reduction is allowed on all symbols in **FOLLOW(R)**. →→

The problem (lack of context)

- In context of S → L = R, if lookahead is **=**, the parser should **shift**, not reduce.
- But SLR only looks at FOLLOW sets, not the *specific item context*.
- Since = is in FOLLOW(R), SLR wrongly allows reduction, creating a **shift/reduce conflict**.

Why LR(1) solves it

- In LR(1), items carry lookahead symbols. LR (1) = LR (0) Item + Lookahead Symbols
- So instead of just R → L •, we'd have:
  - R → L •, $ (reduce only on $)
  - R → L •, = would not exist in this state, because in the context of S → L = R, reduction on = is invalid.
- This disambiguates the decision:
  - On **=**, shift.
  - On **$**, reduce.

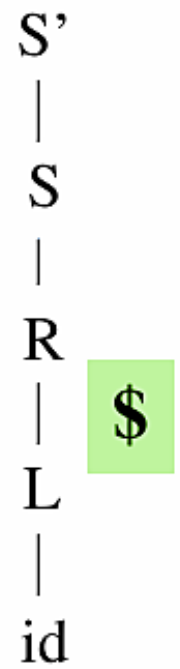
- Note:
  - SLR limitation: it uses FOLLOW sets without context, so reductions may be applied in situations where they are not valid, leading to false conflicts.
  - LR(1) overcomes this by attaching the *precise lookahead* to each item, **preserving context**.

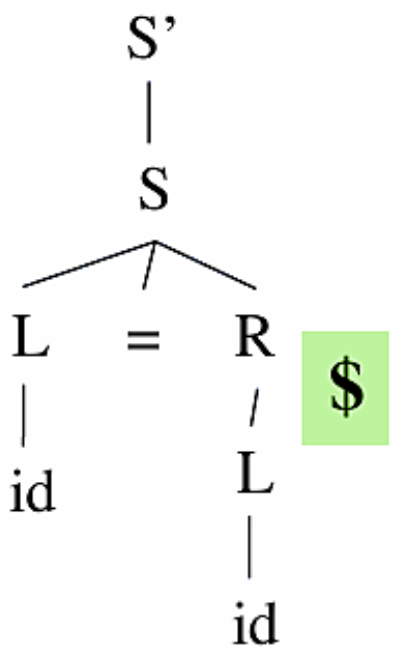
$S' \rightarrow S$
$S \rightarrow L = R \mid R$
$L \rightarrow *R \mid id$
$R \rightarrow L$

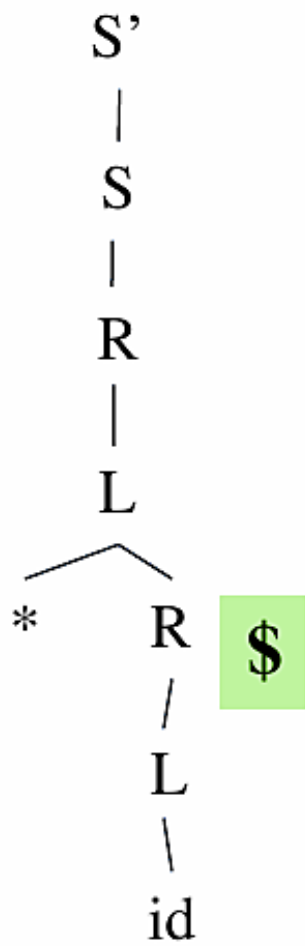
Follow(R) = { =, $ }

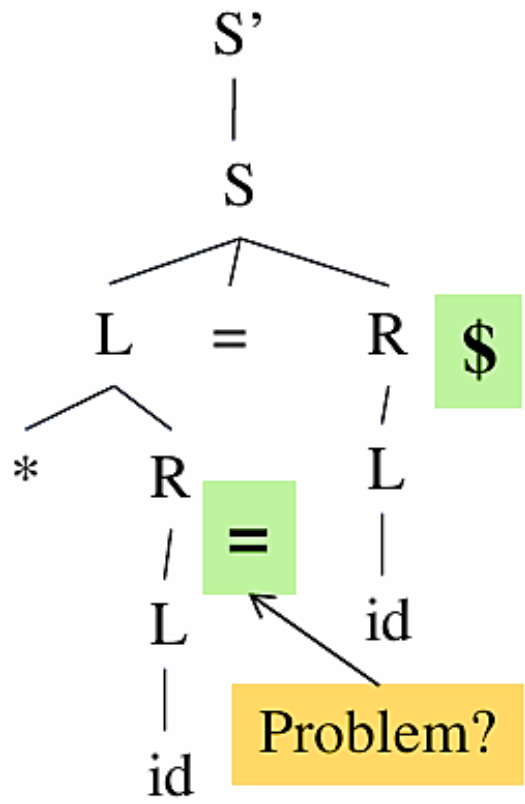
2: $S \rightarrow L \bullet = R$
$R \rightarrow L \bullet$

Find all lookaheads for reduce $R \rightarrow L \bullet$









No! $R \rightarrow L \bullet$ reduce and $S \rightarrow L \bullet = R$ do not co-occur due to the $L \rightarrow *R$ rule

Solution: Canonical LR(1)

- Extend the definition of configuration
  - Remember lookahead
- New closure method
- Extend the definition of Successor

Extend the definition of configuration

- In LR(0)/SLR, a configuration (item) looks like: [A → α • β]
- In LR(1), we extend it to: [A → α • β, a]
  - where a is a lookahead terminal that is expected to follow A.
- So, each item carries both the parsing progress (•) and a lookahead context.

$[A \rightarrow \alpha \bullet \beta, a]$ for $a \in T$ is valid for a viable prefix $\delta\alpha$ if there is a rightmost derivation

$S \Rightarrow^* \delta A\eta \Rightarrow^* \delta\alpha\beta\eta$ and $(\eta = a\gamma)$ or $(\eta = \varepsilon$ and $a = \$)$

where:

- $A \rightarrow \alpha\beta$ is a production.
- • marks the current parsing position in the right-hand side.
- $a \in T \cup \{\$\}$ is the lookahead symbol → the terminal that is expected to follow A.

So, it encodes both progress in parsing and what we expect next.

Contd., Remember lookahead

- Validity of a Configuration

$[A \rightarrow \alpha \bullet \beta, a]$ for $a \in T$ is valid for a viable prefix $\delta\alpha$ if there is a rightmost derivation

$S \Rightarrow^* \delta A \eta \Rightarrow^* \delta\alpha\beta\eta$ and $(\eta = a\gamma)$ or $(\eta = \varepsilon$ and $a = \$)$

- Start from the augmented start symbol S'.
- There exists a rightmost derivation where:
  1. $S \Rightarrow^* \delta A h$ (we are about to expand A).
  2. Then $A \Rightarrow \alpha \beta$, so $S \Rightarrow^* \delta \alpha \beta \eta$.
  3. The lookahead condition:
     - If $\eta = a\gamma$ → the first symbol of the suffix is a.
     - If $\eta = \varepsilon$ (we're at the end), then $a = \$$.

Notation: $[A \rightarrow \alpha \bullet \beta, a/b/c]$

- if $[A \rightarrow \alpha \bullet \beta, a]$, $[A \rightarrow \alpha \bullet \beta, b]$, $[A \rightarrow \alpha \bullet \beta, c]$ are valid configurations

Points to be Noted

- LR(1) item = production + dot position + lookahead symbol.
- An item [A → α • β, a] is valid if it matches a rightmost derivation where a is the actual next terminal (or $ at the end).
- Multiple lookaheads can be grouped as [A → α • β, a/b/c].
- *This precise context handling is why LR(1) parsers are more powerful than SLR(1).*

LR(1) Closure: The new Closure Method

- Closure property: If [A → a•B, a] is in set, then [B →• γ, b] is in set if b € First(a)

1st : [A→α•Bβ,a] means: we are in the middle of parsing B in production A → αBβ, with lookahead a.

2nd: What does Closure mean? If you expect a B next, you must also consider all ways of deriving B.

So for every production B→γ, add: [B→•γ, b]

This is the key LR(1) difference from SLR/CLR:

- Lookahead set = First(β a), where
  - β is what follows B in the original item,
  - a is the original lookahead.

Example:

S → A a
A → B
B → b

→→ Closure

[S → • A a, $]
[A → • B, a]
[B → • b, a]

Start item: [S→•Aa,$]

By closure:

- Since dot before A, add [A → • B, a] because First(a $) = {a}.
- Since dot before B, add [B → • b, a] because First(a) = {a}.

Starting Configuration

- Augment Grammar with S’ just like for LR(0), SLR(1)

- Initial configuration set is

$$I = \text{closure}([S' \rightarrow \bullet\, S, \$])$$

Example: closure([S' → • S, $])

$S' \rightarrow S$
$S \rightarrow L = R \mid R$
$L \rightarrow *R \mid id$
$R \rightarrow L$

[S' → • S, $]
[S → • L = R, $]
[S → • R, $]
[L → • * R, =]
[L → • id, =]
[R → • L, $]
[L → • *R, $]
[L → • id, $]

concisely written as:

S' → • S, $
S → • L = R, $
S → • R, $
L → • * R, =/$
L → • id, =/$
R → • L, $

LR(1) Successor (I, X) or GOTO Function

- Let I = [A → α•Bβ, a] **or** [A → α•bβ, a]
- Successor(I, B)

  = closure([A → αB • β, a])
- Successor(I, b)

  = closure([A → αb • β, a])

Usually, for terminals, closure doesn't add new items since terminals don't expand

Example:

S → A
A → a B
B → b

Initial Closure (I0):

[S → • A, $]
[A → • a B, $]

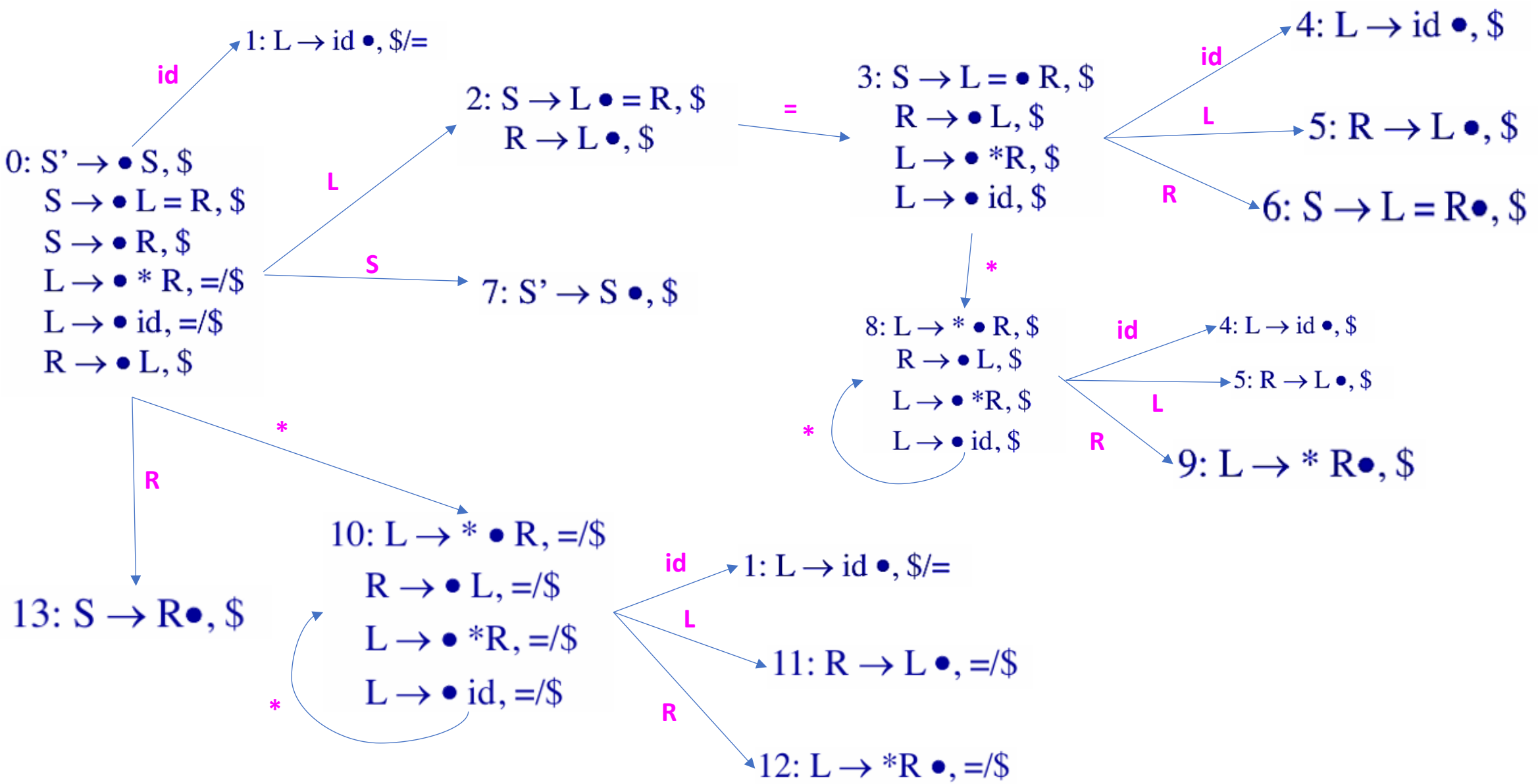
Now compute Successor(I0, A):

- Shift dot: [S → A •, $]
- Closure adds nothing (dot not before nonterminal).
  So, Successor(I0, A) = {[S → A •, $]}

Compute Successor(I0, a):

- From [A → • a B, $], shift dot: [A → a • B, $]
- Closure: dot before B, so add [B → • b, $]
  So, Successor(I0, a) = {[A → a • B, $],
  [B → • b, $]}

LR(1) Example


0: S' → • S, $
S → • L = R, $
S → • R, $
L → • * R, =/$
L → • id, =/$
R → • L, $
id
1: L → id •, $/=
L
2: S → L • = R, $
R → L •, $
=
3: S → L = • R, $
R → • L, $
L → • *R, $
L → • id, $
id
4: L → id •, $
L
5: R → L •, $
R
6: S → L = R•, $
S
7: S' → S •, $
*
8: L → * • R, $
R → • L, $
L → • *R, $
L → • id, $
*
id
4: L → id •, $
L
5: R → L •, $
R
9: L → * R•, $
R
13: S → R•, $
*
10: L → * • R, =/$
R → • L, =/$
L → • *R, =/$
L → • id, =/$
*
id
1: L → id •, $/=
L
11: R → L •, =/$
R
12: L → *R •, =/$

Productions	
1	S → L = R
2	S → R
3	L → * R
4	L → id
5	R → L

	id	=	*	$	S	L	R
0	S1		S10		7	2	13
1		R4		R4			
2		S3		R5			
3	S4		S8			5	6
4				R4			
5				R5			
6				R1			
7				Acc			
8	S4		S8			5	9
9				R3			
10	S1		S10			11	12
11		R5		R5			
12		R3		R3			
13				R2			

Example 1: do the rest

S → aAd | bBd | aBe | bAe
A→ c
B → c

S’ → .S, $
S → . aAd, $
S → .bBd, $
S → .aBe, $
S → .bAe, $

LR(1) Construction

1. Construct F = $\{I_0, I_1, \ldots I_n\}$
2. a) if $[A \rightarrow \alpha\bullet, a] \in I_i$ and A != S'

   then action[i, a] := reduce $A \rightarrow \alpha$

   b) if $[S' \rightarrow S\bullet, \$] \in I_i$

   then action[i, $] := accept

   c) if $[A \rightarrow \alpha\bullet a\beta, b] \in I_i$ and Successor(I_i, a)=I_j
   then action[i, a] := shift j
3. if Successor(I_i, A) = I_j then goto[i, A] := j

LR(1) Construction (cont'd)

4. All entries not defined are errors
5. Make sure I_0 is the initial state

- Note: LR(1) only reduces using $A \rightarrow \alpha$ for

  $[A \rightarrow \alpha\bullet, a]$ if a is the next input symbol
- LR(1) states remember context by virtue of lookahead
- Possibly many more states than LR(0) due to the lookahead!
  - LALR(1) combines some states

$S \rightarrow AaAb$
$S \rightarrow BbBa$
$A \rightarrow \varepsilon$
$B \rightarrow \varepsilon$

Q: Write down the LR(1) automaton and parse table for the above grammar. Is it an LR(1) grammar?

LR(1) Conditions

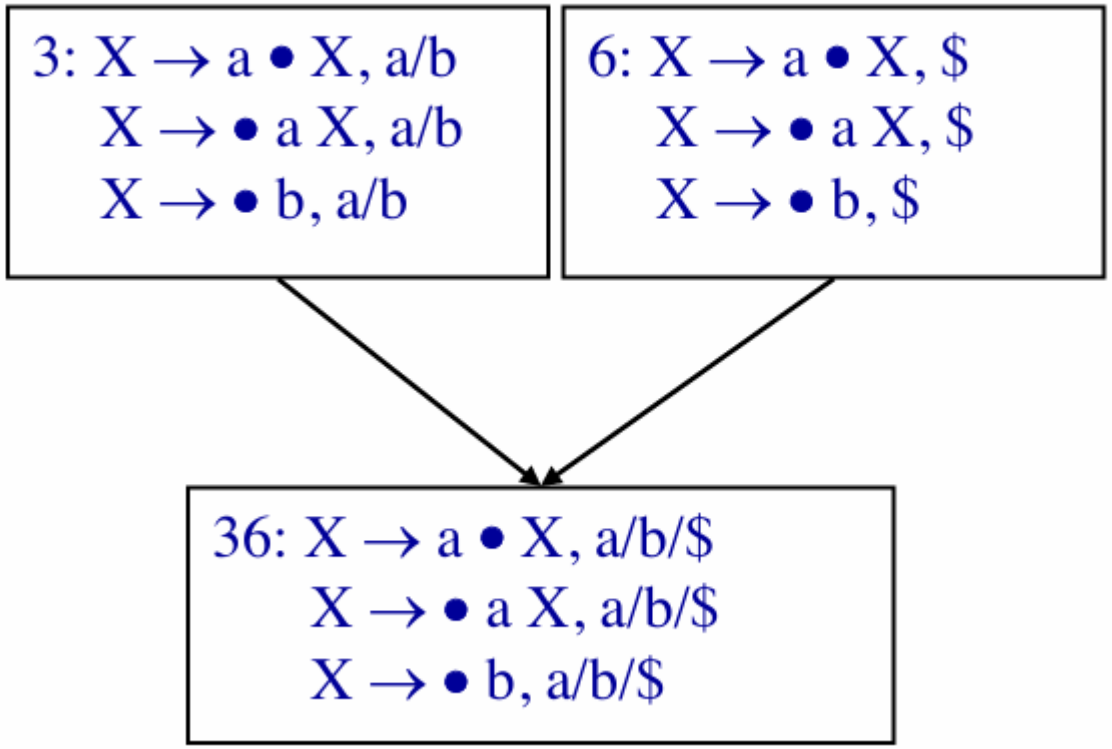
- A grammar is LR(1) if for each configuration set (itemset) the following holds:
  - For any item $[A \rightarrow \alpha \bullet x \beta, a]$ with $x \in T$ there is no $[B \rightarrow \gamma \bullet, x]$
  - For any two complete items $[A \rightarrow \gamma \bullet, a]$ and $[B \rightarrow \beta \bullet, b]$ then a != b.
- Grammars:
  - $LR(0) \subset SLR(1) \subset LR(1) \subset LR(k)$
- Languages expressible by grammars:
  - $LR(0) \subset SLR(1) \subset LR(1) = LR(k)$

Canonical LR(1) Recap

- LR(1) uses left context, current handle, and lookahead to decide when to reduce or shift
- Most powerful parser so far (can handle more context-free grammars)
- LALR(1) is a practical simplification with fewer states used by yacc/bison to avoid the very large tables generated by LR(1)

Merging States in LALR(1)

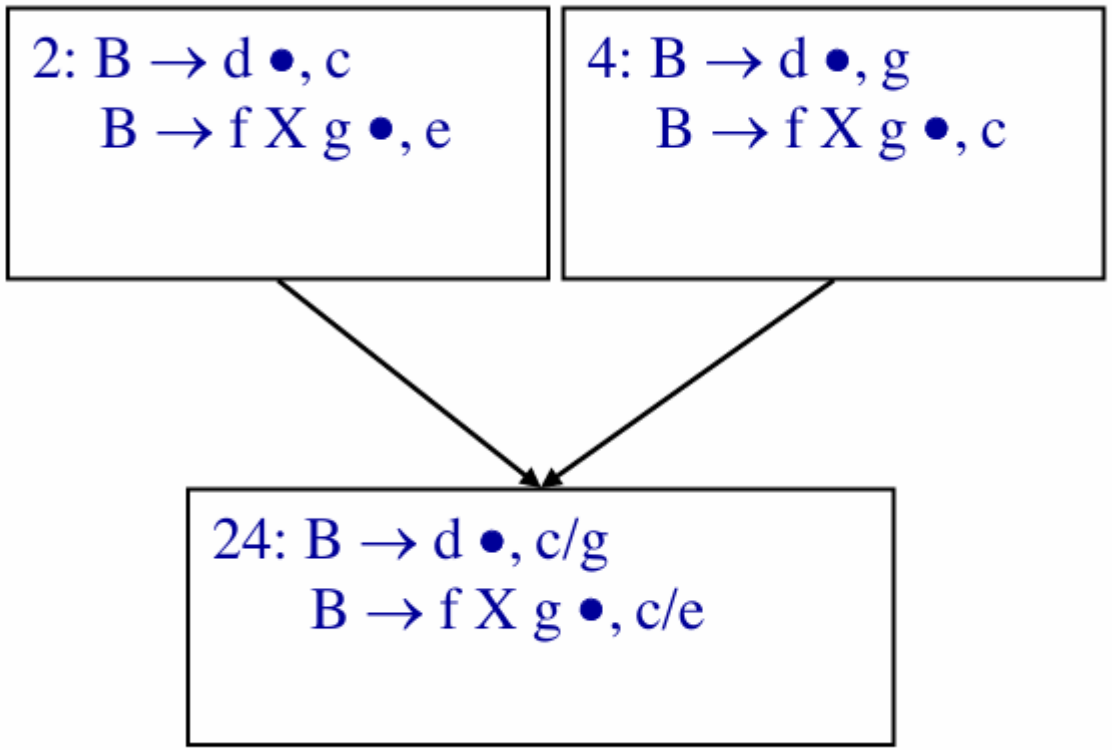
- $S' \rightarrow S$
  $S \rightarrow XX$
  $X \rightarrow aX$
  $X \rightarrow b$
- Same **Core Set**
- Different lookaheads



R/R conflicts when merging

- $B \rightarrow d$
  $B \rightarrow f\ X\ g$
  $X \rightarrow \ldots$

- If R/R conflicts are introduced, grammar is not LALR(1)!



LALR(1)

- LALR(1) Condition:
  - Assumption: merging does not introduce reduce/reduce conflicts
  - Shift/reduce cannot be introduced
- Merging brute force or step-by-step
- More compact than canonical LR, like SLR(1)
- More powerful than SLR(1)
  - Not always merge to full Follow Set

Set of items with Epsilon rules