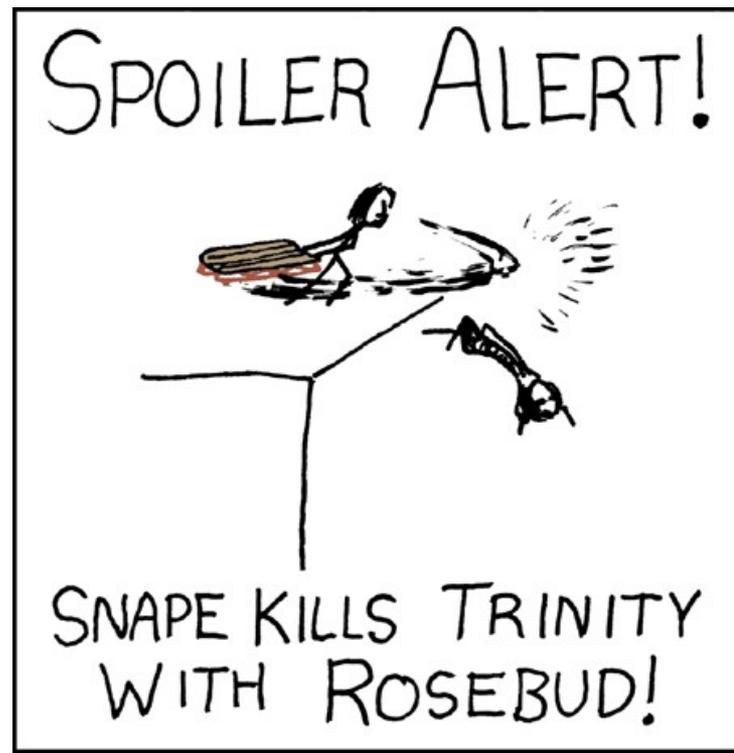


LR Parsing

Table Construction

Outline

- Review of bottom-up parsing
- Computing the parsing DFA
 - Closures, LR(1) Items, States
 - Transitions
- Using parser generators
 - Handling Conflicts



In One Slide

- An **LR(1) parsing table** can be constructed automatically from a CFG. An **LR(1) item** is a pair made up of a **production** and a **lookahead** token; it represents a possible **parser context**. After we **extend** LR(1) items by **closing** them they become LR(1) **DFA states**. Grammars can have **shift/reduce** or **reduce/reduce conflicts**. You can fix most conflicts with **precedence** and **associativity** declarations. **LALR(1) tables** are formed from LR(1) tables by **merging** states with similar **cores**.

Bottom-up Parsing (Review)

- A bottom-up parser rewrites the input string to the start symbol
- The state of the parser is described as

$$\alpha \blacktriangleright \gamma$$

- α is a **stack** of terminals and non-terminals
 - γ is the string of terminals not yet examined
-
- Initially: $\blacktriangleright x_1 x_2 \cdot \cdot \cdot x_n$

Shift and Reduce Actions (Review)

- Recall the CFG: $E \rightarrow \text{int} \mid E + (E)$
- A bottom-up parser uses two kinds of actions:
- **Shift** pushes a **terminal** from input on the **stack**
$$E + (\blacktriangleright \text{int}) \Rightarrow E + (\text{int} \blacktriangleright)$$
- **Reduce** pops 0 or more symbols off of the stack (production RHS) and pushes a non-terminal on the stack (production LHS)

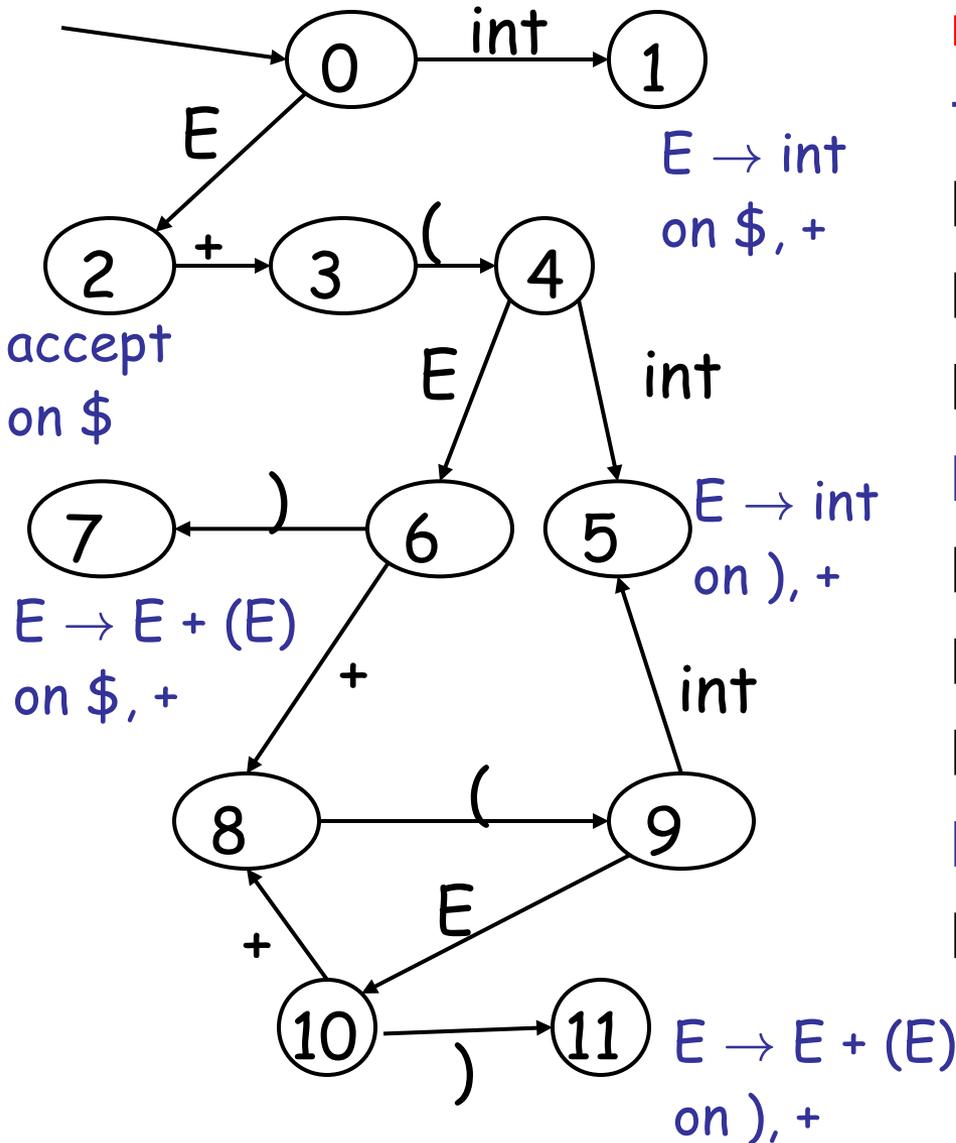
$$E + (\underline{E + (E)} \blacktriangleright) \Rightarrow E + (\underline{E} \blacktriangleright)$$

Key Issue:

When to Shift or Reduce?

- Idea: use a finite automaton (DFA) to decide when to shift or reduce
 - The **input is the stack**
 - The language consists of terminals and non-terminals
- We run the DFA on the stack and we examine the resulting state X and the token tok after 
 - If X has a transition labeled tok then **shift**
 - If X is labeled with “ $A \rightarrow \beta$ on tok ” then **reduce**

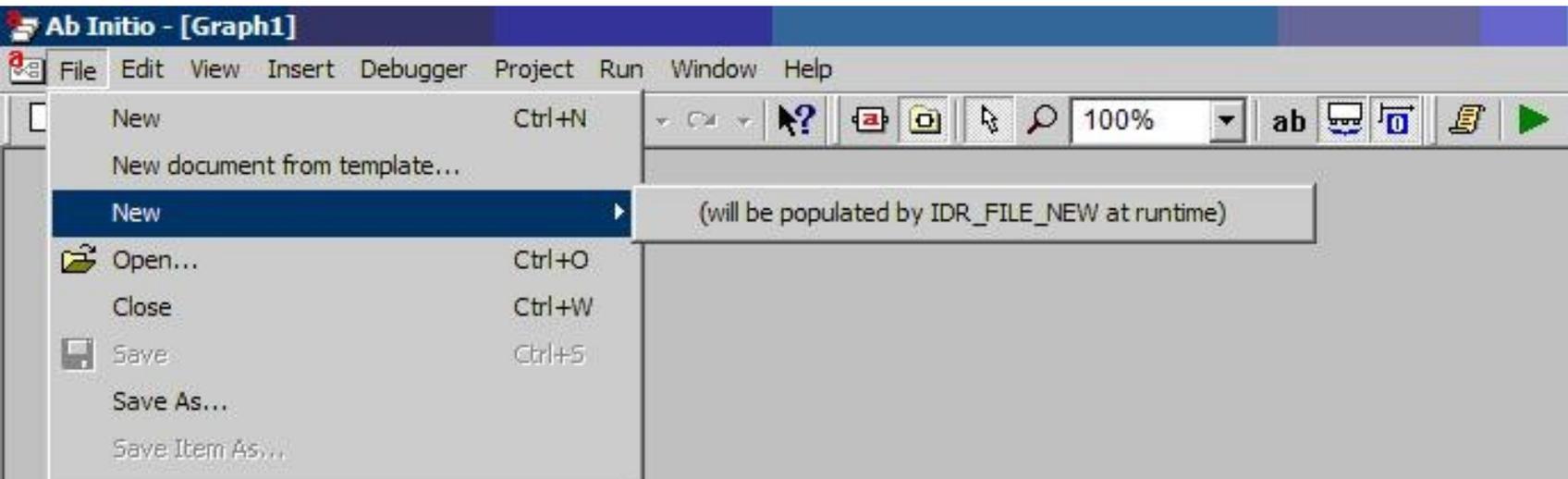
LR(1) Parsing. An Example

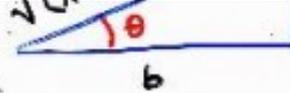


- ▶ $\text{int} + (\text{int}) + (\text{int})\$$ shift
- int ▶ $+ (\text{int}) + (\text{int})\$$ $E \rightarrow \text{int}$
- E ▶ $+ (\text{int}) + (\text{int})\$$ shift(x3)
- $E + (\text{int} \blacktriangleright) + (\text{int})\$$ $E \rightarrow \text{int}$
- $E + (E \blacktriangleright) + (\text{int})\$$ shift
- $E + (E) \blacktriangleright + (\text{int})\$$ $E \rightarrow E + (E)$
- $E \blacktriangleright + (\text{int})\$$ shift (x3)
- $E + (\text{int} \blacktriangleright)\$$ $E \rightarrow \text{int}$
- $E + (E \blacktriangleright)\$$ shift
- $E + (E) \blacktriangleright \$$ $E \rightarrow E + (E)$
- $E \blacktriangleright \$$ accept

Key Issue: How is the DFA Constructed?

- The **stack** describes the **context** of the parse
 - What non-terminal we are looking for
 - What production rhs we are looking for
 - What we have seen so far from the rhs





$$\int \frac{b dx}{(x-a)^2 + b^2} = \int \frac{dx}{b} \cos^2 \theta = \int d\theta$$

$$\left. \begin{aligned} d(\tan \theta) &= \sec^2 \theta d\theta = \frac{d\theta}{\cos^2 \theta} \\ &= d\left(\frac{x-a}{b}\right) = \frac{dx}{b} \end{aligned} \right\} \frac{dx}{b} = \frac{d\theta}{\cos^2 \theta}$$

$$\int \frac{b dx}{(x-a)^2 + b^2} = \int d\left[\tan^{-1}\left(\frac{x-a}{b}\right)\right]$$

The integral is bounded from below, but not from above

$$\int_0^{\infty} \frac{b dx}{(x-a)^2 + b^2} = \int_{x=0}^{x=\infty} d\left[\tan^{-1}\left(\frac{x-a}{b}\right)\right] = \int_{\xi=-\frac{a}{b}}^{\xi=\infty} d(\tan^{-1} \xi)$$

ξ
 $-\infty$ for $b < 0$

$$= \tan^{-1}(\infty) - \tan^{-1}(-\infty) = \pi$$

LR(1) Table Construction

Three hours later, you can finally parse $E \rightarrow E + E \mid \text{int}$

Parsing Contexts

- Consider the state:

$$\begin{array}{c} E \\ / \\ \text{int} + (\text{int}) + (\text{int}) \end{array}$$
 - The stack is

$$E + (\blacktriangleright \text{int}) + (\text{int})$$
- Context:
 - We are looking for an $E \rightarrow E + (\bullet E)$
 - Have have seen $E + ($ from the right-hand side
 - We are also looking for $E \rightarrow \bullet \text{int}$ or $E \rightarrow \bullet E + (E)$
 - Have seen nothing from the right-hand side
- One DFA state describes several contexts

Red dot =
where we are.

LR(1) Items

- An LR(1) item is a pair:

$$X \rightarrow \alpha \bullet \beta, a$$

- $X \rightarrow \alpha \beta$ is a production
 - a is a terminal (the **lookahead** terminal)
 - LR(1) means 1 lookahead terminal
- $[X \rightarrow \alpha \bullet \beta, a]$ describes a **context** of the parser
 - We are trying to find an X followed by an a , and
 - We have α already on top of the stack
 - Thus we need to see next a prefix derived from βa

Note

- The symbol \blacktriangleright was used before to separate the stack from the rest of input
 - $\alpha \blacktriangleright \gamma$, where α is the stack and γ is the remaining string of terminals
- In LR(1) items \bullet is used to mark a prefix of a production rhs:

$$X \rightarrow \alpha \bullet \beta, a$$

- Here β might contain non-terminals as well
- In both case **the stack is on the left**

Convention

- We add to our grammar a fresh new start symbol S and a production $S \rightarrow E$
 - Where E is the old start symbol
 - No need to do this if E had only one production
- The **initial parsing context** contains:
 - $S \rightarrow \bullet E, \$$
 - Trying to find an S as a string derived from $E\$$
 - The stack is empty

LR(1) Items (Cont.)

- In context containing

$$E \rightarrow E + \bullet (E), +$$

- If (follows then we can perform a shift to context containing

$$E \rightarrow E + (\bullet E), +$$

- In context containing

$$E \rightarrow E + (E) \bullet , +$$

- We can perform a reduction with $E \rightarrow E + (E)$
- But **only if** a + follows

LR(1) Items (Cont.)

- Consider a context with the item

$$E \rightarrow E + (\bullet E) , +$$

- We expect next a string derived from $E) +$
- There are two productions for E

$$E \rightarrow \text{int} \quad \text{and} \quad E \rightarrow E + (E)$$

- We describe this by **extending** the context with two more items:

$$E \rightarrow \bullet \text{int} ,)$$

$$E \rightarrow \bullet E + (E) ,)$$

The Closure Operation

- The operation of extending the context with items is called the **closure** operation

Closure(Items) =

repeat

for each $[X \rightarrow \alpha \bullet Y\beta, a]$ **in** Items

for each production $Y \rightarrow \gamma$

for each $b \in \text{First}(\beta a)$

add $[Y \rightarrow \bullet \gamma, b]$ **to** Items

until Items is unchanged

Constructing the Parsing DFA (1)

- Construct the start context:

$$\text{Closure}(\{S \rightarrow \bullet E, \$\}) =$$

$S \rightarrow \bullet E, \$$
$E \rightarrow \bullet E+(E), \$$
$E \rightarrow \bullet \text{int}, \$$
$E \rightarrow \bullet E+(E), +$
$E \rightarrow \bullet \text{int}, +$

- We abbreviate as:

$$\begin{aligned} S &\rightarrow \bullet E, \$ \\ E &\rightarrow \bullet E+(E), \$/+ \\ E &\rightarrow \bullet \text{int}, \$/+ \end{aligned}$$



PLANNING

You... have a plan, right?

Constructing the Parsing DFA (2)

- An **LR(1) DFA state** is a **closed** set of **LR(1) items**
 - This means that we performed **Closure**
- The start state contains **$[S \rightarrow \bullet E, \$]$**
- A state that contains **$[X \rightarrow \alpha \bullet, b]$** is labeled with “**reduce with $X \rightarrow \alpha$ on b** ”
- And now the transitions ...

The DFA Transitions

- A state “**State**” that contains $[X \rightarrow \alpha \bullet y \beta, b]$ has a transition labeled y to a state that contains the items “Transition(**State**, y)”
 - y can be a terminal or a non-terminal

Transition(State, y) =

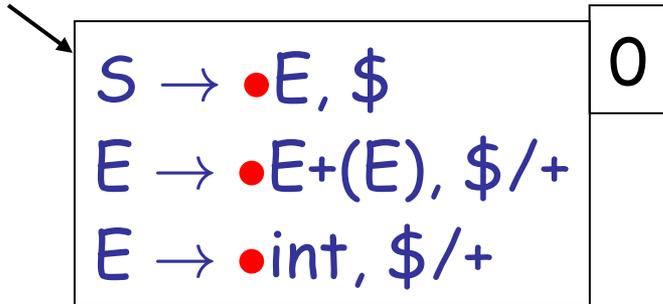
Items $\leftarrow \emptyset$

for each $[X \rightarrow \alpha \bullet y \beta, b] \in \text{State}$

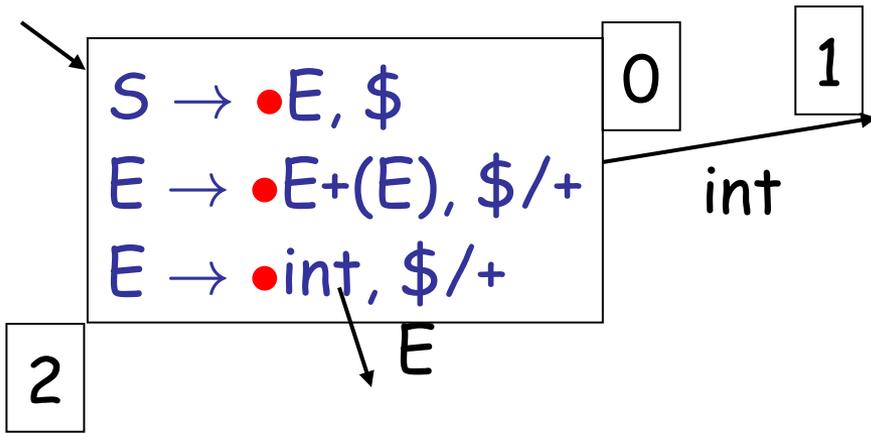
add $[X \rightarrow \alpha y \bullet \beta, b]$ to Items

return Closure(Items)

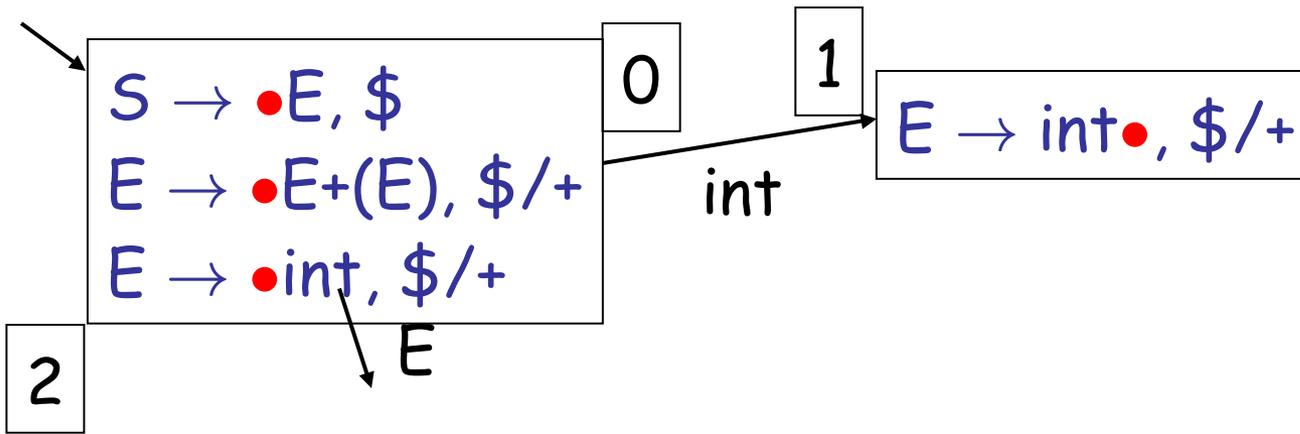
LR(1) DFA Construction Example



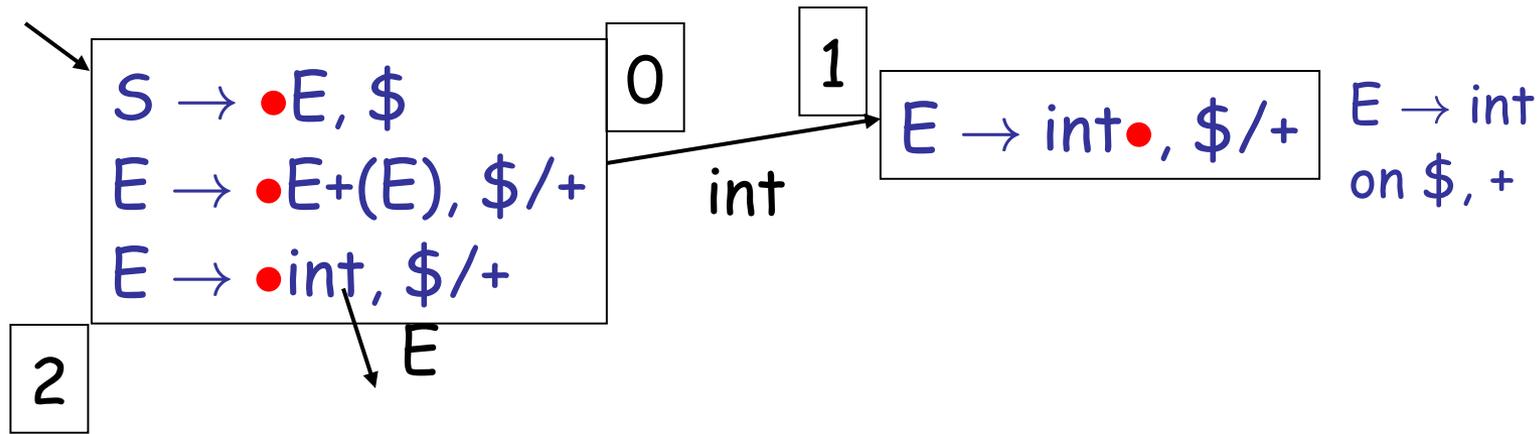
LR(1) DFA Construction Example



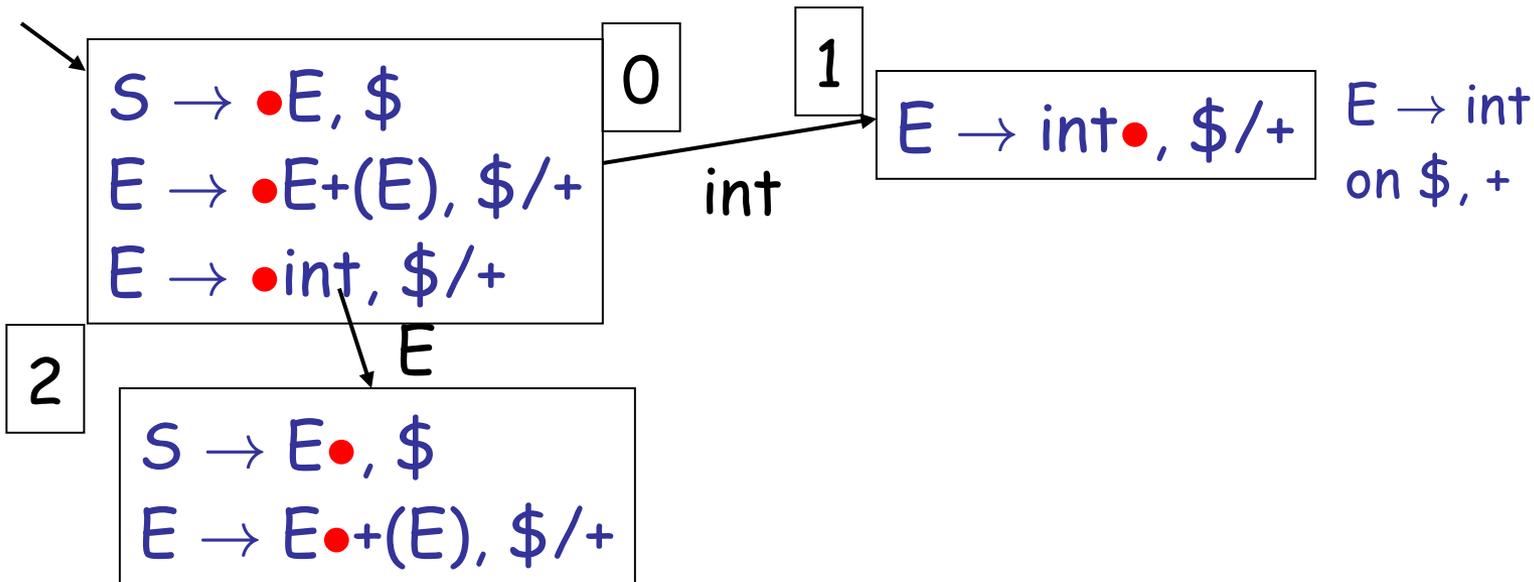
LR(1) DFA Construction Example



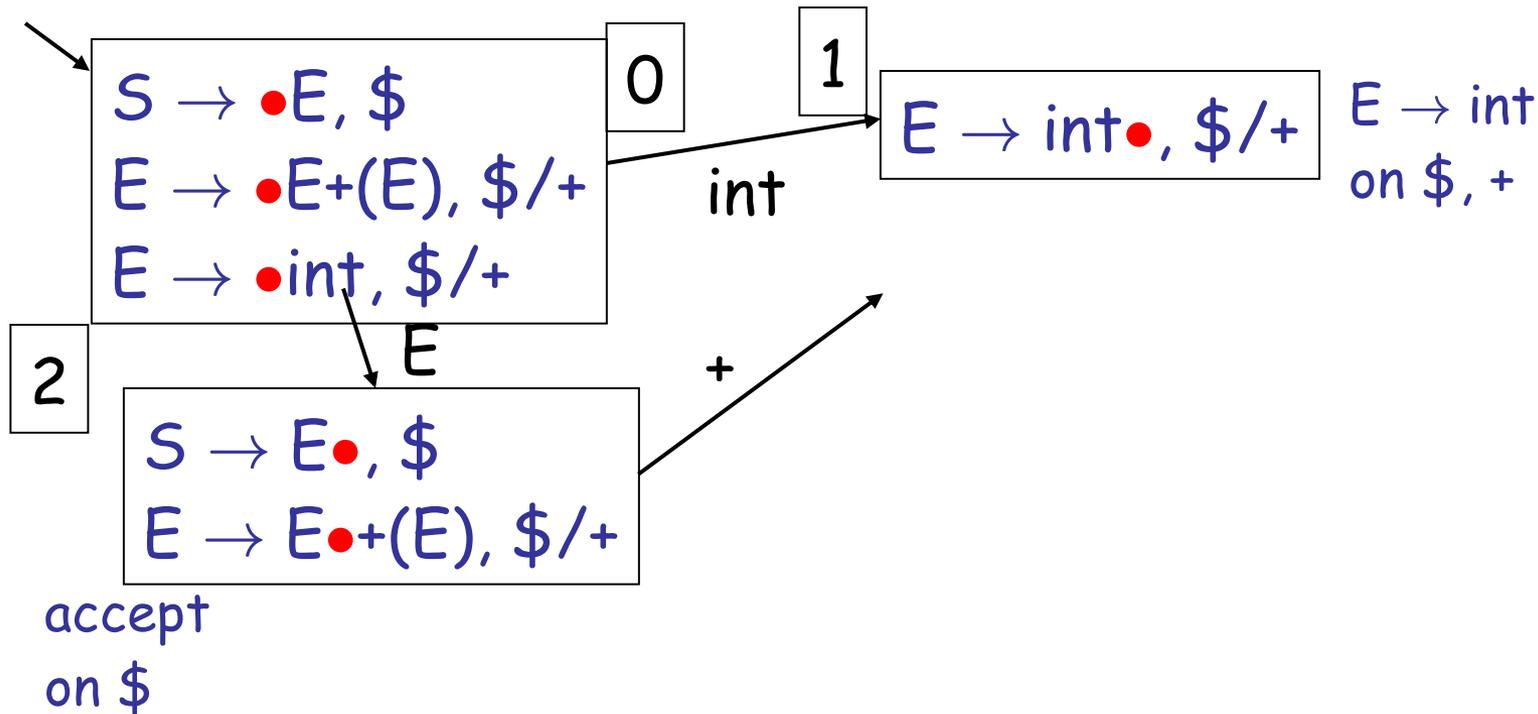
LR(1) DFA Construction Example



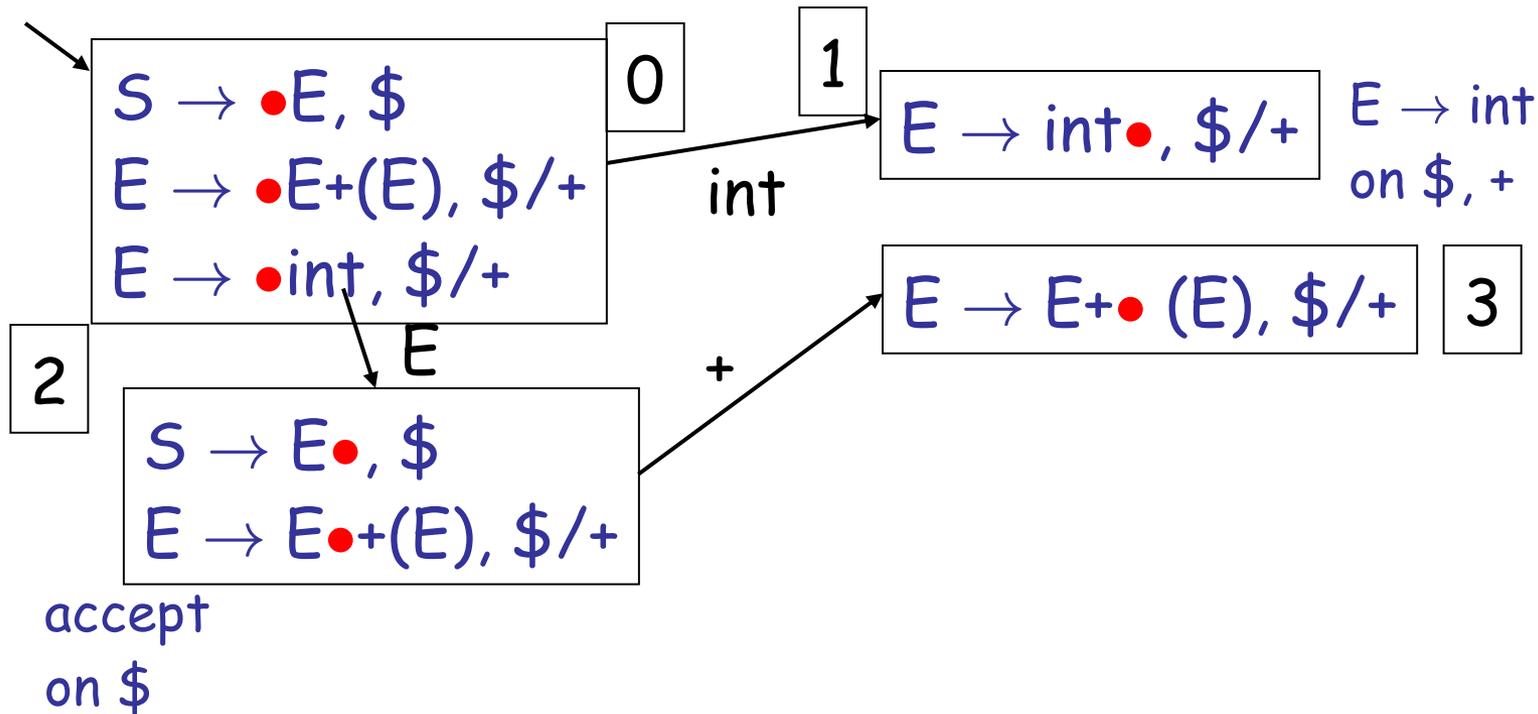
LR(1) DFA Construction Example



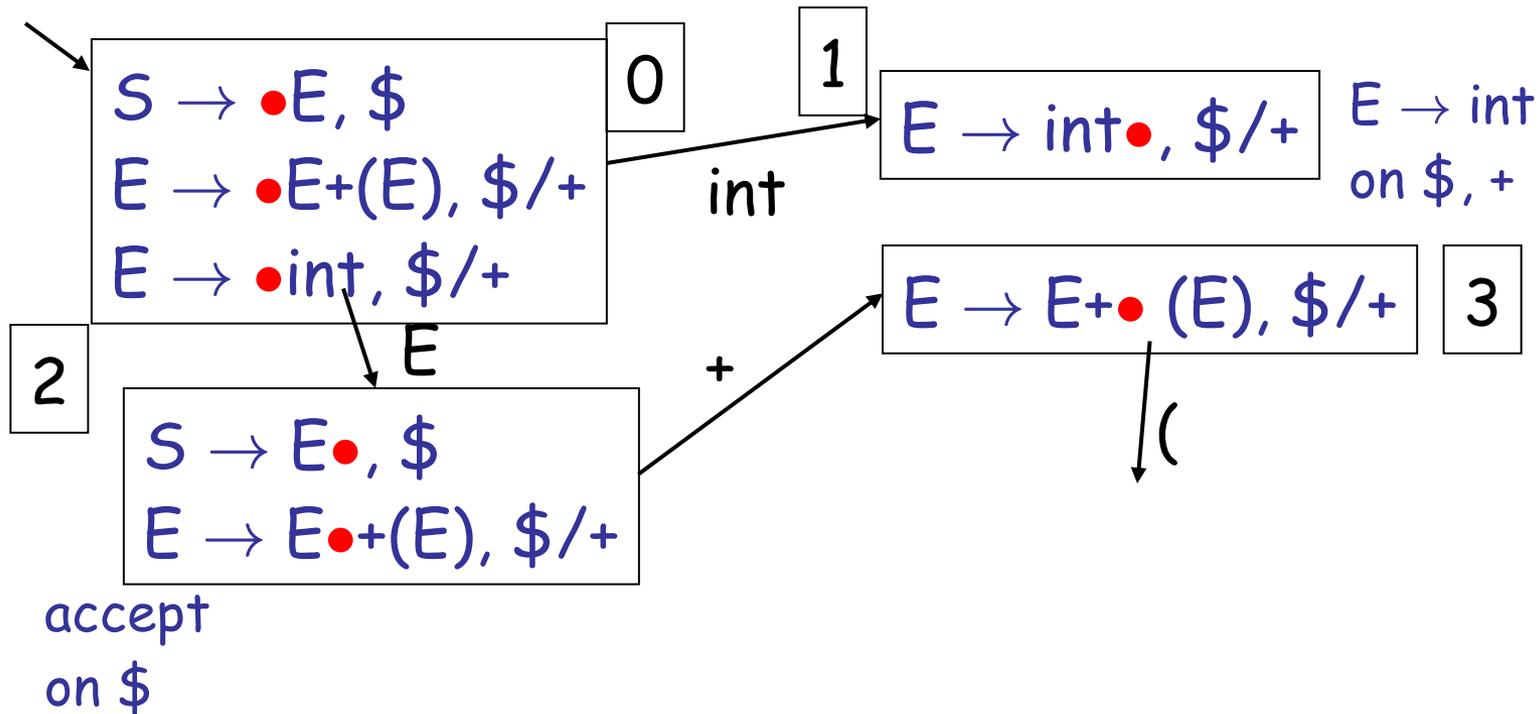
LR(1) DFA Construction Example



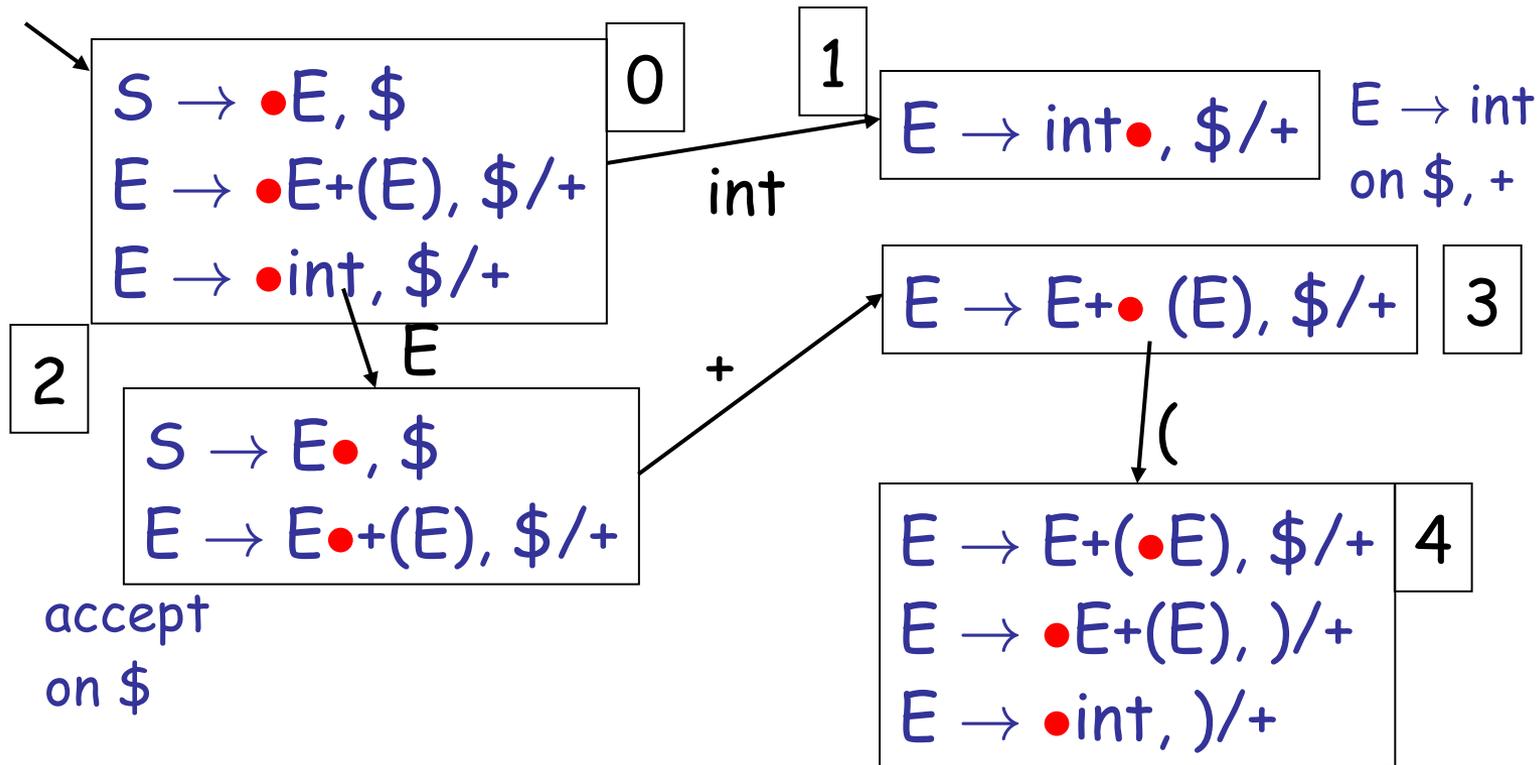
LR(1) DFA Construction Example



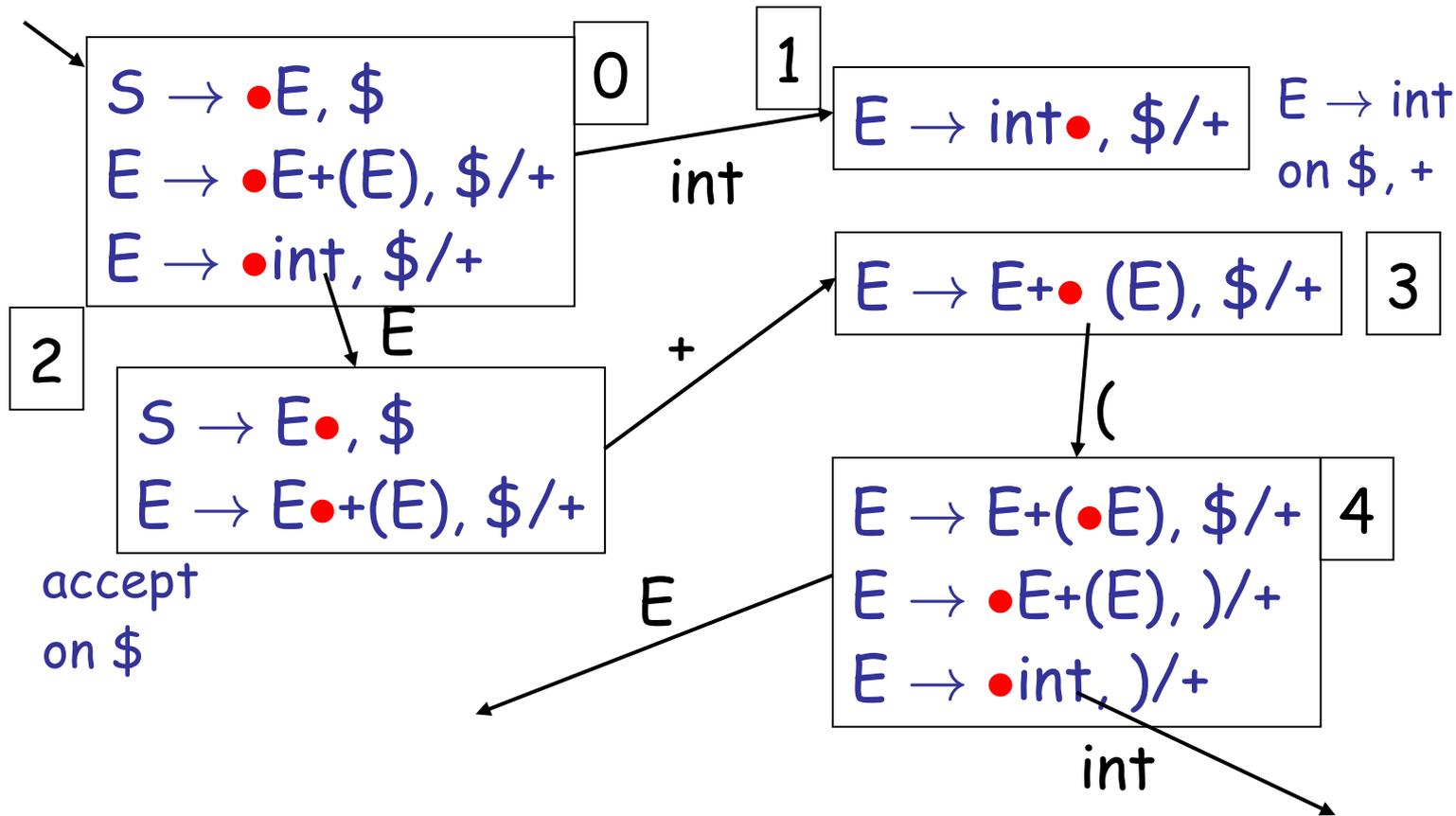
LR(1) DFA Construction Example



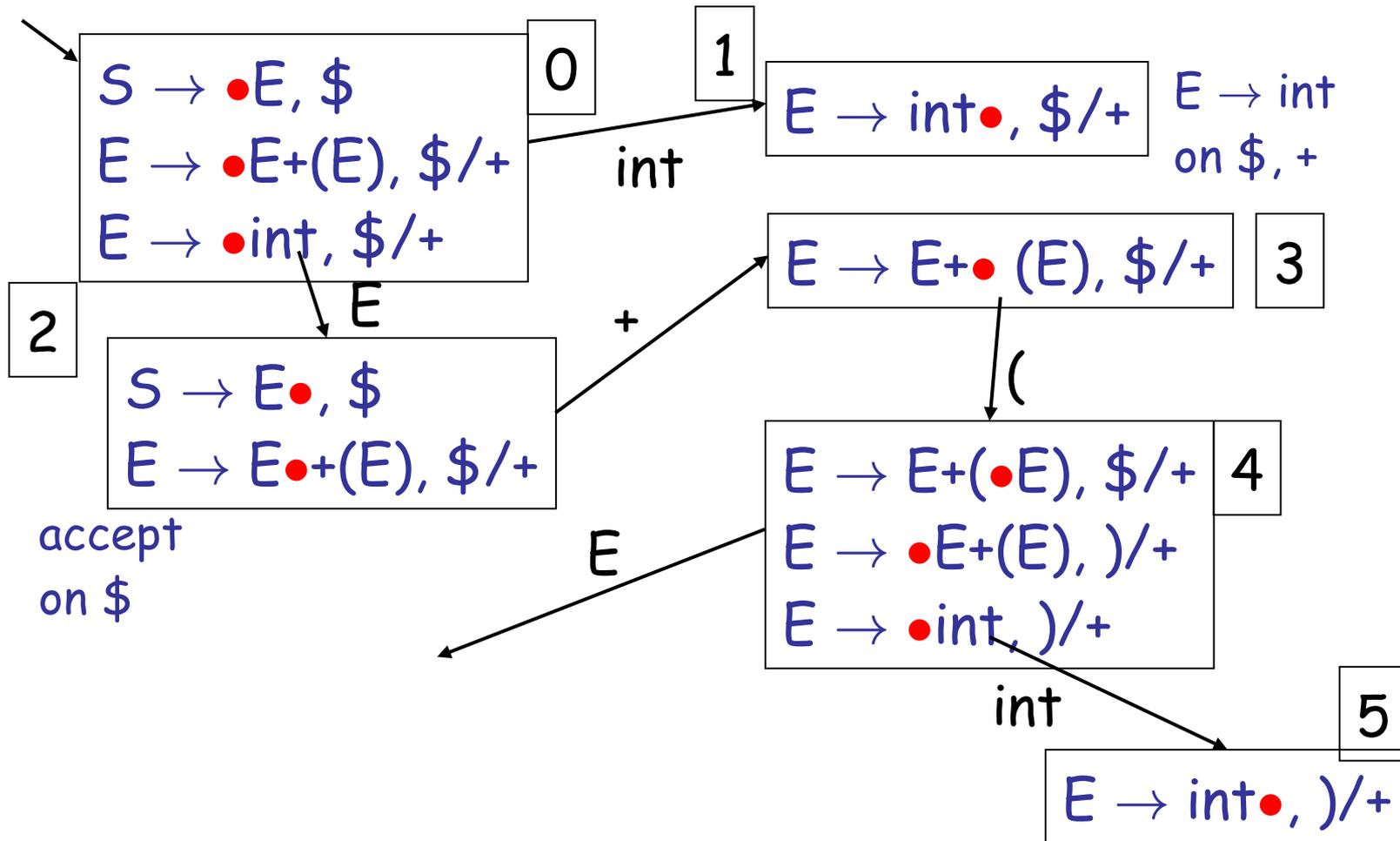
LR(1) DFA Construction Example



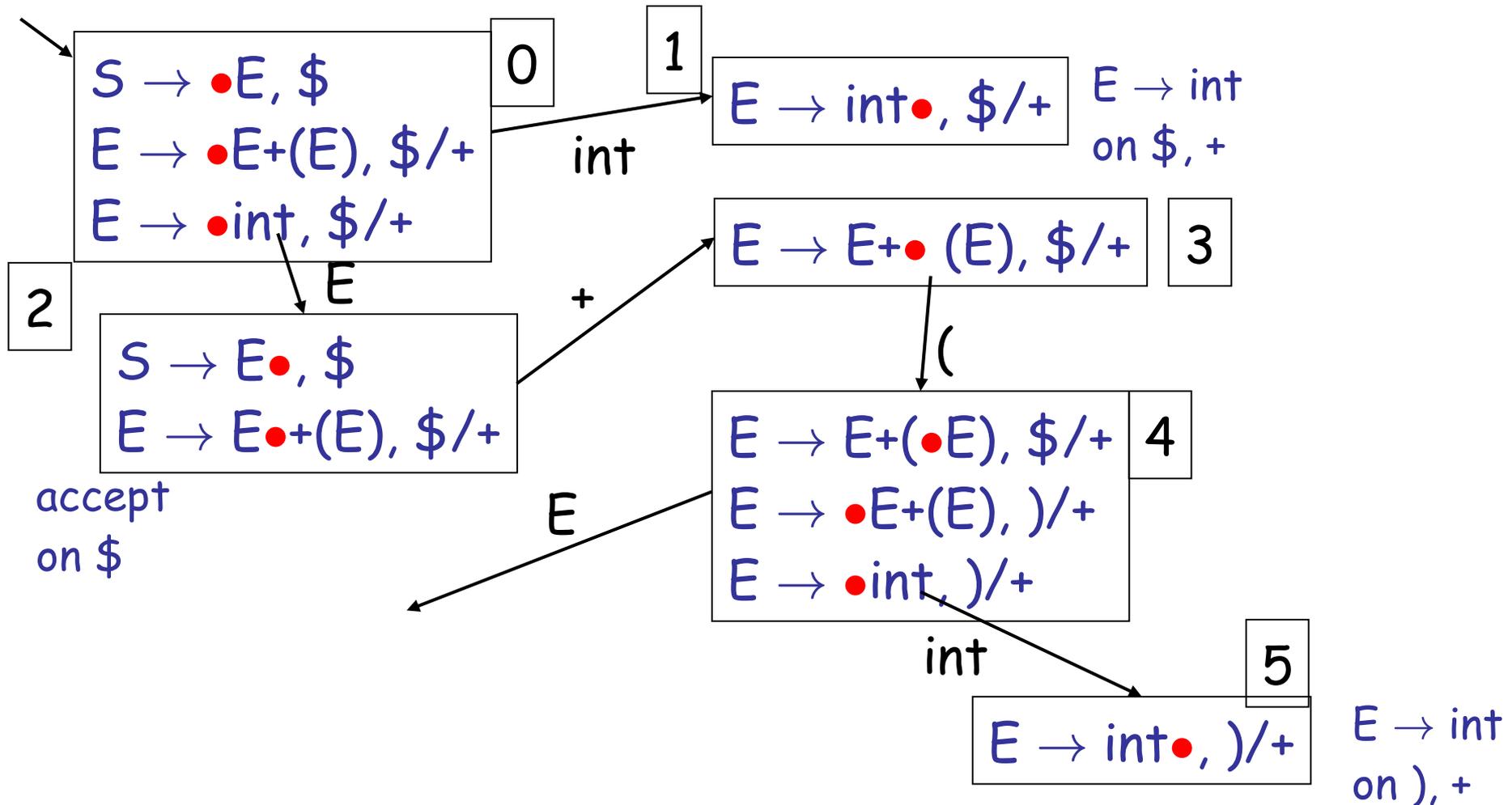
LR(1) DFA Construction Example



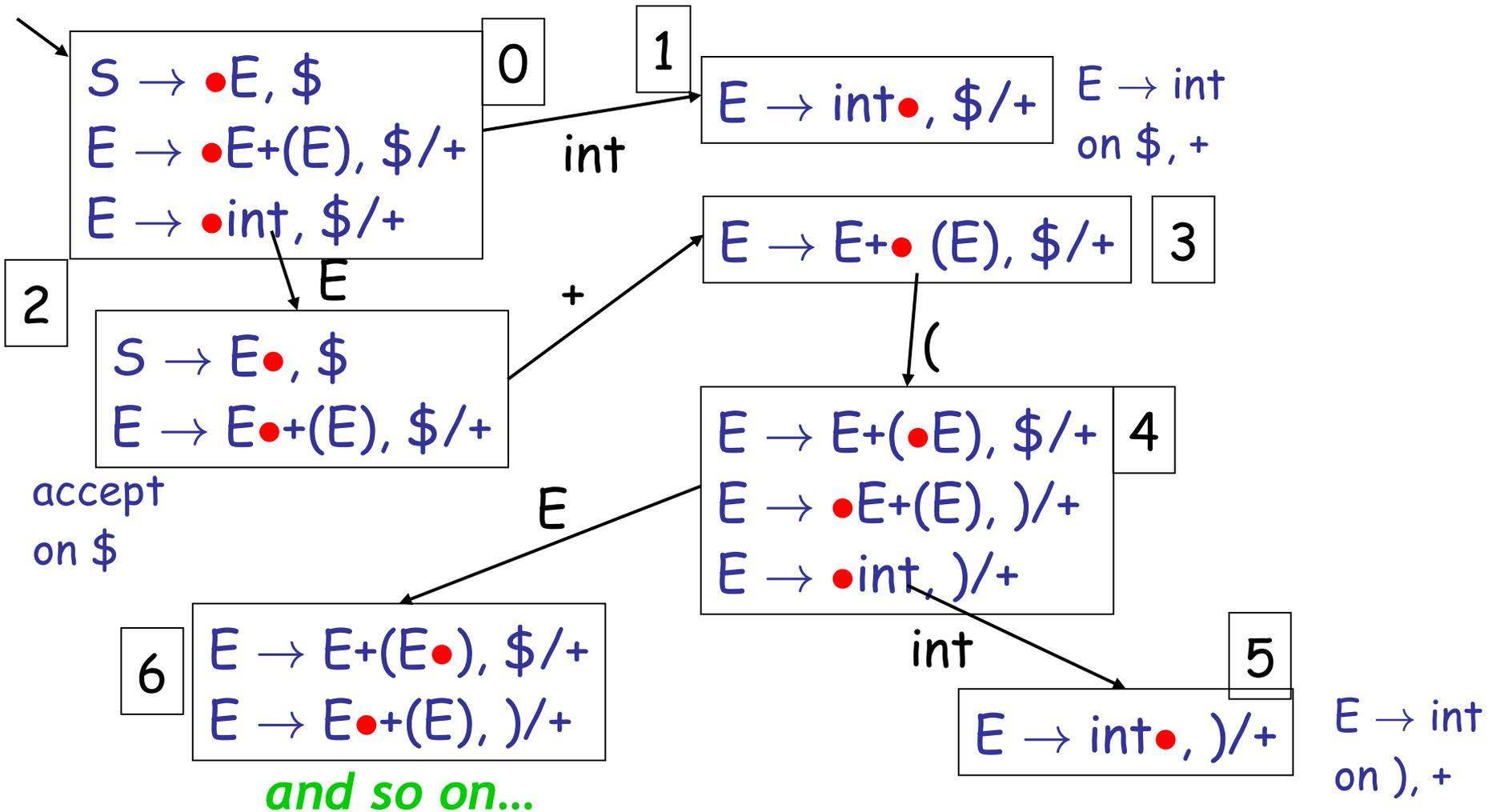
LR(1) DFA Construction Example



LR(1) DFA Construction Example



LR(1) DFA Construction Example



Natural Language, 20th Century

- Identify the United Kingdom poet associated with one of these verses:

He was my North, my South, my East and West,
My working week and my Sunday rest
My noon, my midnight, my talk, my song;
I thought that love would last forever, I was wrong.

Turning and turning in the widening gyre
The falcon cannot hear the falconer;
Things fall apart; the centre cannot hold;
Mere anarchy is loosed upon the world,
The blood-dimmed tide is loosed, and everywhere
The ceremony of innocence is drowned;
The best lack all conviction, while the worst
Are full of passionate intensity.

Q: Movie Music (420 / 842)

- In a 1995 Disney movie that has been uncharitably referred to as "Hokey-Hontas", the Stephen Schwartz lyrics "*what I love most about rivers is: / you can't step in the same river twice*" refer to the ideas of which Greek philosopher?

Leading Pop Culture Question

- Name either one of:
 - The *Star Trek: The Next Generation* two-part cliffhanger episode in which the Borg attack Earth and Picard becomes Locutus.
 - The bubblegum rock *Hannah Montana* main theme, performed by Miley Cyrus, that showcases the protagonist's double life.

Philosophy

- In his 1710 French work *Essais de Théodicée sur la bonté de Dieu, la liberté de l'homme et l'origine du mal*, Gottfried Leibniz (yes, the Calculus one) coined this term to describe his solution to the problem of evil in theodicy:
 - If God is omnibenevolent, omnipotent and omniscient, how do we account for the suffering and injustice that exist?

Natural Language, Poetry

- This metrical line is commonly used in verse. The rhythm involves a fixed number of double-syllable “feet”, as in this example by Keats:

˘	/	˘	/	˘	/	˘	/	˘	/
To	swell	the	gourd,	and	plump	the	ha-	zel	shells

LR Parsing Tables. Notes

- Parsing tables (= the DFA) can be constructed **automatically** for a CFG
 - “The tables which cannot be constructed are constructed automatically in response to a CFG input. You asked for a miracle, Theo. I give you the L-R-1.” - Hans Gruber, Die Hard
- But we still need to understand the construction to work with parser generators
 - e.g., they **report errors in terms of sets of items**
- What kind of errors can we expect?



PARTY CONFLICT

Sometimes, you should back down.

Shift/Reduce Conflicts

- If a DFA state contains both
 $[X \rightarrow \alpha \bullet a \beta, b]$ and $[Y \rightarrow \gamma \bullet, a]$
- Then on input “a” we could either
 - Shift into state $[X \rightarrow \alpha a \bullet \beta, b]$, or
 - Reduce with $Y \rightarrow \gamma$
- This is called a **shift-reduce conflict**

Shift/Reduce Conflicts

- Typically due to **ambiguities in the grammar**
- Classic example: the dangling else
 $S \rightarrow \text{if } E \text{ then } S \mid \text{if } E \text{ then } S \text{ else } S \mid \text{OTHER}$
- Will have DFA state containing
 $[S \rightarrow \text{if } E \text{ then } S \bullet, \quad \text{else}]$
 $[S \rightarrow \text{if } E \text{ then } S \bullet \text{ else } S, \quad x]$
- If **else** follows then we can shift or reduce
- Default (bison, CUP, etc.) is to shift
 - Default behavior is as needed in this case

More Shift/Reduce Conflicts

- Consider the ambiguous grammar

$$E \rightarrow E + E \mid E * E \mid \text{int}$$

- We will have the states containing

$$\begin{array}{cc} [E \rightarrow E * \bullet E, +] & [E \rightarrow E * E \bullet, +] \\ [E \rightarrow \bullet E + E, +] \Rightarrow^E [E \rightarrow E \bullet + E, +] & \\ \dots & \dots \end{array}$$

- Again we have a shift/reduce on input +
 - We need to reduce (* binds more tightly than +)
 - Solution: declare the **precedence** of * and +

More Shift/Reduce Conflicts

- In bison declare **precedence** and **associativity**:

```
%left +  
%left * // high precedence
```

- **Precedence** of a rule = that of its last terminal
 - See bison manual for ways to override this default
- Resolve shift/reduce conflict with a **shift** if:
 - no precedence declared for either rule or terminal
 - input terminal has higher precedence than the rule
 - the precedences are the same and right associative

Using Precedence to Solve S/R Conflicts

- Back to our example:

$$\begin{array}{l} [E \rightarrow E * \bullet E, +] \quad [E \rightarrow E * E \bullet, +] \\ [E \rightarrow \bullet E + E, +] \Rightarrow^E [E \rightarrow E \bullet + E, +] \\ \dots \qquad \qquad \qquad \dots \end{array}$$

- Will choose **reduce** on input + because precedence of rule $E \rightarrow E * E$ is higher than of terminal +

Using Precedence to Solve S/R Conflicts

- Same grammar as before

$$E \rightarrow E + E \mid E * E \mid \text{int}$$

- We will also have the states

$$[E \rightarrow E + \bullet E, +]$$

$$[E \rightarrow E + E \bullet, +]$$

$$[E \rightarrow \bullet E + E, +] \Rightarrow^E$$

$$[E \rightarrow E \bullet + E, +]$$

...

...

- Now we also have a shift/reduce on input +
 - We choose **reduce** because $E \rightarrow E + E$ and + have the same precedence and + is left-associative

Using Precedence to Solve S/R Conflicts

- Back to our dangling else example

[S → if E then S•, else]

[S → if E then S• else S, x]

- Can eliminate conflict by declaring **else** with **higher precedence** than **then**
 - Or just rely on the default shift action
- But this starts to look like “hacking the parser”
- Avoid overuse of precedence declarations or you’ll end with unexpected parse trees
 - The kiss of death ...

Reduce/Reduce Conflicts

- If a DFA state contains both
 $[X \rightarrow \alpha \bullet, a]$ and $[Y \rightarrow \beta \bullet, a]$
 - Then on input “a” we don’t know which production to reduce
- This is called a **reduce/reduce conflict**

Reduce/Reduce Conflicts

- Usually due to **gross ambiguity** in the grammar

- Example: a sequence of identifiers

$$S \rightarrow \varepsilon \mid id \mid id S$$

- There are **two parse trees** for the string **id**

$$S \rightarrow id$$

$$S \rightarrow id S \rightarrow id$$

- How does this confuse the parser?

More on Reduce/Reduce Conflicts

- Consider the states

$[S' \rightarrow \bullet S, \$]$		$[S \rightarrow id \bullet, \$]$
$[S \rightarrow \bullet, \$]$	\Rightarrow_{id}	$[S \rightarrow \bullet id, \$]$
$[S \rightarrow \bullet id, \$]$		$[S \rightarrow \bullet id S, \$]$
$[S \rightarrow \bullet id S, \$]$		

- Reduce/reduce conflict on input $\$$

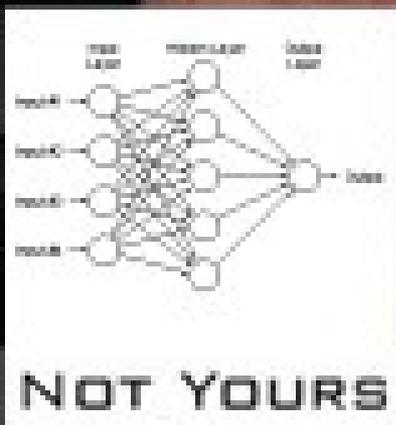
$S' \rightarrow S \rightarrow id$

$S' \rightarrow S \rightarrow id S \rightarrow id$

- Better **rewrite the grammar**: $S \rightarrow \epsilon \mid id S$

Can's someone learn this for me?

No, you can't have a neural network



Using Parser Generators

- **Parser generators** construct the parsing DFA given a CFG
 - Use precedence declarations and default conventions to **resolve conflicts**
 - The **parser algorithm is the same** for all grammars (and is provided as a library function)
- But most parser generators do not construct the DFA as described before
 - *Why might that be?*

Using Parser Generators

- **Parser generators** construct the parsing DFA given a CFG
 - Use precedence declarations and default conventions to **resolve conflicts**
 - The **parser algorithm is the same** for all grammars (and is provided as a library function)
- But most parser generators do not construct the DFA as described before
 - Because the LR(1) parsing DFA has 1000s of states even for a simple language

LR(1) Parsing Tables are Big

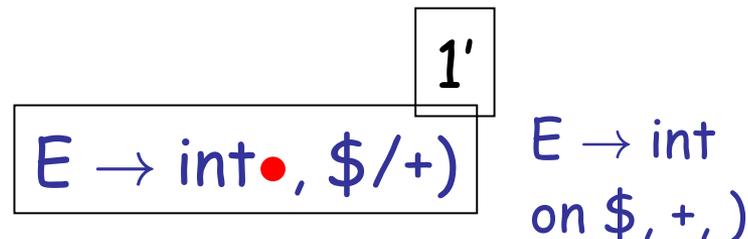
- But many states are similar, e.g.



- Idea: **merge** the DFA states whose items **differ only in the lookahead tokens**

- We say that such states have the same **core**

- We obtain



The Core of a Set of LR Items

- Definition: The **core** of a set of LR items is the set of first components
 - **Without** the lookahead terminals

- Example: the core of

$$\{ [X \rightarrow \alpha \bullet \beta, b], [Y \rightarrow \gamma \bullet \delta, d] \}$$

is

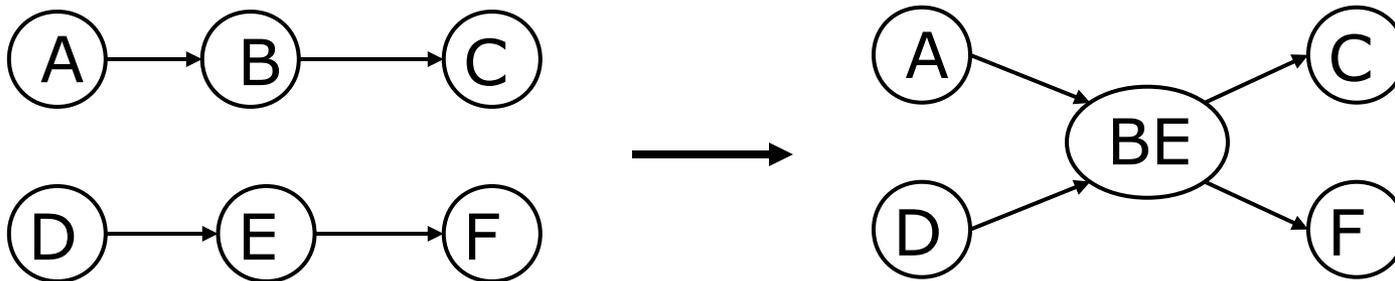
$$\{ X \rightarrow \alpha \bullet \beta, Y \rightarrow \gamma \bullet \delta \}$$

LALR States

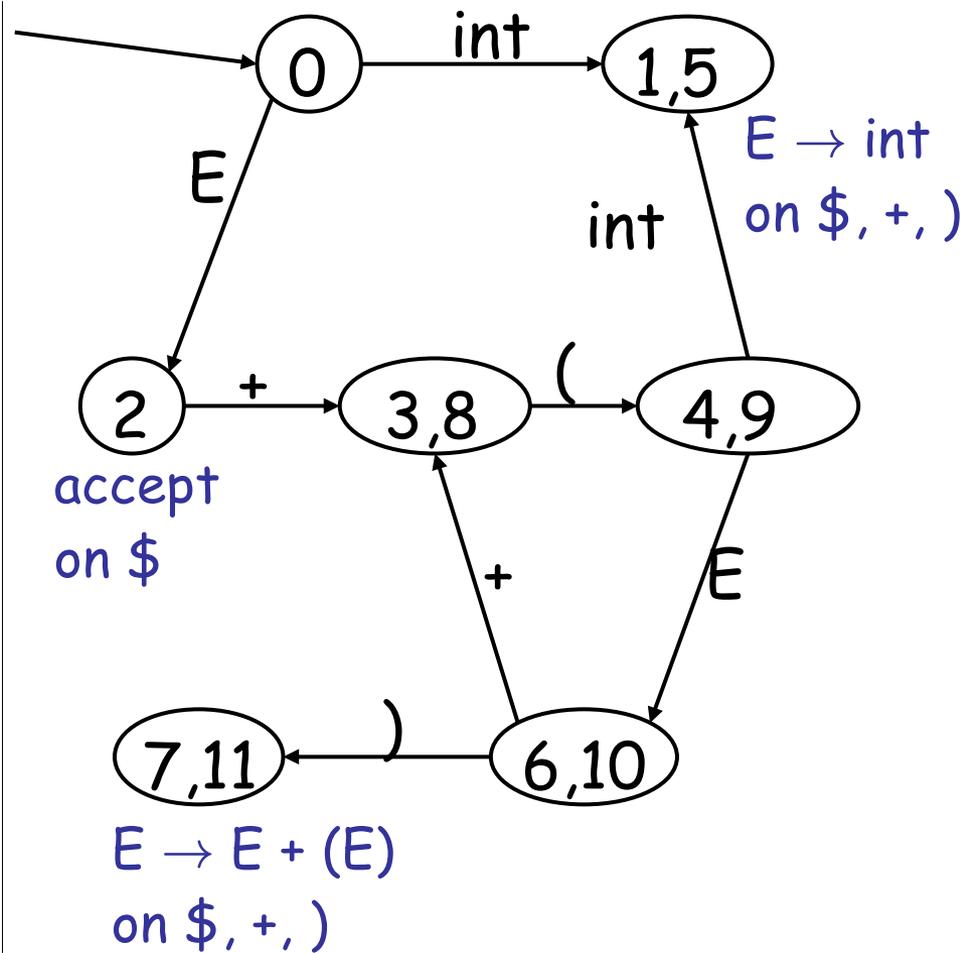
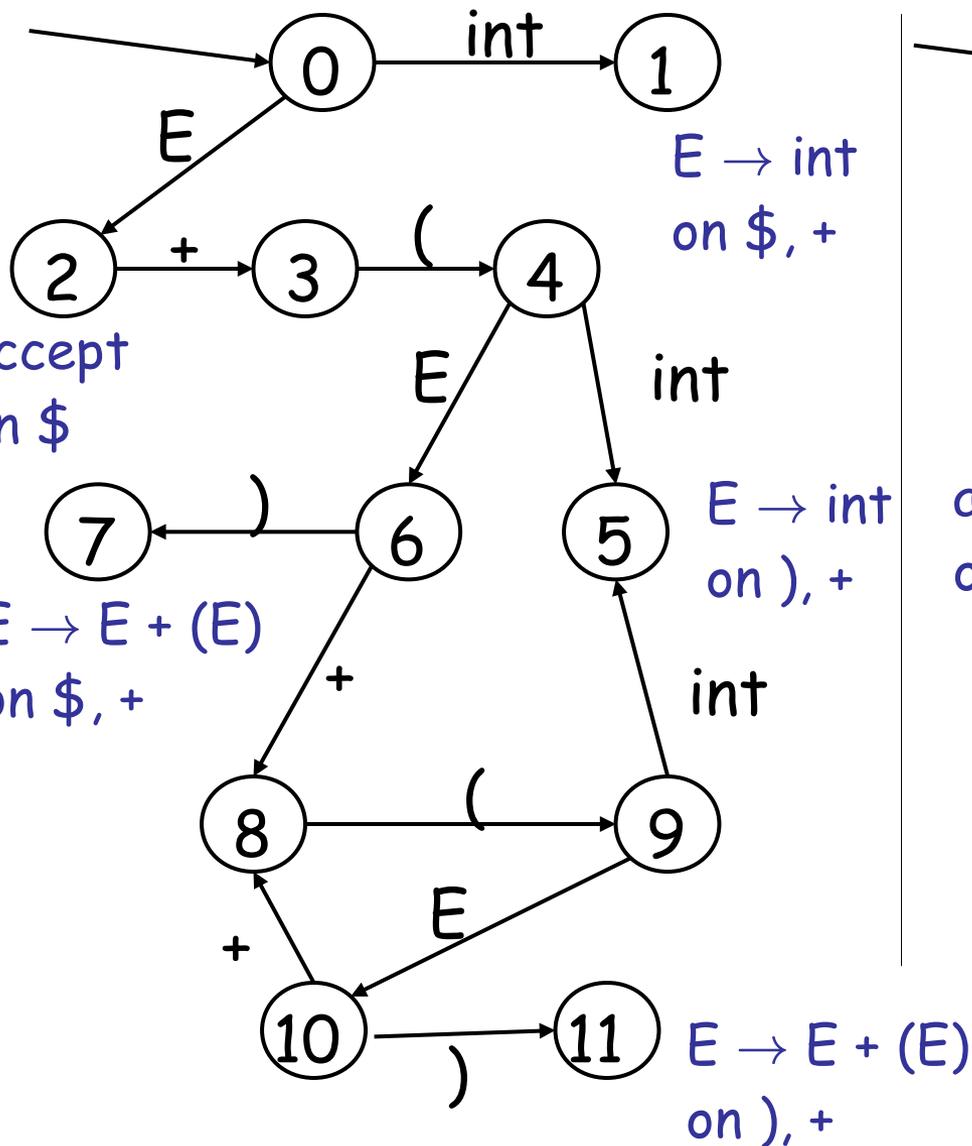
- Consider for example the LR(1) states
$$\{[X \rightarrow \alpha \bullet, a], [Y \rightarrow \beta \bullet, c]\}$$
$$\{[X \rightarrow \alpha \bullet, b], [Y \rightarrow \beta \bullet, d]\}$$
- They have the **same core** and can be merged
- And the merged state contains:
$$\{[X \rightarrow \alpha \bullet, a/b], [Y \rightarrow \beta \bullet, c/d]\}$$
- These are called **LALR(1)** states
 - Stands for **LookAhead LR**
 - Typically 10x fewer LALR(1) states than LR(1)

LALR(1) DFA

- **Repeat** until all states have distinct core
 - Choose two distinct states with same core
 - Merge the states by creating a new one with the union of all the items
 - Point edges from predecessors to new state
 - New state points to all the previous successors



Example LALR(1) to LR(1)



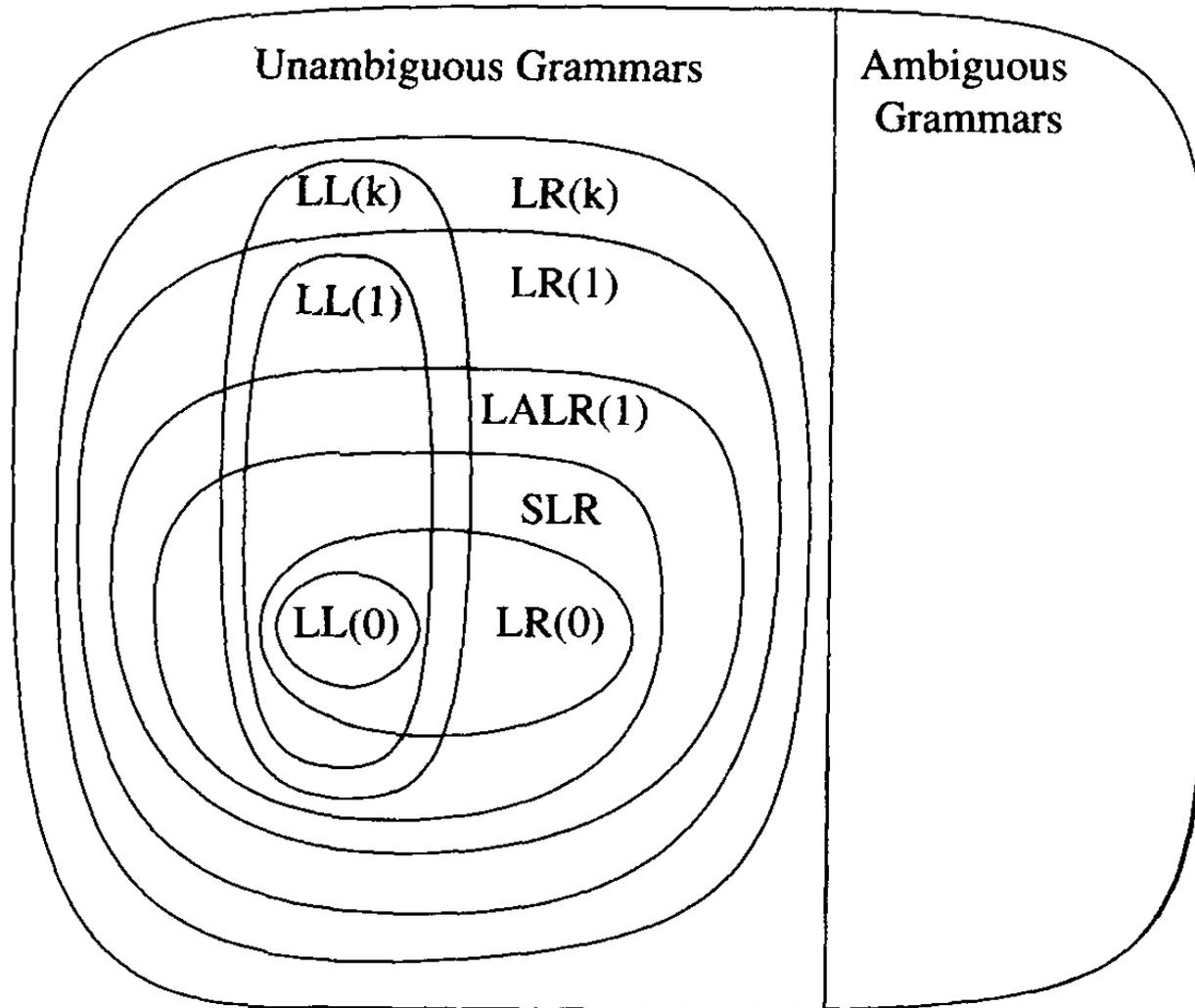
The LALR Transform Can Introduce Conflicts

- Consider for example the LR(1) states
$$\{[X \rightarrow \alpha \bullet, a], [Y \rightarrow \beta \bullet, b]\}$$
$$\{[X \rightarrow \alpha \bullet, b], [Y \rightarrow \beta \bullet, a]\}$$
- And the merged LALR(1) state
$$\{[X \rightarrow \alpha \bullet, a/b], [Y \rightarrow \beta \bullet, a/b]\}$$
- Has a new reduce-reduce conflict
- In practice such cases are rare

LALR vs. LR Parsing

- LALR languages are **not natural**
 - They are an **efficiency hack** on LR languages
- Any “**reasonable**” programming language has a LALR(1) grammar
- LALR(1) has become a **standard** for programming languages and for parser generators

A Hierarchy of Grammar Classes



From Andrew Appel,
“Modern Compiler
Implementation in Java”

Notes on Parsing

- Parsing
 - A solid foundation: context-free grammars
 - A simple parser: LL(1)
 - A more powerful parser: LR(1)
 - An efficiency hack: LALR(1)
 - LALR(1) parser generators
- Now we move on to semantic analysis

Take a bow, you survived!



Supplement to LR Parsing

Strange Reduce/Reduce Conflicts
Due to LALR Conversion
(from the bison manual)

Strange Reduce/Reduce Conflicts

- Consider the grammar

$S \rightarrow P R , \quad NL \rightarrow N \mid N , NL$

$P \rightarrow T \mid NL : T \quad R \rightarrow T \mid N : T$

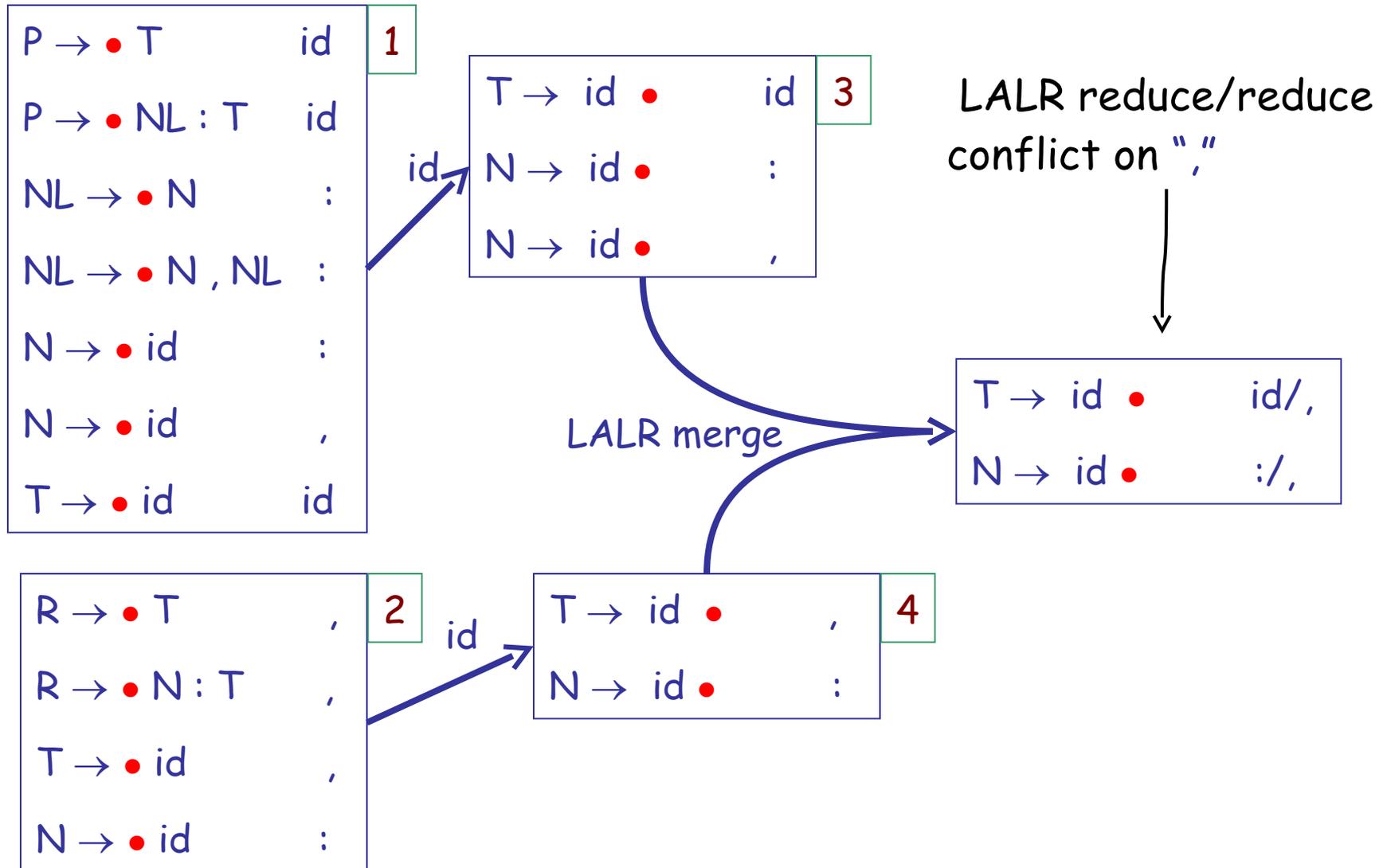
$N \rightarrow id \quad T \rightarrow id$

- **P** - parameters specification
- **R** - result specification
- **N** - a parameter or result name
- **T** - a type name
- **NL** - a list of names

Strange Reduce/Reduce Conflicts

- In P an id is a
 - N when followed by $,$ or $:$
 - T when followed by id
- In R an id is a
 - N when followed by $:$
 - T when followed by $,$
- This is an LR(1) grammar.
- But it is not LALR(1). Why?
 - For obscure reasons

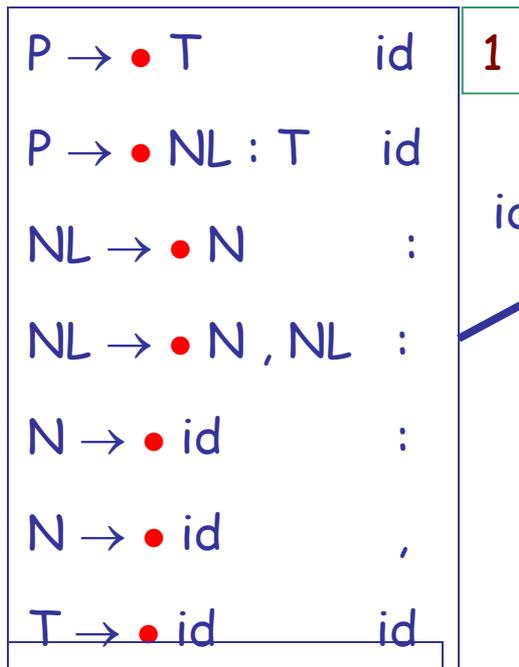
A Few LR(1) States



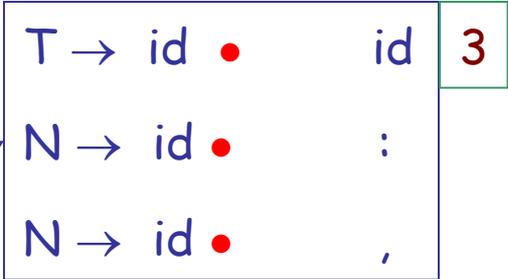
What Happened?

- Two distinct states were confused because they have the same core
- Fix: add dummy productions to distinguish the two confused states
- E.g., add
 - $R \rightarrow \text{id bogus}$
 - **bogus** is a terminal not used by the lexer
 - This production will never be used during parsing
 - But it distinguishes **R** from **P**

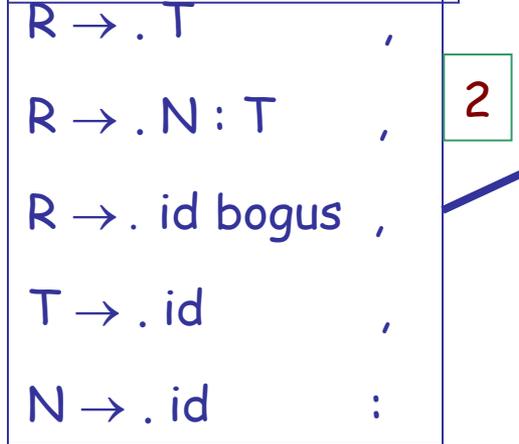
A Few LR(1) States After Fix



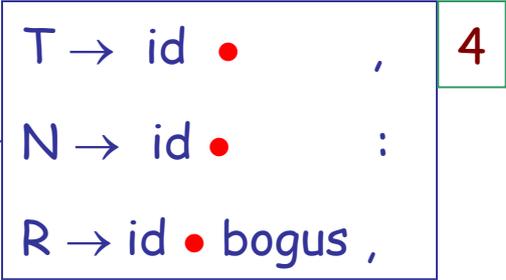
id



Different cores \Rightarrow no LALR merging



id



Homework

- WA2 Due
- **Midterm 1 in Class**