中山大學 SUN YAT-SEN UNIVERSITY

国家超级计算广州中心 NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

Compilation Principle
编 译 原 理

第11讲：语法分析(8)

张献伟

xianweiz.github.io

DCS290, 3/29/2022

中山大學 SUN YAT-SEN UNIVERSITY

Review Questions

- Action table entries can be si and rj, what are i and j?
  - si: shift the input symbol and move to state i
  - rj: reduce by production numbered j
- Item/Configuration: what does A → XYZ• mean?
  - We have seen the body XYZ and it is time to reduce XYZ to A
- State: why we put the items into a configuration set?
  - Closure: we hope to see one symbol in First(Y) $Y \rightarrow u|w$ $A \rightarrow X \bullet YZ$, $Y \rightarrow \bullet u$, $Y \rightarrow \bullet w$
- What is augmented grammar?
  - Add one extra rule S' → S to guarantee only one 'acc' in the table
- What are the possible items of S' → S?
  - S' → •S: initial item, haven't seen any input symbol
  - S' → S•: accept item, have reduced the input string to start symbol

The Example

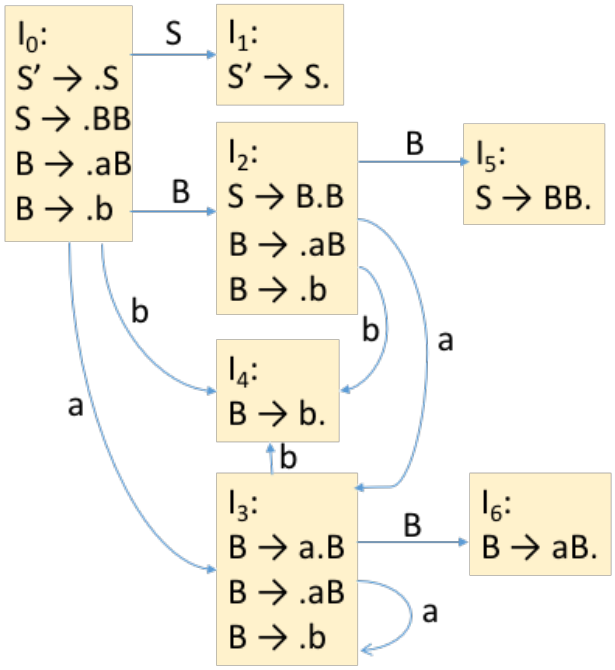
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$ => Bab#$ => BaB#$ => BB#$ => S#$

The Example

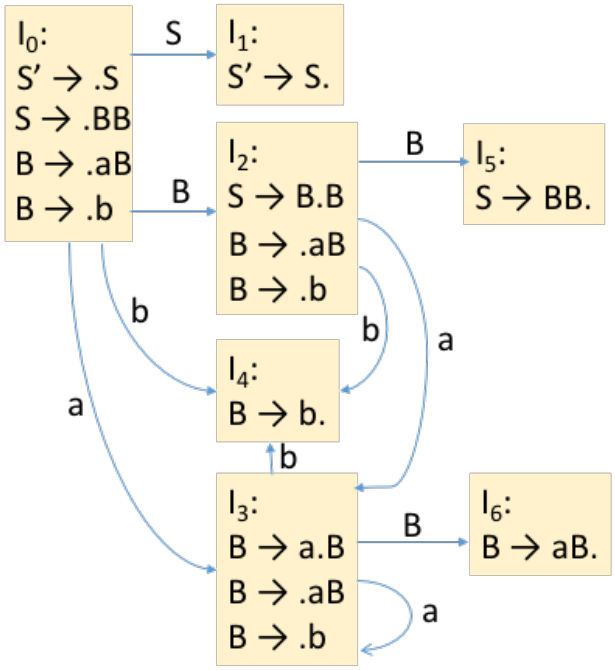
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



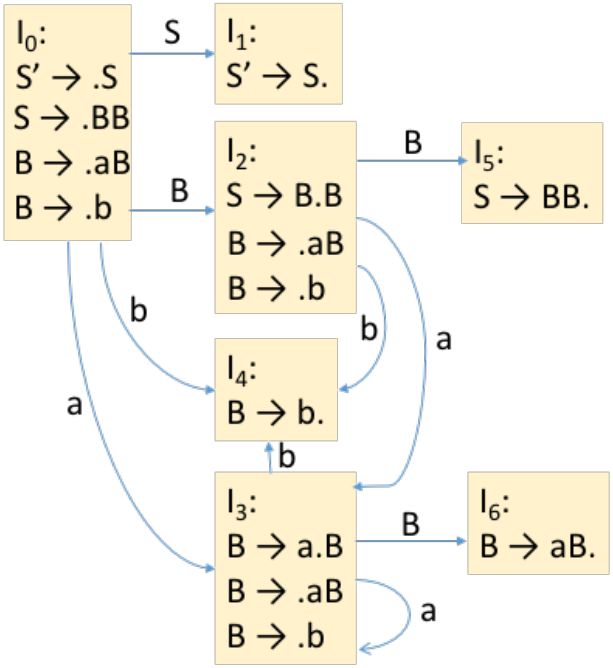
State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

☆ 是LR(0)，没有任何lookahead ☆

- state直接决定了是shift/reduce，并不需要看输入符号
- 若reduce，输入符号及整个input buffer没有任何变化
- 若shift，输入符号从input buffer移入stack

LR(0) Parsing

- Construct LR(0) automaton from the Grammar[由文法构建自动机]
- Idea: assume
  - Input buffer contains α[但buffer不止有α]
  - Next input is *t*[α后是t]
  - DFA on input α terminates in state s
    - α处理完毕后处于状态s
- Reduce by X → β if[归约]
  - s contains item X → β·
- Shift if[移进]
  - s contains item X → β·*t*ω
  - Equivalent to saying s has a transition labeled *t*



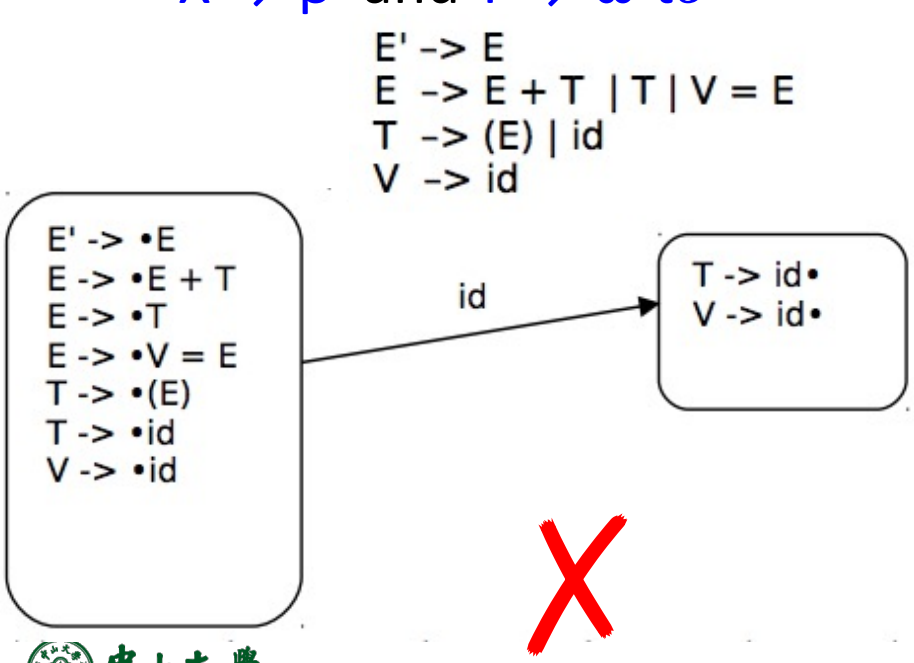
LR(0) Parsing (cont.)

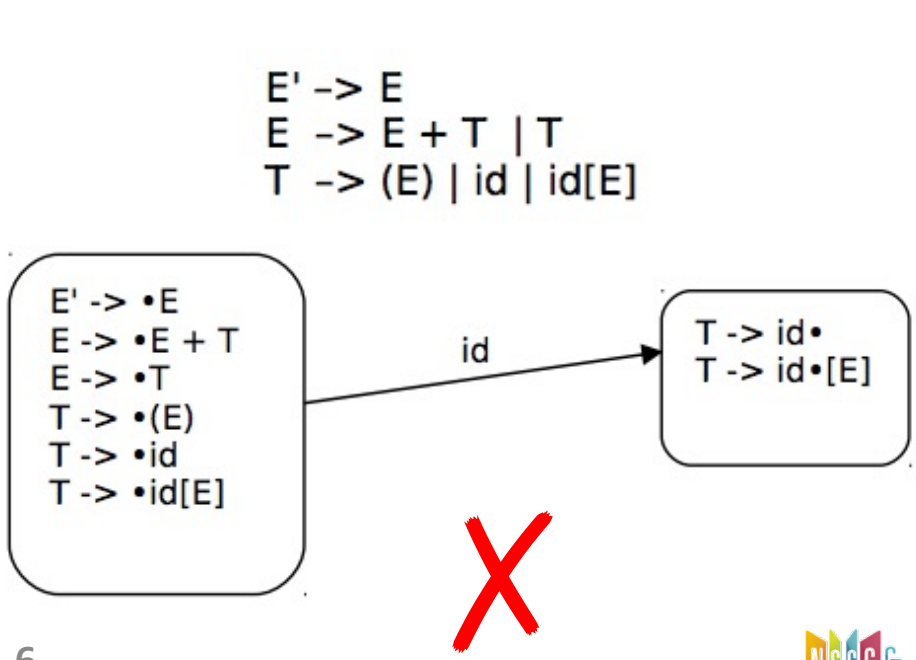
- The parser must be able to determine what action to take in each state without looking at any further input symbols [没有展望即决定动作]
  - i.e. by only considering what the parsing stack contains so far
  - This is the '**0**' in the parser name
- In a LR(0) table, each state must only shift or reduce[确定性移进或归约]
  - Thus an LR(0) configurating set can only have <u>exactly one</u> reduce item
    - cannot have both shift and reduce items

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

LR(0) Conflicts[冲突]

- LR(0) has a **reduce/reduce** conflict[归约-归约冲突] if:
  - Any state has two reduce items:
  - $X \rightarrow \beta\cdot$ and $Y \rightarrow \omega\cdot$
- LR(0) has a **shift/reduce** conflict[移进-归约冲突] if:
  - Any state has a reduce item and a shift item:
  - $X \rightarrow \beta\cdot$ and $Y \rightarrow \omega\cdot t\sigma$

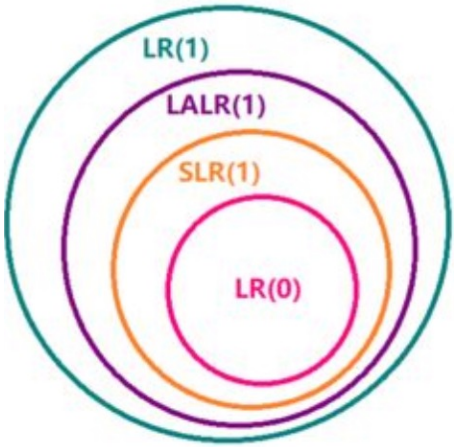




LR(0) Summary[小结]

- LR(0) is the simplest LR parsing[最简单]
  - Table-driven shift-reduce parser[表驱动]
    - Action table[s, a] + Goto table[s, X]
  - Weakest, not used much in practice[实际很少使用]
  - Parses without using any lookahead[没有任何展望]

- Adding just one token of lookahead vastly increases the parsing power[考虑展望]
  - SLR(1): simple LR(1), use FOLLOW[归约用FOLLOW]
  - LR(1): use dedicated symbols[比FOLLOW更精细]
  - LALR(1): balance SLR(1) and LR(1)[折衷]

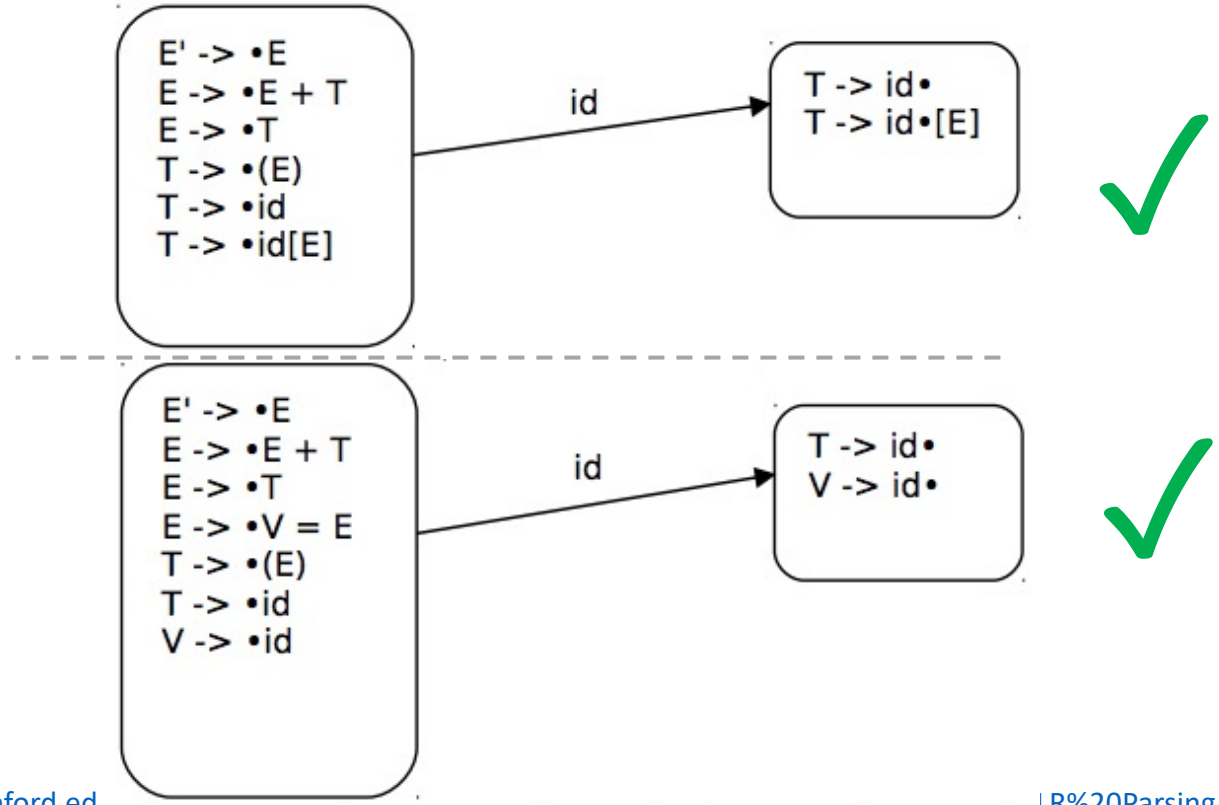


SLR(1) Parsing

- LR(0) conflicts are generally caused by **reduce** actions
  - If the item is complete (A → α.), the parser must choose to reduce[项目形式完整就归约]
    - Is this always appropriate?
    - The next upcoming token may tell us something different
  - What tokens may tell the reduction is not appropriate?
    - Perhaps **Follow(A)** could be useful here
- **SLR** = Simple LR
  - Use the same LR(0) configurating sets and have the same table structure and parser operation[表结构一致]
  - The difference comes in assigning table actions[动作填充不同]
    - Use one token of lookahead to help arbitrate among the conflicts
    - Reduce only if the next input token is a member of the FOLLOW set of the nonterminal being reduced to[下一token在FOLLOW集才归约]

SLR(1) Parsing (cont.)

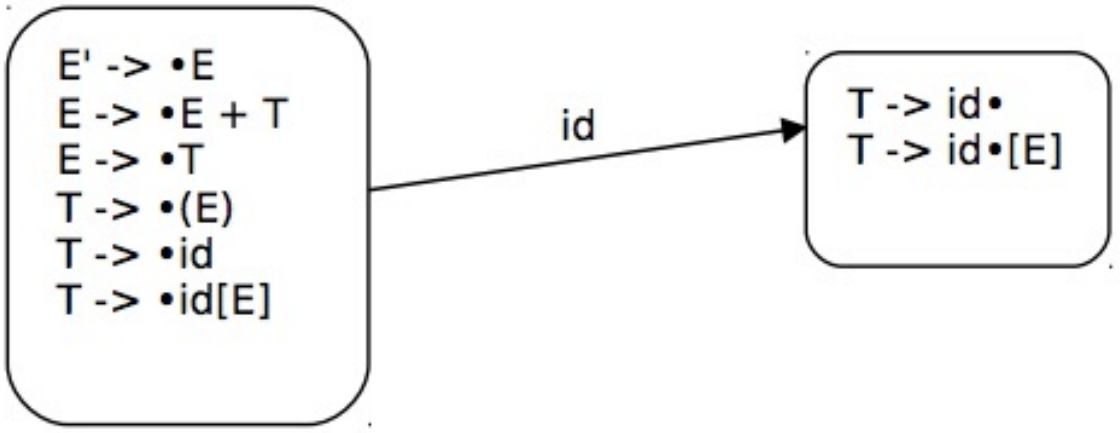
- In the SLR(1) parser, it is allowable for there to be <u>both shift and reduce items</u> in the same state as well as <u>multiple reduce items</u>
  - The SLR(1) parser will be able to determine which action to take as long as the FOLLOW sets are **disjoint**[可区分即可]



Example

- The first two LR(0) configurating sets entered if *id* is the first token of the input[识别id的前两个状态]
  - LR(0) parser: the set on the right side has a shift-reduce conflict
  - SLR(1) parser:
    - Compute Follow(T) = { +,),], $ }, i.e., only reduce on those tokens
      - Follow(T) = Follow(E) = {+,),], $}
    - The input [will shift and there is no conflict

```
E' -> E
E  -> E + T  | T
T  -> (E) | id | id[E]
```

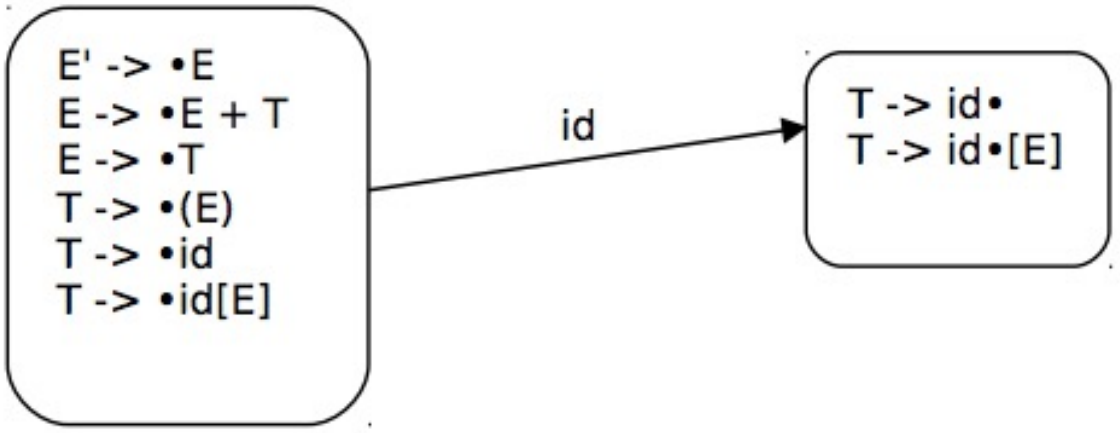


https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/110%20LR%20and%20SLR%20Parsing.pdf

Example

- The first two LR(0) configurating sets entered if *id* is the first token of the input[识别id的前两个状态]
  - LR(0) parser: the set on the right side has a shift-reduce conflict
  - SLR(1) parser:
    - Compute Follow(T) = { +,),], $ }, i.e., only reduce on those tokens
      - Follow(T) = Follow(E) = {+,),], $} id + id
    - The input [will shift and there is no conflict

E' -> E
E -> E + T | T
T -> (E) | id | id[E]

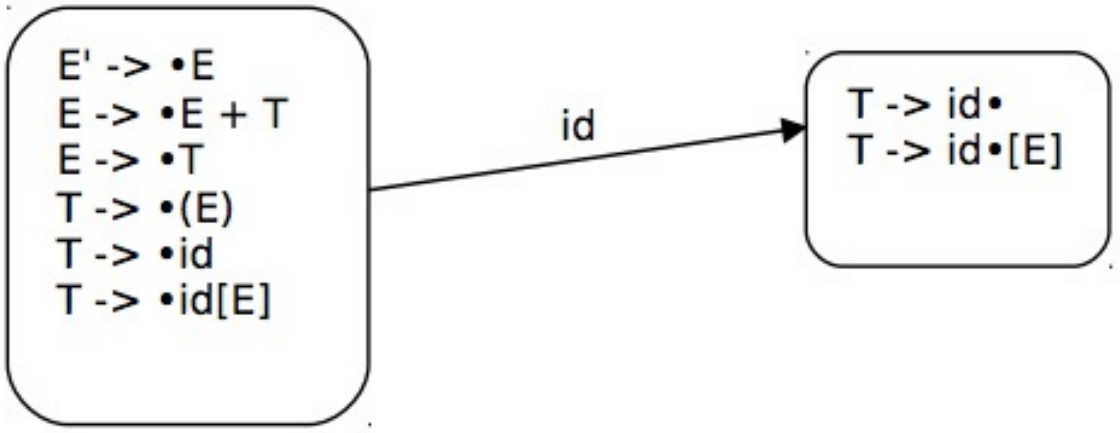


https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/110%20LR%20and%20SLR%20Parsing.pdf

Example

- The first two LR(0) configurating sets entered if *id* is the first token of the input[识别id的前两个状态]
  - LR(0) parser: the set on the right side has a shift-reduce conflict
  - SLR(1) parser:
    - Compute Follow(T) = { +,),], $ }, i.e., only reduce on those tokens
      - Follow(T) = Follow(E) = {+,),], $} id + id
    - The input [will shift and there is no conflict id[id]

E' -> E
E -> E + T | T
T -> (E) | id | id[E]

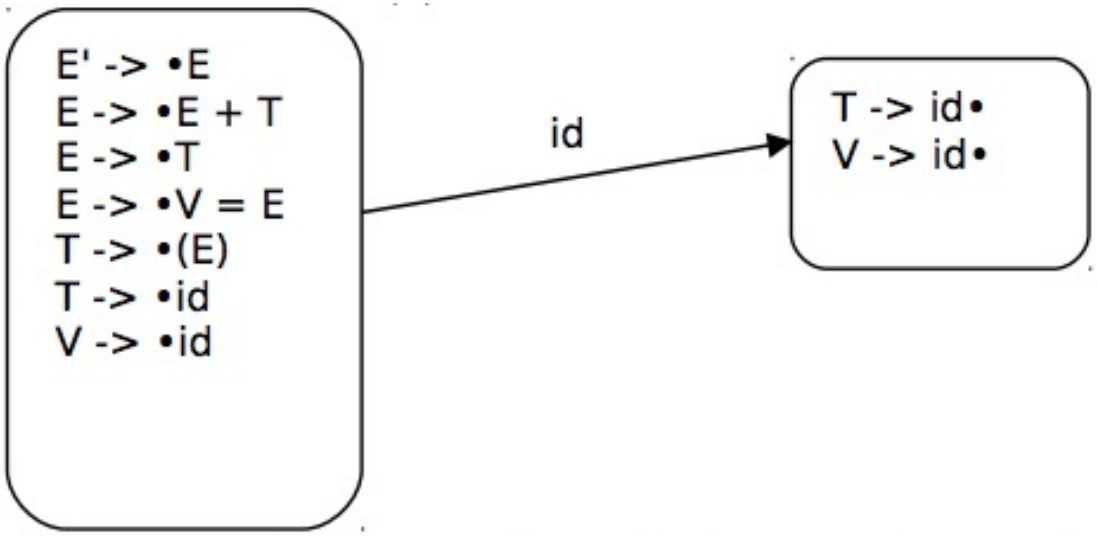


https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/110%20LR%20and%20SLR%20Parsing.pdf

Example (cont.)

- The first two LR(0) configurating sets entered if *id* is the first token of the input[识别id的前两个状态]
  - LR(0) parser: the right set has a reduce-reduce conflict
  - SLR(1) parser:
    - Capable to distinguish which reduction to apply depending on the next input token, no conflict
    - Compute Follow(T) = { +,), $ } and Follow(V) = { = }

E' -> E
E -> E + T | T | V = E
T -> (E) | id
V -> id



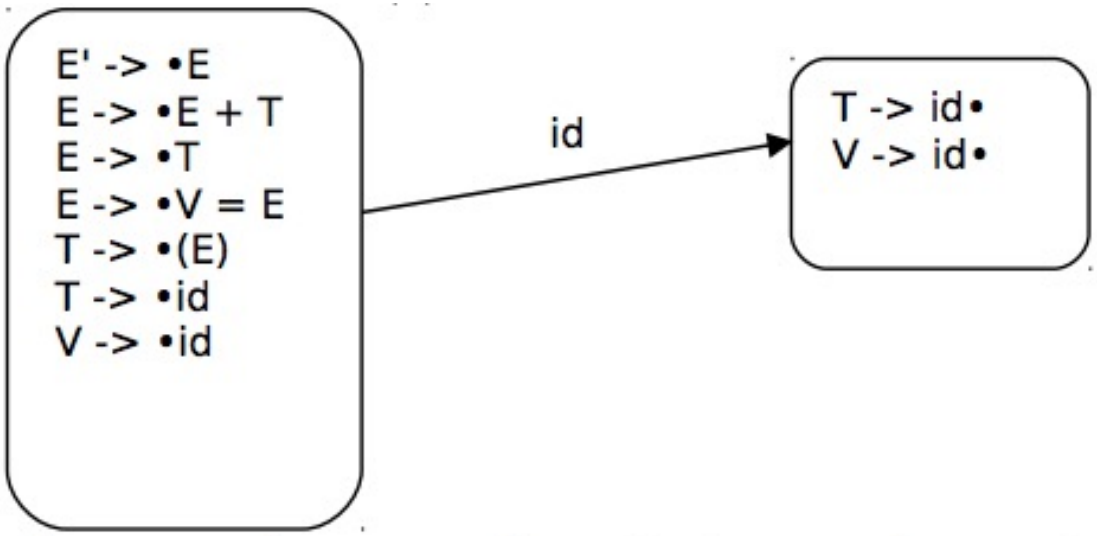
https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/110%20LR%20and%20SLR%20Parsing.pdf

Example (cont.)

- The first two LR(0) configurating sets entered if *id* is the first token of the input[识别id的前两个状态]
  - LR(0) parser: the right set has a reduce-reduce conflict
  - SLR(1) parser:
    - Capable to distinguish which reduction to apply depending on the next input token, no conflict
    - Compute Follow(T) = { +,), $ } and Follow(V) = { = }

id + id
id = id

E' -> E
E -> E + T | T | V = E
T -> (E) | id
V -> id



https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/110%20LR%20and%20SLR%20Parsing.pdf

SLR(1) Grammars[文法]

- A grammar is SLR(1) if the following two conditions hold for each configurating set[可区分]
- (1) For any item A → u·xv in the set, with terminal *x*, there is no complete item B → w· in that set with *x* in Follow(B)
  - In the table, this translates no shift-reduce conflict on any state
- (2) For any two complete items A → u· and B → v· in the set, the follow sets must be disjoint, i.e. Follow(A) ∩ Follow(B) is empty
  - This translates to no reduce-reduce conflict on any state
  - If more than one nonterminal could be reduced from this set, it must be possible to uniquely determine which using only one token of lookahead

SLR(1) Limitations[限制]

- SLR(1) vs. LR(0)
  - Adding just one token of lookahead and using the Follow set greatly expands the class of grammars that can be parsed without conflict
- When we have a completed configuration (i.e., dot at the end) such as X –> u·, we know that it is reducible[可归约]
  - We allow such a reduction whenever the next symbol is in Follow(X)[使用Follow集]
  - However, it may be that we should not reduce for every symbol in Follow(X), because the symbols below u on the stack preclude u being a handle for reduction in this case[Follow集不够]
  - In other words, SLR(1) states only tell us about the sequence on top of the stack, not what is below it on the stack
  - We may need to divide an SLR(1) state into separate states to differentiate the possible means by which that sequence has appeared on the stack[额外使用栈信息，FOLLOW是input buffer信息]

Example

- For input string: id = id, at $\mathbf{I_2}$ after having reduced id_{Left} to L
  - Initially, at S_0
  - Move to S_5, after shifting id to stack (S_5 is also pushed to stack)
  - Reduce, and back to S_0, and further GOTO S_2
    - S_5 has a completed item, and next '=' is in Follow(L)
    - S_5 and id are popped from stack, and L is pushed onto stack
    - GOTO(S_0, L) = S_2

```
S' -> S
S -> L = R
S -> R
L -> *R
L -> id
R -> L
```

```
I0:  S' -> •S
     S -> •L = R
     S -> •R
     L -> •*R
     L -> •id
     R -> •L

I1:  S' -> S•

I2:  S -> L• = R
     R -> L•

I3:  S -> R•

I4:  L -> *•R
     R -> •L
     L -> •*R
     L -> •id

I5:  L -> id•

I6:  S -> L =•R
     R -> •L
     L -> •*R
     L -> •id

I7:  L -> *R•

I8:  R -> L•

I9:  S -> L = R•
```


https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/110%20LR%20and%20SLR%20Parsing.pdf

Example (cont.)

- Choices upon seeing = coming up in the input:
  - Action[2, =] = s6
    - Move on to find the rest of assignment
  - Action[2, =] = r5
    - = ∈ Follow(R): S => L=R => *R = R
- Shift-reduce conflict
  - SLR parser fails to remember enough info
  - Reduce using R -> L only after seeing * or =

```
S' -> S
S -> L = R
S -> R
L -> *R
L -> id
R -> L
```

I_0:
```
S' -> •S
S -> •L = R
S -> •R
L -> •*R
L -> •id
R -> •L
```

I_1:
```
S' -> S•
```

I_2:
```
S -> L• = R
R -> L•
```

I_3:
```
S -> R•
```

I_4:
```
L -> *•R
R -> •L
L -> •*R
L -> •id
```

I_5:
```
L -> id•
```

I_6:
```
S -> L =•R
R -> •L
L -> •*R
L -> •id
```

I_7:
```
L -> *R•
```

I_8:
```
R -> L•
```

I_9:
```
S -> L = R•
```


https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/110%20LR%20and%20SLR%20Parsing.pdf

SLR(1) Improvement[改进]

- We don't need to see additional symbols beyond the first token in the input, we have already seen the info that allows us to determine the correct choice[展望信息已足够]
- Retain a little more of the left **context** that brought us here[历史路径]
  - Divide an SLR(1) state into separate states to differentiate the possible means by which that sequence has appeared on the stack
- Just using the entire Follow set is not discriminating enough as the guide for when to reduce[FOLLOW集不够]
  - For the example, the Follow set contains symbols that can follow *R* in any position within a valid sentence
  - But it does not precisely indicate which symbols follow *R* at this particular point in a derivation

LR(1) Parsing

- LR parsing adds the required extra info into the state
  - By redefining items to include a **terminal symbol** as an added component[让项目中包含终结符]
- General form of **LR(1) item**s[项目]
  - A –> $X_1 \ldots X_i \bullet X_{i+1} \ldots X_j$, a
  - We have states $X_1 \ldots X_i$ on the stack and are looking to put states $X_{i+1} \ldots X_j$ on the stack and then reduce
    - But only if the token following X_j is the terminal *a*
    - *a* is called the lookahead of the configuration
- The lookahead **only** works with completed items[完成项]
  - A –> $X_1 \ldots X_j \bullet$, a
  - All states are now on the stack, but only reduce when next symbol is *a* (*a* is either a terminal or $)
  - Multi lookahead symbols: A -> u•, a/b/c

LR(1) Parsing (cont.)

- When to reduce?
  - LR(0): if the configuration set has a **completed item** (i.e., dot at the end)
  - SLR(1): only if the next input token is in the **Follow()** set
  - LR(1): only if the next input token is exactly ***α*** (terminal or $)
  - Trend: **more and more precise**
- **LR(1) items**: LR(0) item + lookahead terminals
  - Many differ only in their lookahead components[仅展望不同]
  - The extra lookahead terminals allow to make parsing decisions beyond the SLR(1) capability, but with a big price[代价]
    - More distinguished items and thus more sets
    - Greatly increased Goto and Action table sizes

S' -> ·S

LR(0)

S' -> ·S, $

LR(1)

LR(1) Construction

- Configuration sets
  - Sets construction are essentially the same with SLR, but differing on Closure() and Goto()
    - Because with must respect the lookahead
- **Closure()**
  - For each item [A -> u·Bv, a] in *I*, for each production rule B -> w in G', add [B -> ·w, b] to *I*, if
    - b ∈ First(va) and [B -> ·w, b] is not already in *I*
  - Lookahead is the **First(va)**, which are what can follow B
    - v can be nullable

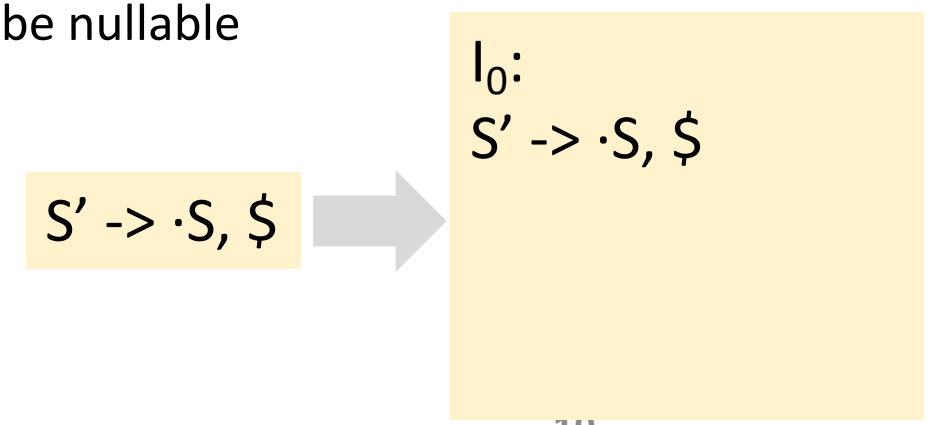
(0) S' -> S
(1) S -> XX
(2) X -> aX
(3) X -> b

S' -> ·S, $

LR(1) Construction

- Configuration sets
  - Sets construction are essentially the same with SLR, but differing on Closure() and Goto()
    - Because with must respect the lookahead
- **Closure()**
  - For each item [A -> u·Bv, a] in *I*, for each production rule B -> w in G', add [B -> ·w, b] to *I*, if
    - b ∈ First(va) and [B -> ·w, b] is not already in *I*
  - Lookahead is the **First(va)**, which are what can follow B
    - v can be nullable

(0) S' -> S
(1) S -> XX
(2) X -> aX
(3) X -> b



LR(1) Construction

- Configuration sets
  - Sets construction are essentially the same with SLR, but differing on Closure() and Goto()
    - Because with must respect the lookahead
- **Closure()**
  - For each item [A -> u·Bv, a] in *I*, for each production rule B -> w in G', add [B -> ·w, b] to *I*, if
    - b ∈ First(va) and [B -> ·w, b] is not already in *I*
  - Lookahead is the **First(va)**, which are what can follow B
    - v can be nullable

(0) S' -> S
(1) S -> XX
(2) X -> aX
(3) X -> b

S' -> ·S, $

I_0:
S' -> ·S, $
S -> .XX, First(ε$)

LR(1) Construction

- Configuration sets
  - Sets construction are essentially the same with SLR, but differing on Closure() and Goto()
    - Because with must respect the lookahead
- **Closure()**
  - For each item [A -> u·Bv, a] in *I*, for each production rule B -> w in G', add [B -> ·w, b] to *I*, if
    - b ∈ First(va) and [B -> ·w, b] is not already in *I*
  - Lookahead is the **First(va)**, which are what can follow B
    - v can be nullable

(0) S' -> S
(1) S -> XX
(2) X -> aX
(3) X -> b

S' -> ·S, $

I_0:
S' -> ·S, $
S -> .XX, First(ε$)
X -> .aX, First(X$)
X -> .b, First(X$)

LR(1) Construction

- Configuration sets
  - Sets construction are essentially the same with SLR, but differing on Closure() and Goto()
    - Because with must respect the lookahead
- **Closure()**
  - For each item [A -> u·Bv, a] in *I*, for each production rule B -> w in G', add [B -> ·w, b] to *I*, if
    - b ∈ First(va) and [B -> ·w, b] is not already in *I*
  - Lookahead is the **First(va)**, which are what can follow B
    - v can be nullable

(0) S' -> S
(1) S -> XX
(2) X -> aX
(3) X -> b

S' -> ·S, $

I_0:
S' -> ·S, $
S -> .XX, First(ε$)
X -> .aX, First(X$)
X -> .b, First(X$)

I_0:
S' -> ·S, $
S -> .XX, $
X -> .aX, a/b
X -> .b, a/b

LR(1) Construction (cont.)

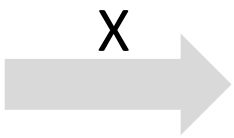
- **Goto(I, X)**
  - For item [A -> u·Xv, a] in *I*, Goto(I, X) = Closure ([A -> uX·v, a])
  - Basically the same Goto function as defined for LR(0)
    - But have to **propagate the lookahead**[传递] when computing the transitions
- Overall steps
  - Start from the initial set Closure([S' -> ·S, $])
  - Construct configuration sets following Goto(I, X)
  - Repeat until no new sets can be added

I_0:
S' -> ·S, $
S -> .XX, $
X -> .aX, a/b
X -> .b, a/b

LR(1) Construction (cont.)

- **Goto(I, X)**
  - For item [A -> u·Xv, a] in *I*, Goto(I, X) = Closure ([A -> uX·v, a])
  - Basically the same Goto function as defined for LR(0)
    - But have to **propagate the lookahead**[传递] when computing the transitions
- Overall steps
  - Start from the initial set Closure([S' -> ·S, $])
  - Construct configuration sets following Goto(I, X)
  - Repeat until no new sets can be added

I_0:
S' -> ·S, $
S -> .XX, $
X -> .aX, a/b
X -> .b, a/b

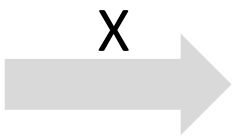


I_2:
S -> X.X, $

LR(1) Construction (cont.)

- **Goto(I, X)**
  - For item [A -> u·Xv, a] in *I*, Goto(I, X) = Closure ([A -> uX·v, a])
  - Basically the same Goto function as defined for LR(0)
    - But have to **propagate the lookahead**[传递] when computing the transitions
- Overall steps
  - Start from the initial set Closure([S' -> ·S, $])
  - Construct configuration sets following Goto(I, X)
  - Repeat until no new sets can be added

I_0:
S' -> ·S, $
S -> .XX, $
X -> .aX, a/b
X -> .b, a/b



I_2:
S -> X.X, $
X -> .aX, First(ε$)
X -> .b, First(ε$)

LR(1) Construction (cont.)

- **Goto(I, X)**
  - For item [A -> u·Xv, a] in *I*, Goto(I, X) = Closure ([A -> uX·v, a])
  - Basically the same Goto function as defined for LR(0)
    - But have to **propagate the lookahead**[传递] when computing the transitions
- Overall steps
  - Start from the initial set Closure([S' -> ·S, $])
  - Construct configuration sets following Goto(I, X)
  - Repeat until no new sets can be added

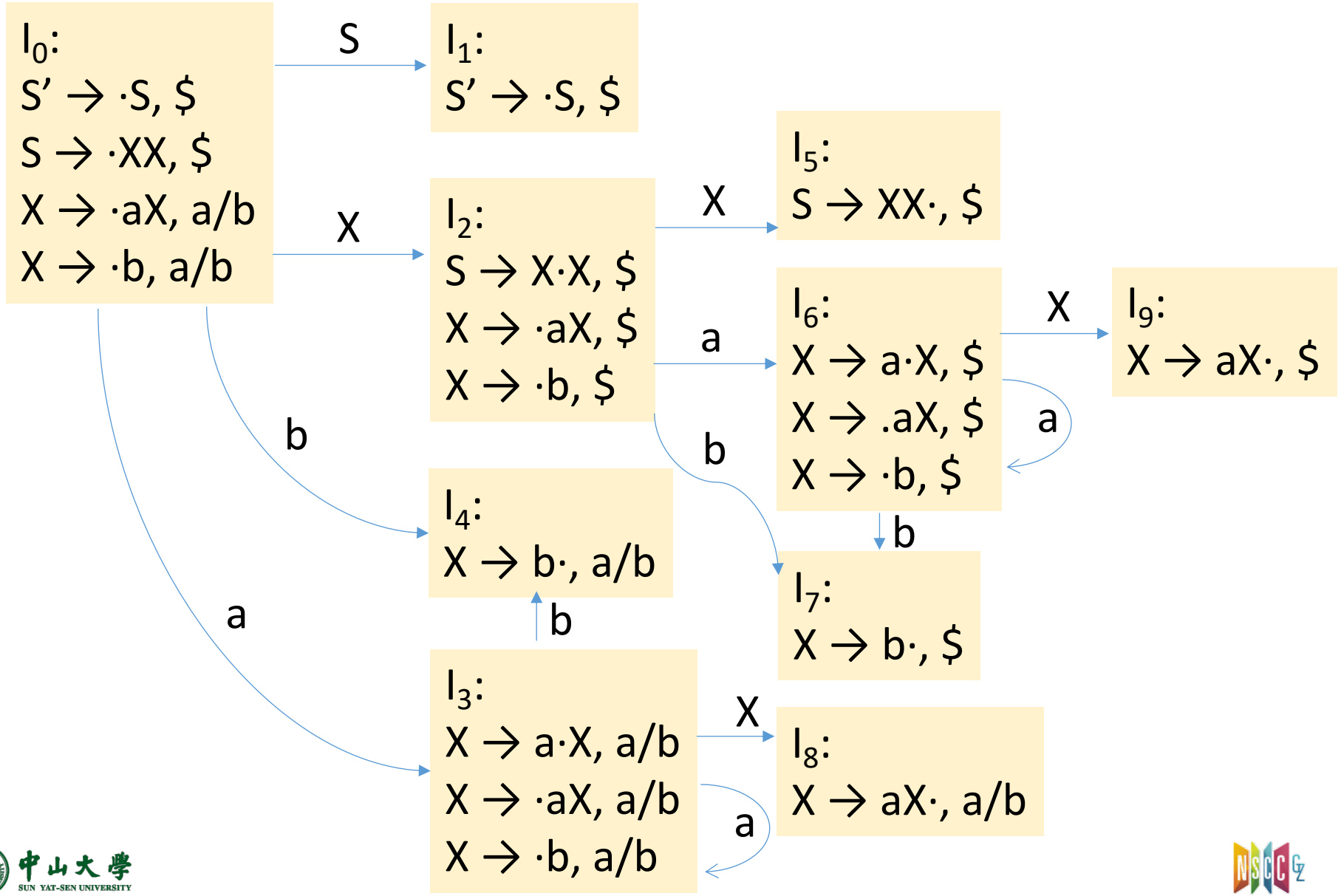
I_0:
S' -> ·S, $
S -> .XX, $
X -> .aX, a/b
X -> .b, a/b

X

I_2:
S -> X.X, $
X -> .aX, First(ε$)
X -> .b, First(ε$)

I_2:
S -> X.X, $
X -> .aX, $
X -> .b, $

Example

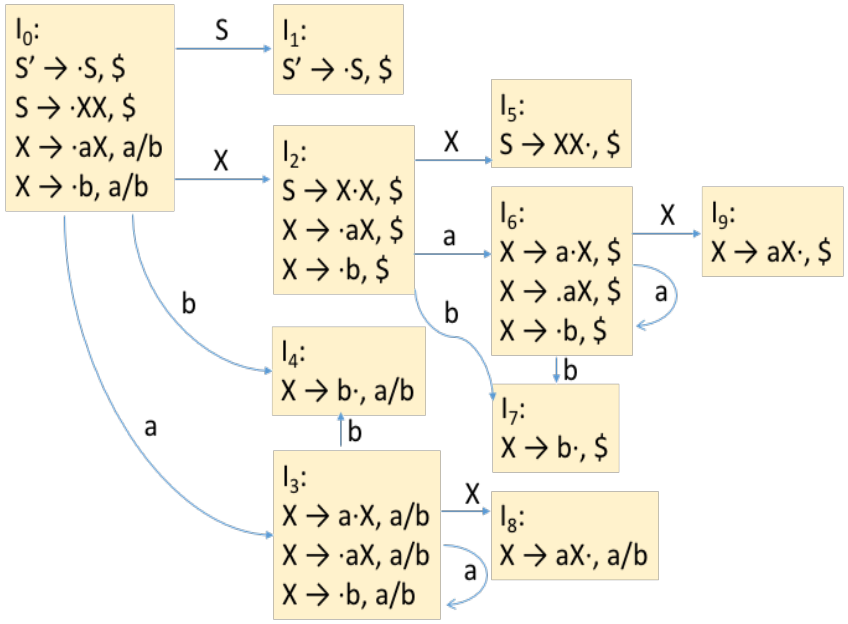

I0:
S' → ·S, $
S → ·XX, $
X → ·aX, a/b
X → ·b, a/b
S
I1:
S' → ·S, $
X
I2:
S → X·X, $
X → ·aX, $
X → ·b, $
X
I5:
S → XX·, $
a
I6:
X → a·X, $
X → .aX, $
X → ·b, $
X
I9:
X → aX·, $
a
b
b
b
I4:
X → b·, a/b
I7:
X → b·, $
a
b
I3:
X → a·X, a/b
X → ·aX, a/b
X → ·b, a/b
X
I8:
X → aX·, a/b
a

LR(1) Parse Table[解析表]

- Shift[移进]
  - Same as LR(0) and SLR(1)
  - Don't care the lookahead symbols
- Reduce[归约]
  - Don't use Follow set (too coarse-grain[粗粒度])
  - Reduce only if input matches lookahead for item
- ACTION and GOTO[表格]
  - If [A -> α·aβ, b] ∈ S_i and goto(S_i, a) = S_j, Action[i, a] = sj
    - Shift *a* and goto state *j*
    - Same as SLR(1)
  - If [A -> α·, a] ∈ S_i , Action[i, a] = r[R]
    - Reduce R: A -> α if input matches *a*
    - For SLR, reduced if put input matches *Follow(A)*

Example

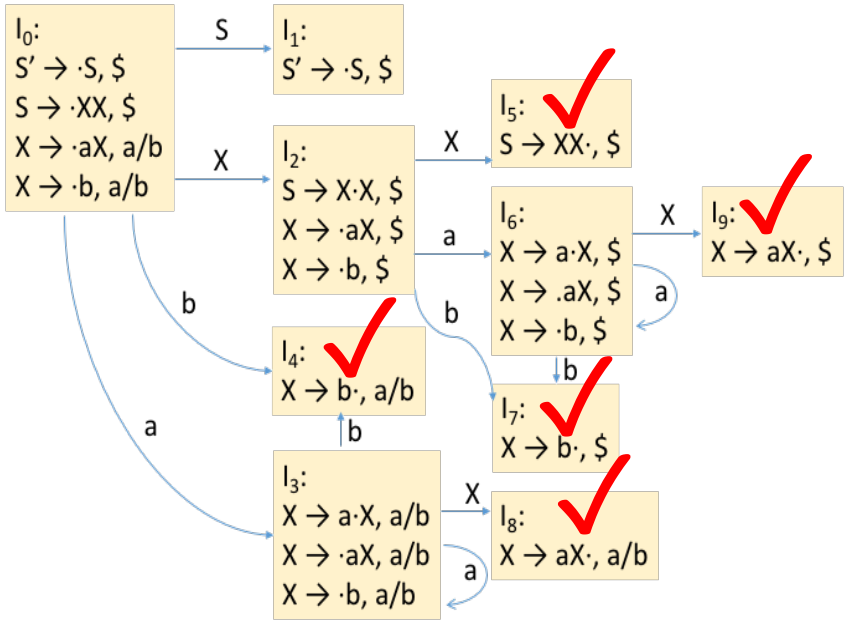
(0) S’ -> S
(1) S -> XX
(2) X -> aX
(3) X -> b



State	ACTION			GOTO	
	a	b	$	S	X
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		

Example

(0) S’ -> S
(1) S -> XX
(2) X -> aX
(3) X -> b



State	ACTION			GOTO	
	a	b	$	S	X
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		