中山大學 SUN YAT-SEN UNIVERSITY 计算机学院（软件学院） SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Compilation Principle
编 译 原 理

第13讲：语法分析(10)

张献伟

xianweiz.github.io

DCS290, 4/6/2023

中山大學 SUN YAT-SEN UNIVERSITY

Review Questions

- Action table entries can be si and rj, what do they mean?

    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 → u|w

    A → X • YZ
    Y → •u
    Y → •w

- What is augmented grammar?

    Add one extra rule S' → S to guarantee only one 'acc' in the table

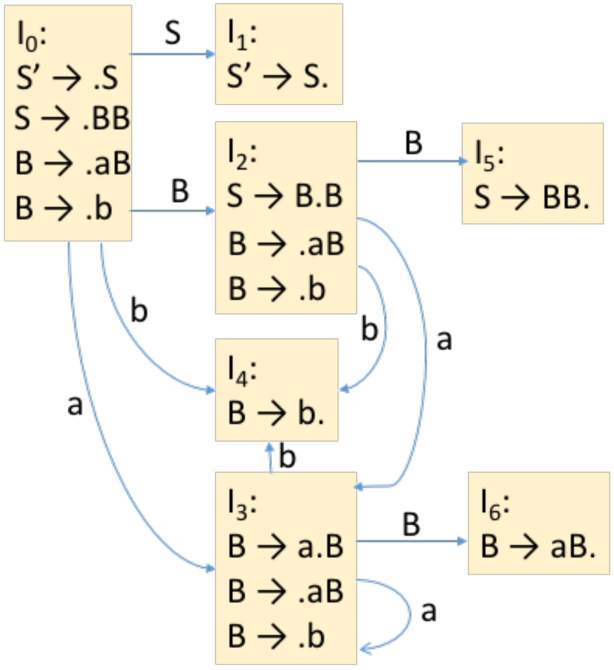
- What are the possible items of S' → S?

    S' → •S: initial item, haven't seen any input symbol

    S' → S•: accept item, have reduced the input string to start symbol

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
- Next: **reduce** by X → β if[归约]
  - s contains item X → β·
- Or, **shift** if[移进]
  - s contains item X → β·*tω*
  - Equivalent to saying s has a transition labeled *t*



LR(0) Parsing (cont.)

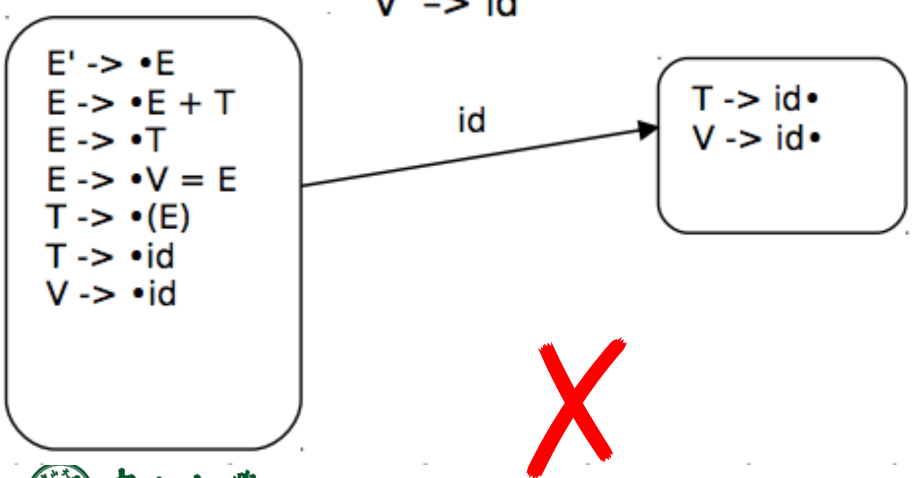
- The parser must be able to determine what action to take in each state <u>without looking at any further input</u> symbols [没有展望就可以决定动作]
  - i.e. by only considering what the parsing stack contains so far
  - This is the ‘**0**’ in the parser name
- In a LR(0) table, each state must only shift or reduce[确定性移进或归约]
  - Thus an LR(0) configurating set can only have <u>exactly one</u> reduce item[每个归约item自成一个状态]
    - cannot have both shift and reduce items; otherwise, conflicts

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		

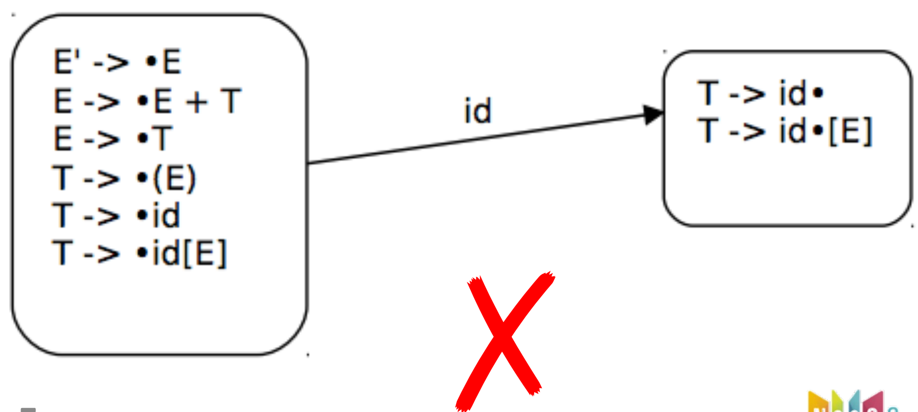
LR(0) Conflicts[冲突]

- 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 $Y \rightarrow \omega\cdot t\sigma$

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



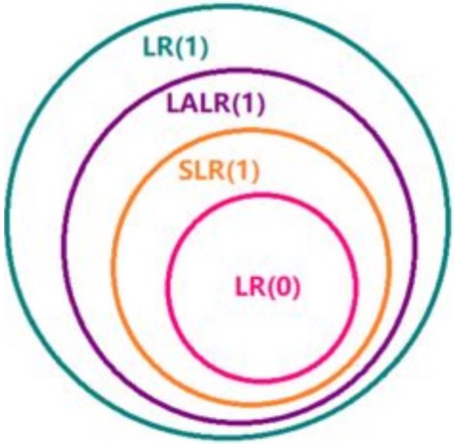
E' -> E
E -> E + T | T
T -> (E) | id | id[E]



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 LR, not used much in practice[实际不常使用]
  - 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

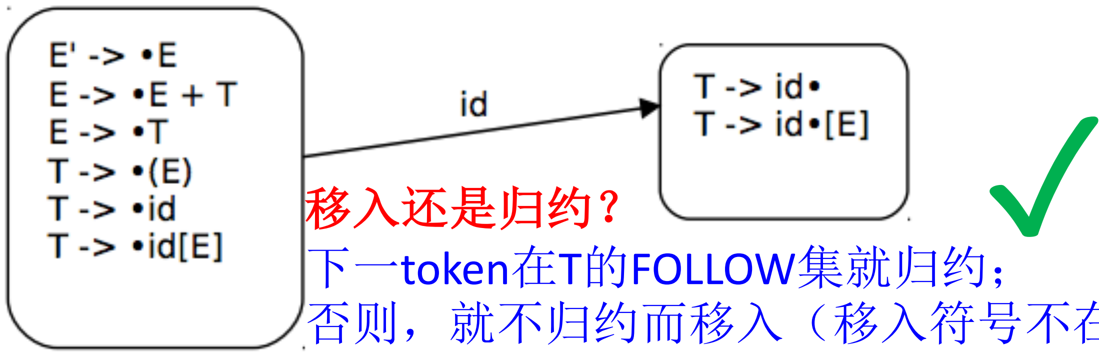
- LR(0) conflicts are generally caused by **reduce** actions
  - If the item is complete (A → α.), the parser must choose to reduce[项目形式完整就归约]
    - Is this always appropriate?
    - The next upcoming token may tell us sth different
  - What tokens may tell the reduction is not appropriate?
    - Perhaps **FOLLOW(A)** could be useful here
      - If the sequence on top of the stack could be reduced to the nonterminal A, what tokens do we expect to find as the next input?

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		

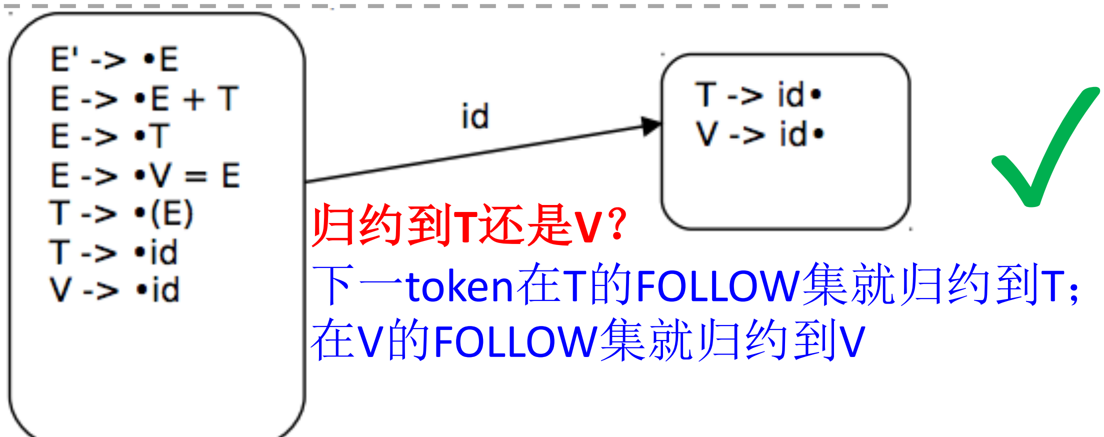
- **SLR** = Simple LR
  - Use the same LR(0) configurating sets and have the same table structure and parser operation[表结构一致]
  - The difference comes in assigning table actions[动作填充不同]
    - Use <u>one token of lookahead</u> to help arbitrate among the conflicts
    - Reduce only if the next input token 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 both shift and reduce items in the same state as well as multiple reduce items
  - 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]



移入还是归约？
下一token在T的FOLLOW集就归约；
否则，就不归约而移入（移入符号不在FOLLOW）

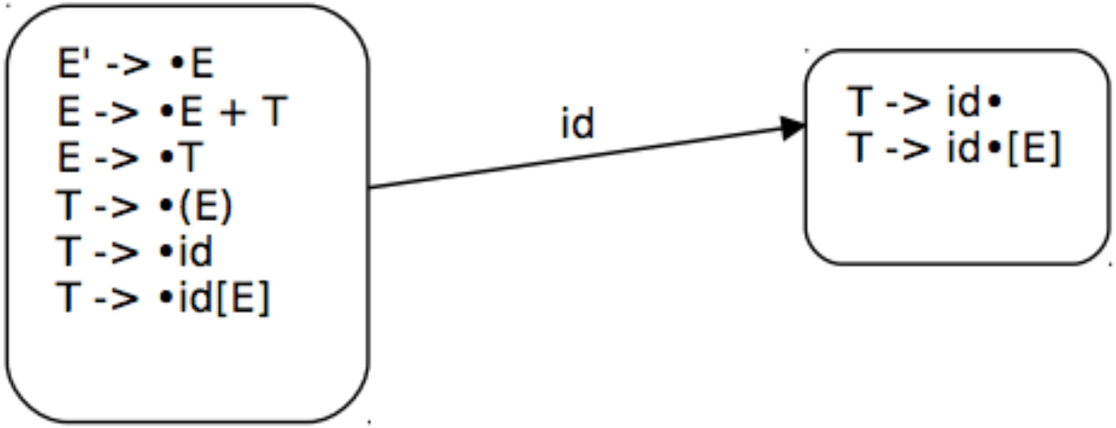


归约到T还是V？
下一token在T的FOLLOW集就归约到T；
在V的FOLLOW集就归约到V

Example

- The first two LR(0) configurating sets entered if *id* is the first token of the input[用于识别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]**: next input is [, not in FOLLOW(T), **shift**
    - **id + id**: next input is +, in FOLLOW(T), **reduce**

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

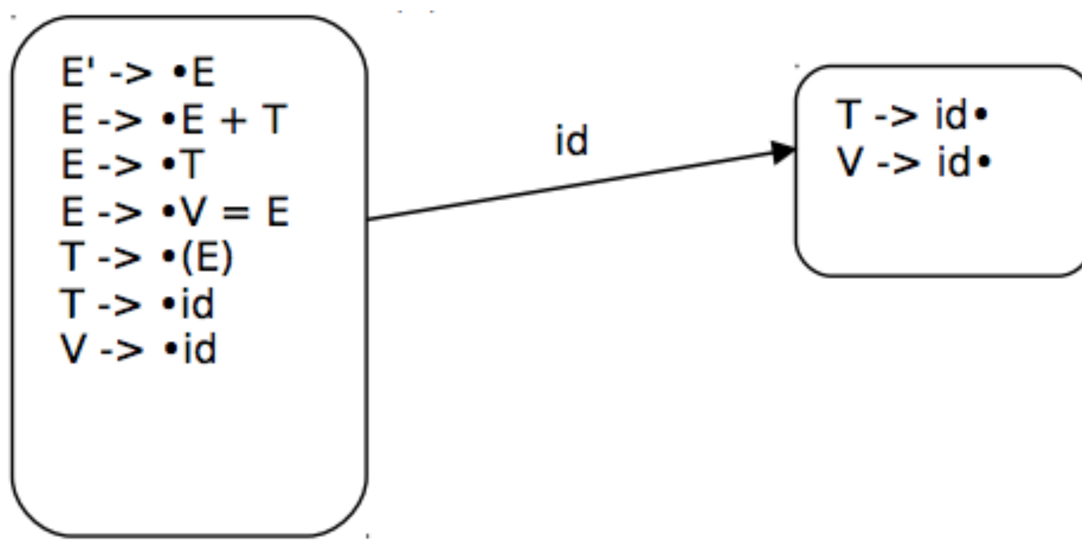


https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/110%20LR%20and%20SLR%20Parsing.pdf

Example (cont.)

- The first two LR(0) configurating sets entered if *id* is the first token of the input[用于识别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)[无移入-归约冲突]
  - 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[无归约-归约冲突]
  - 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

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

SLR(1) Limitations[限制]

- SLR(1) vs. LR(0)
  - Adding just <u>one token of lookahead</u> and using the <u>FOLLOW set</u> 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 <u>top of the stack</u>, 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]

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

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

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

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 =

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)

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•

SUN YAT-SEN UNIVERSITY 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) items**[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 ***a*** (terminal or $)
  - Trend: **more and more precise**
- **LR(1) item**s: 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	S' -> ·S, $
LR(0)	LR(1)

LR(1) Construction

- Configuration sets
  - Sets construction are essentially the same with SLR, but differing on Closure() and Goto()
    - Because we 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

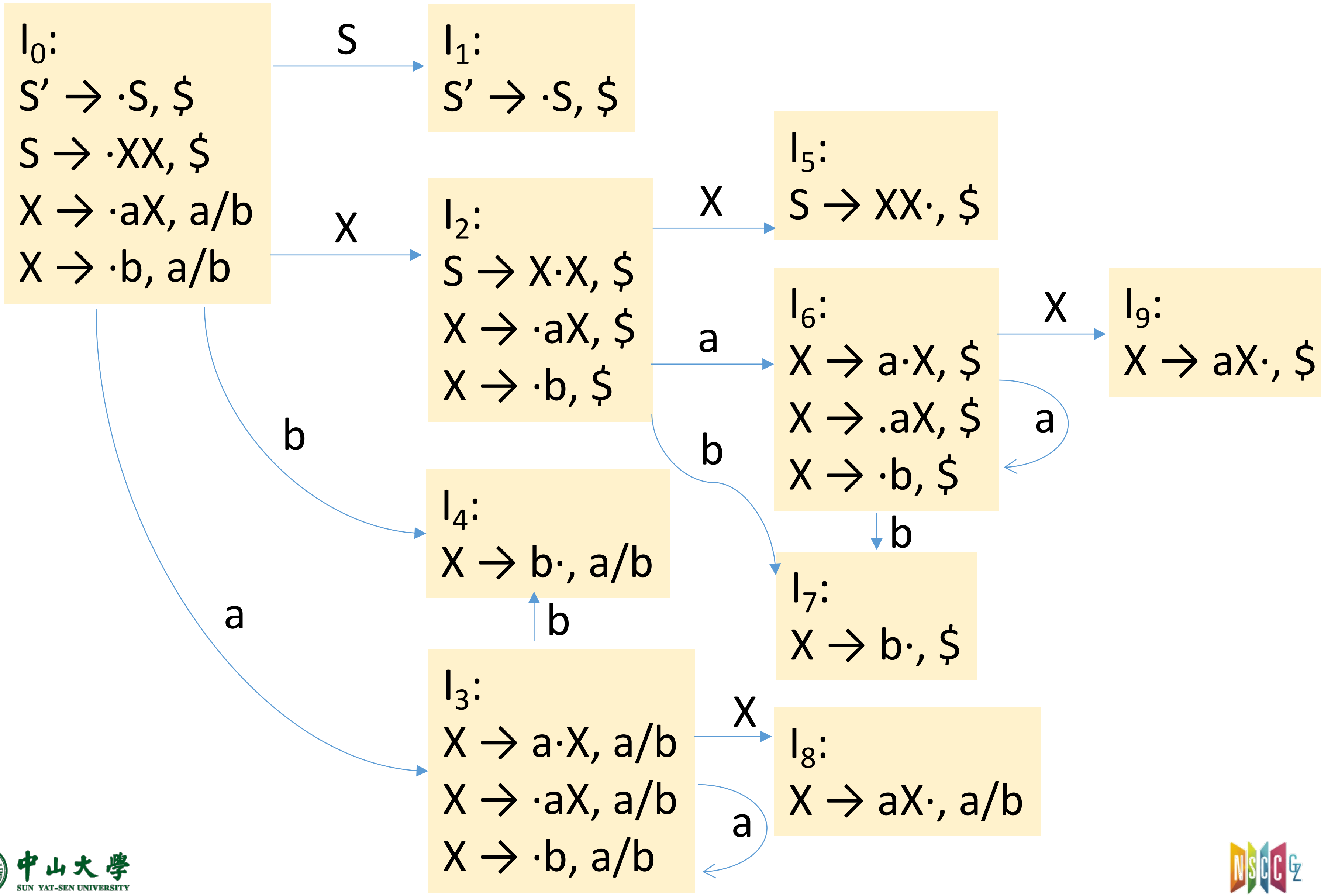
→ X →

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

→

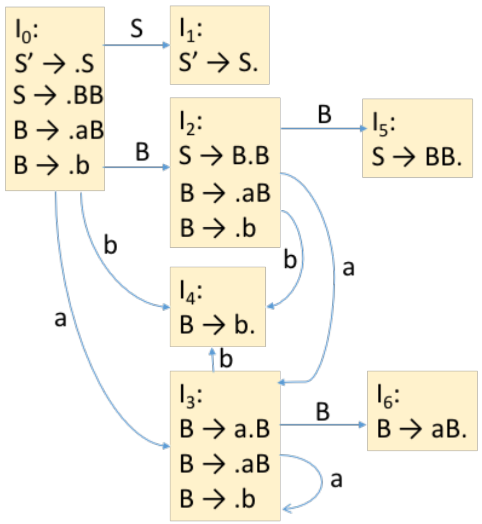
I_2:
S -> X.X, $
X -> .aX, $
X -> .b, $

Example



SUN YAT-SEN UNIVERSITY

Example (cont.)



LR(0)

String: bab

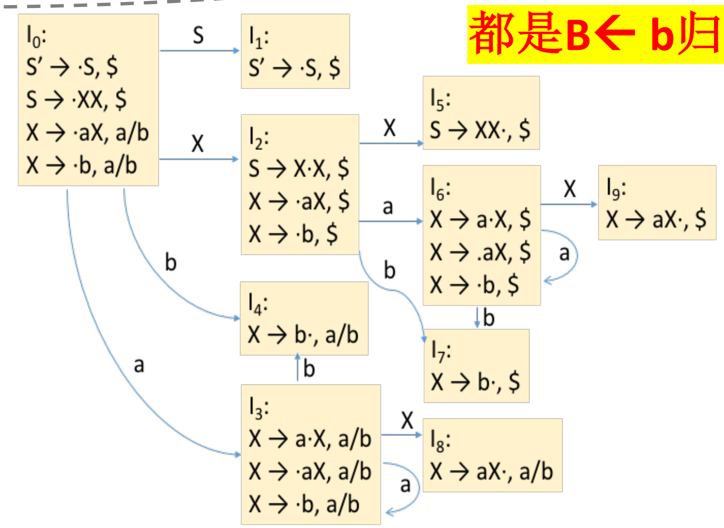
#bab$ => b#ab$ => B#ab$ => Ba#b$

0   0 4   0 2   0 2 3

=> Bab#$ => BaB#$ => BB#$ => S#$

0 2 3 4   0 2 3 6   0 2 5   0 1

都是B← b归约



LR(1)

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$

0   0 4   0 2   0 2 6

=> Bab#$ => BaB#$ => BB#$ => S#$

0 2 3 7   0 2 3 9   0 2 5   0 1

String: abb

都是B← aB归约

#abb$ => a#bb$ => ab#b$ => aB#b$

0   0 3   0 3 4   0 3 8

=> B#b$ => Bb#$ => BB#$ => S#$

0 2   0 2 7   0 2 5   0 1