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(9)

张献伟

xianweiz.github.io

DCS290, 4/16/2024

Quiz

Handwritten, or email to
chenhq79@mail2.sysu.edu.cn

QUIZ

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	r2	r2	r2		5

- Q1: explain action table entries si or rj.

  si: shift the input symbol and move to state *i*

  rj: reduce by production numbered *j*

G:
S → ABC
B → b|c

- Q2: augment the grammar G.

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

- Q3: what is the initial state (S_0) of G’?

  Closure(S’ → •S) = { S’ → •S, S → •ABC }

G’:
S’ → S
S → ABC
B → b|c

- Q4: what does S → ABC• mean?

  We have seen the body ABC and it is time to reduce ABC to S

- Q5: what is the closure of S → A•BC?

  { S → A•BC, B → •b, B → •c }

Deadline[截止时间]

- 宽限期(Slip Days)：proj2/3/4共10天
  - 可用于延期提交，宽限期内不扣分
    - 每人共分配10天，可拆分使用，不可转借
    - 每个slip day为自然日，使用单位为整数日
    - 如使用，截止时间前需填写问卷申请
  - 迟交扣分规则
    - 不填写问卷或宽限期无剩余，视为迟交
    - 迟交每一天扣除得分10%

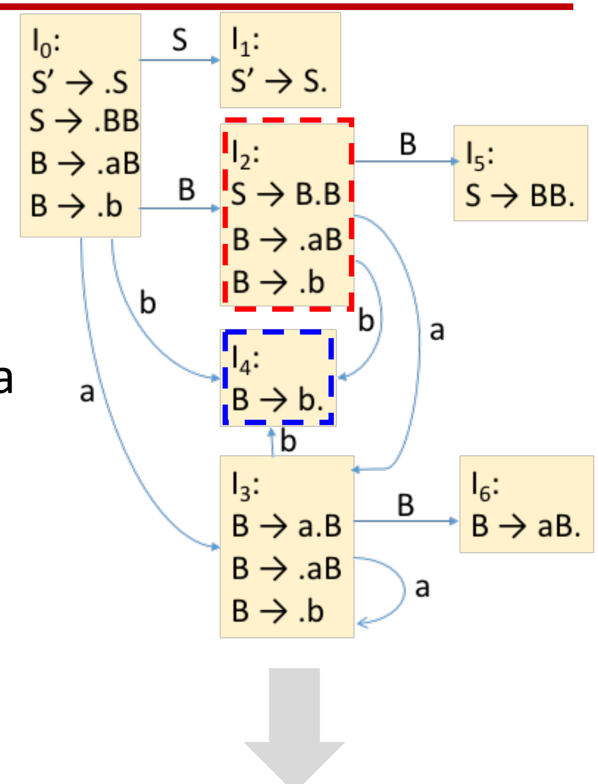
proj1昨晚已经截止，我们会尽快完成批改！未提交的同学：如果截止前填写了问卷，自动宽限5天，请于4.2/周二23:59:59前提交（宽限期内扣除得分的5%），宽限期外每延一天再扣除得分的10%；没有填写问卷的同学，从今天开始每天扣除10%。proj2/3/4将类似执行，具体有所调整。

- 理论
  - 随机点名
    - 缺席优先
  - 随机提问
    - 后排优先
  - 随机测试
    - 不定时间
- 实验
  - 个人完成
    - 杜绝抄袭
  - 按时提交
    - 硬性截止
  - 侧重代码实现
    - 简略报告

https://student.cs.uwaterloo.ca/~cs350/F14/assignments/slipdays.html

Build Parse Table from DFA

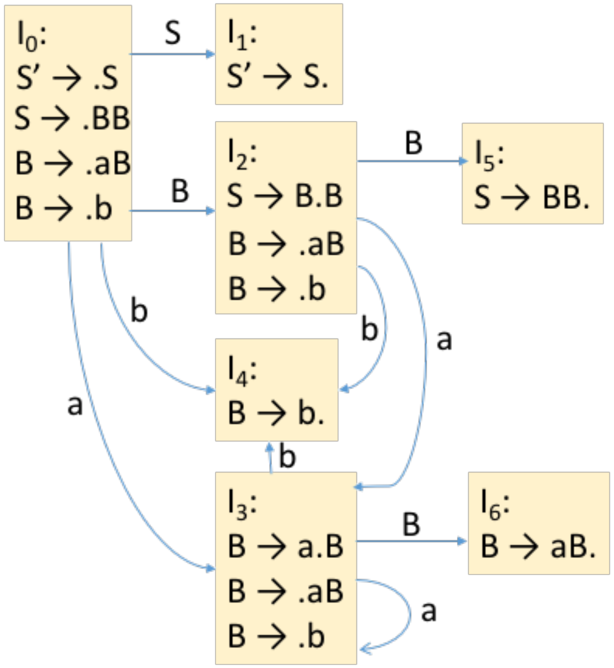
- ACTION: [*state*, *terminal symbol*]
- GOTO: [*state*, *non-terminal symbol*]
- ACTION[动作]
  - If [A→α·aβ] is in S_i and goto(S_i, a) = S_j, where "a" is a terminal, then ACTION[S_i, a] = shift j (s*j*)
  - If [A→α·] is in S_i and A→α is rule numbered j, then ACTION[S_i, a] = reduce j (r*j*)
  - If [S' →S·] is in S_i then ACTION[S_i, $] = accept
  - If no conflicts among 'shift' and 'reduce' (the first two 'if's), then this parser is able to parse the given grammar
- GOTO[跳转]
  - if goto(S_i, A) = S_j then GOTO[S_i, A] = *j*
- All entries not filled are rejects



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) 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*



http://openclassroom.stanford.edu/MainFolder/courses/Compilers/docs/slides/08-05-slr-parsing-annotated.pdf

LR(0) Parsing (cont.)

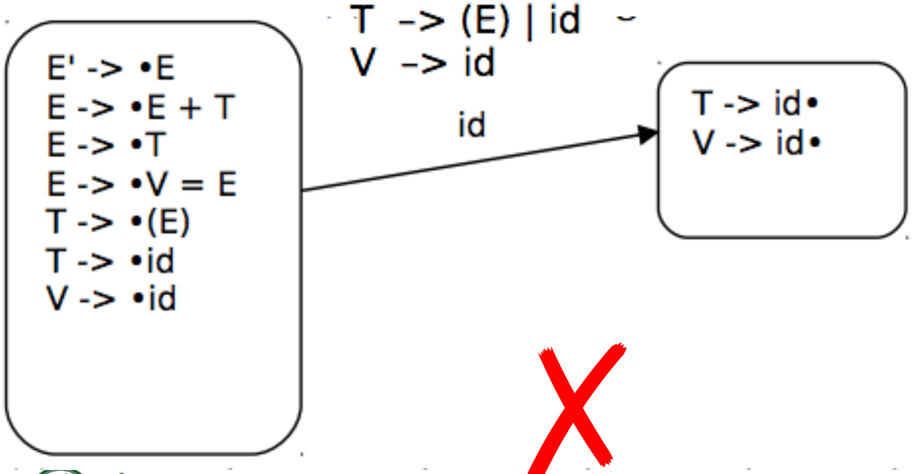
- The parser must be able to determine what action to take in each state <u>without looking at any further input</u> symbols [没有展望就可以决定动作]
  - i.e. by only considering what the parsing stack contains so far
  - ☞ This is the '**0**' in the parser name
- In a LR(0) table, each state must only shift or 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		

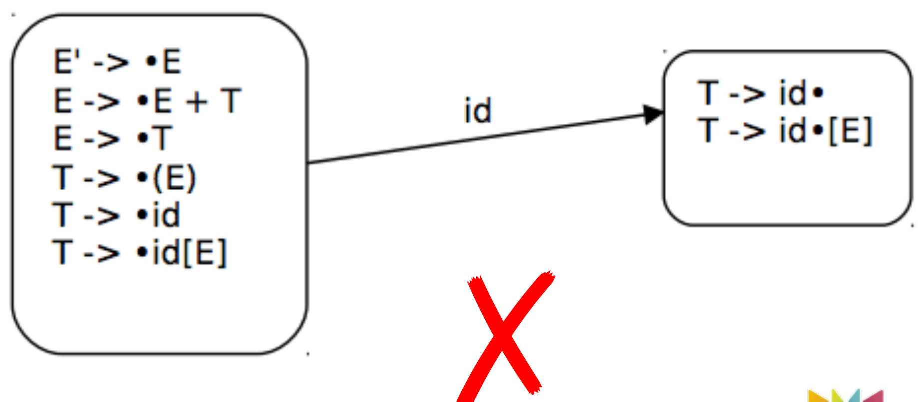
LR(0) Conflicts[冲突]

- 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]

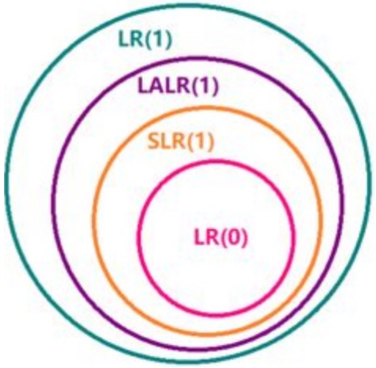


https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/110%20LR%20and%20SLR%20Parsing.pdf

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

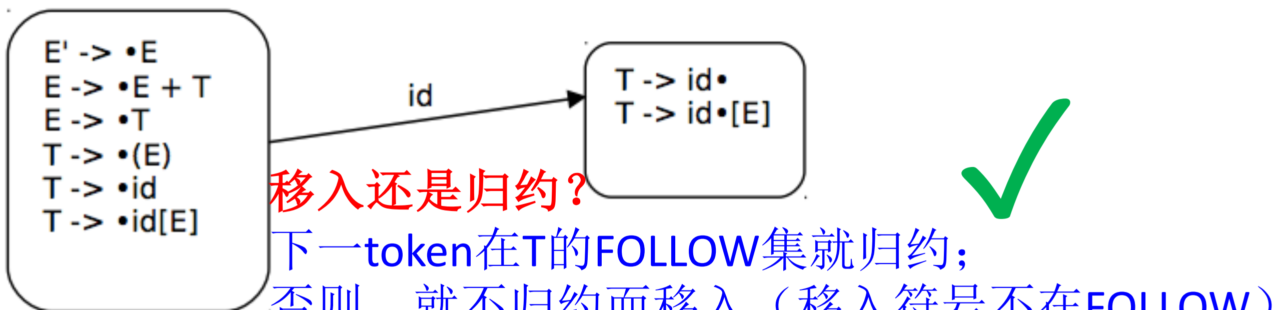
- LR(0) conflicts are generally caused by **reduce** actions
  - If the item is complete (A → α·), the parser must choose to reduce[项目形式完整就归约]
    - Is this always appropriate?
    - The next upcoming token may tell us something different
  - What tokens may tell the reduction is not appropriate?
    - 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?

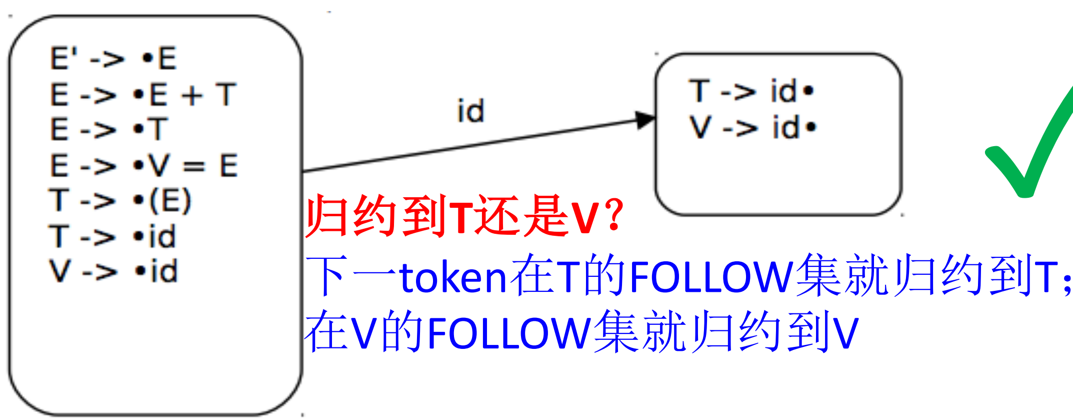
State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

- **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 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 non-terminal 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]



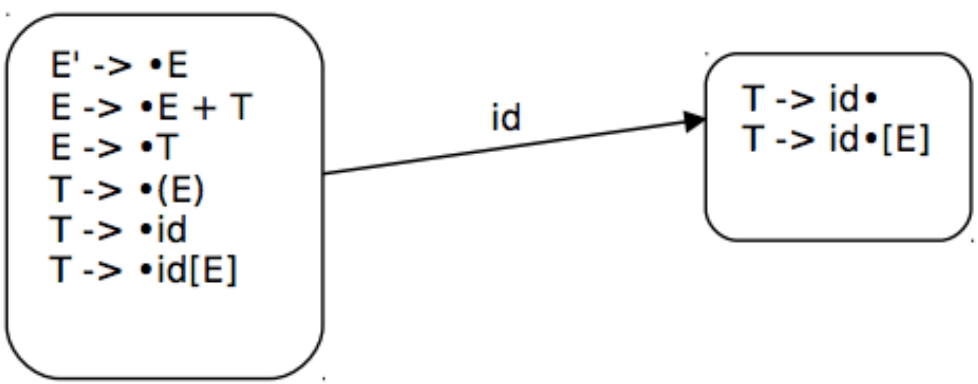


https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/110%20LR%20and%20SLR%20Parsing.pdf

Example

- The first two LR(0) configurating sets entered if *id* is the first token of the input[用于识别id所处的前两个状态]
  - 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]



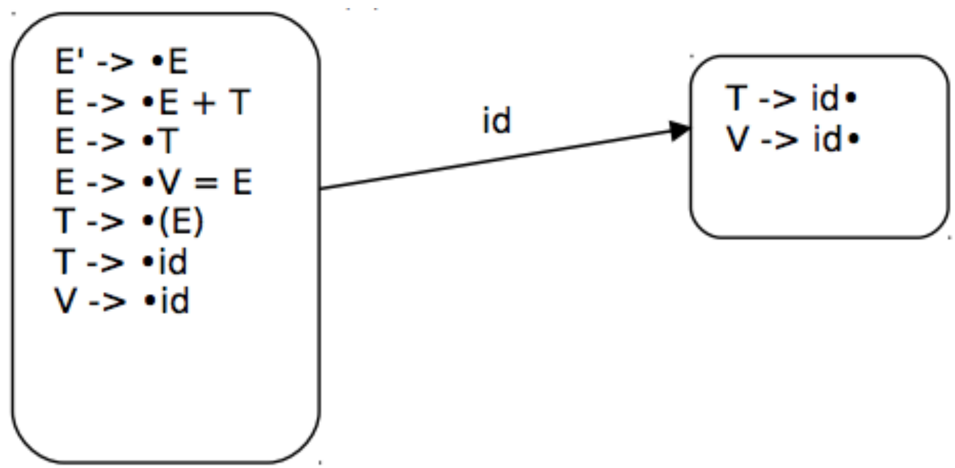
https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/110%20LR%20and%20SLR%20Parsing.pdf

Example (cont.)

- The first two LR(0) configurating sets entered if *id* is the first token of the input[用于识别id所处的前两个状态]
  - 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 => T -> id
id = id => V -> 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

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集不足够]
  - 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[状态是怎么转移过来的？]
  - 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 **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 rest of assignment
    - S => L=R => L=L => …
  - Action[2, =] = r5
    - = ∈ FOLLOW(R): S => L=R => L=L => L=*R => L = *L => …
- 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•

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]
  - $A \rightarrow X_1 \ldots X_i \bullet X_{i+1} \ldots X_j$ **, 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]
  - $A \rightarrow X_1 \ldots X_j \bullet$, 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

SUN YAT-SEN UNIVERSITY

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

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 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 symbols are the **FIRST(va)**, which are what can follow B
    - v can be nullable

(0) S' -> S
(1) S -> BB
(2) B -> aB
(3) B -> b

S' -> ·S, $

→

I_0:
S' -> ·S, $
S -> ·BB, First(ε$)
B -> ·aB, First(B$)
B -> ·b, First(B$)

→

I_0:
S' -> ·S, $
S -> ·BB, $
B -> ·aB, a/b
B -> ·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 -> ·BB, $
B -> ·aB, a/b
B -> ·b, a/b

—B→

I_2:
S -> B·B, $
B -> ·aB, First(ε$)
B -> ·b, First(ε$)

→

I_2:
S -> B·B, $
B -> ·aB, $
B -> ·b, $