CS453
LR(1), LALR, AMBIGUITY

LR(1), LALR, Ambiguity

The plan:

Shift reduce parsing

LR(1) and LALR

Ambiguous grammars

- Precedence and associativity rules
- Dangling else

Java-cup

precedence and associativity declarations

Shift Reduce conflict in dangling else

SLR still not good enough wrt reduce

Grammar for C style assignment:

0: S' → S$ 1: S → V = E 2: S → E 3: E→V 4:V→x 5: V→*E

E is a simplified expression, in reality more complex (E+T …)

Let's build the SLR parse table:

	x	*	=	$	S	V	E
1	s5	s6			g2	g3	g4
2				a			
3			**s7,r3**				
4				r2			
5			r4	r4			
6	s5	s6				g9	g8
7	s5	s6				g9	g10
8			r5	r5			
9			r3	r3			
10			r1	r1			

Is this grammar ambiguous?

NO

But we still have a shift reduce conflict.

We need to look ahead and decide when to reduce.

LR(1) look ahead sets

LR(1) items take a look ahead into account: (A→ α . β , z)

item is pair: (dotted rule, look-ahead symbol)

indicating α is on top of stack, and β z on input, but z is not part of A

We will now reduce to A, not on follow(A), but if αβ is on top of stack and z on input, i.e. we have more precise information.

Closure(I)=
repeat
for all (A→ α . Xβ , z) in I
for any X→γ
for any w in First(βz)
I+= (X→.γ , w)
until I does no change
return I

Goto(I,X)=
J={}
for all (A→ α . Xβ , z) in I
J+=(A→ αX . β , z)
return Closure(J)

Reduce:
R ={}
for all (A→ α.) in I
R+= (I, z, A→α)

(I, z, A→α) means:
in State I,
on symbol z
reduce: A→α

LR(1) state diagram: look-ahead sets

0: S' → S$ 1: S → V = E 2: S → E 3: E→V 4:V→x 5: V→*E

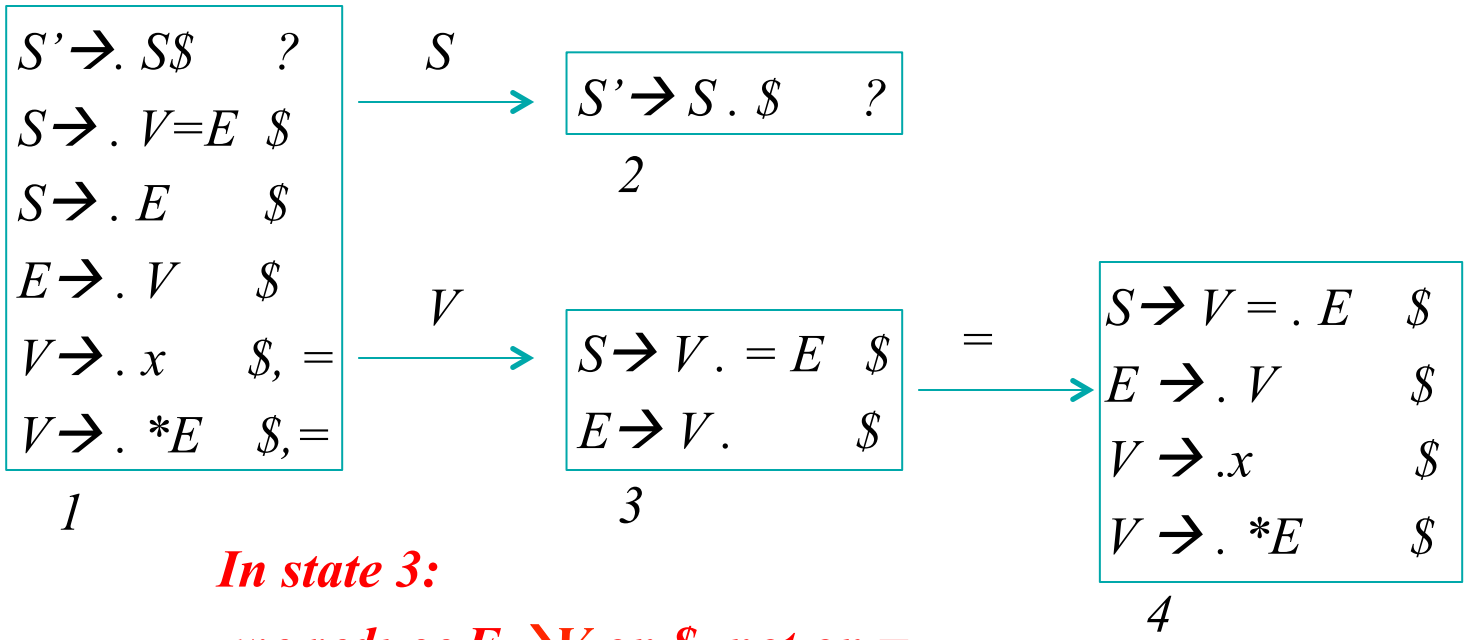
We will not shift the $, so the look ahead symbol for S' does not matter

This is indicated by a ?

S'→. S$	?	when we have	S'→. S$	?
S→ . V=E	$	the same dotted	S→ . V=E	$
S→ . E	$	rule with	S→ . E	$
E→ . V	$	different	E→ . V	$
V→ . x	$	look aheads	V→ . x	$, =
V→ . *E	$	we combine	V→ . *E	$, =
V→ . x	=	them in		
V→ . *E	=	sets		

LR(1) state diagram (incomplete)

0: S' → S$ 1: S → V = E 2: S → E 3: E→V 4:V→x 5: V→*E



In state 3:

we reduce E→V on $, not on =

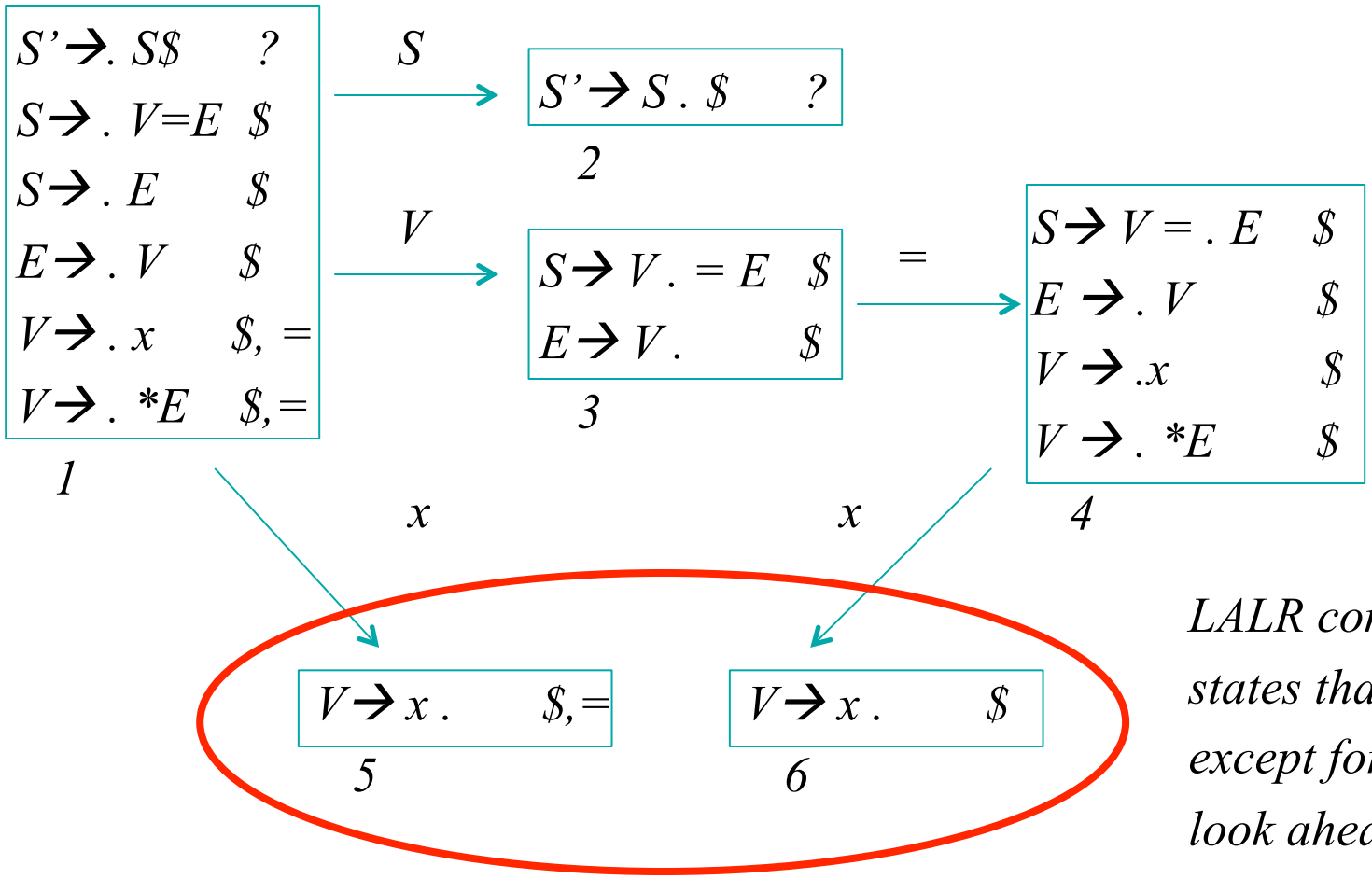
(even though = is in follow(E))

we shift on =

conflict resolved

LALR(1) Look Ahead LR(1) : state diagram (incomplete)

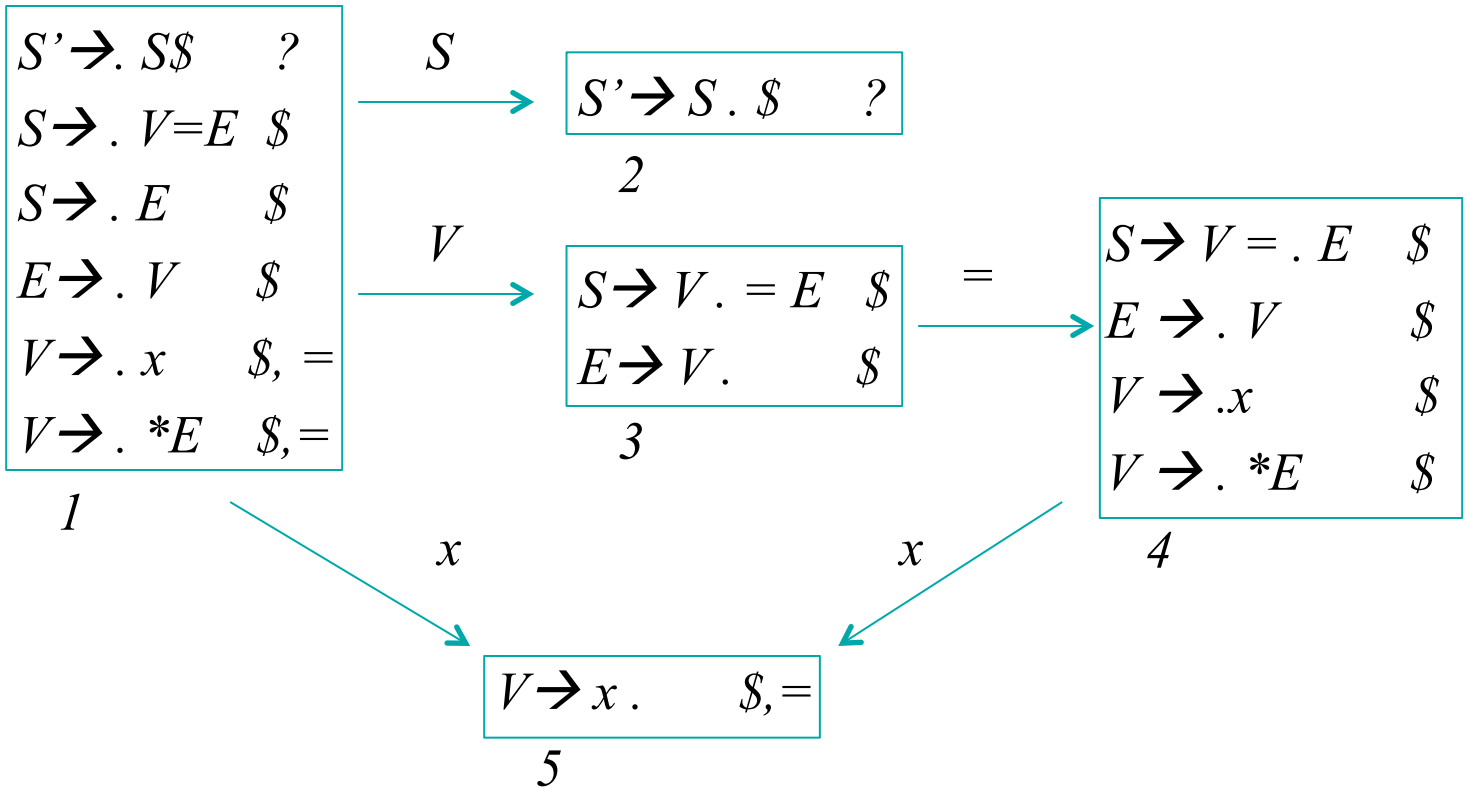
0: S' → S$ 1: S → V = E 2: S → E 3: E→V 4:V→x 5: V→*E



LALR combines states that are equal except for the look aheads

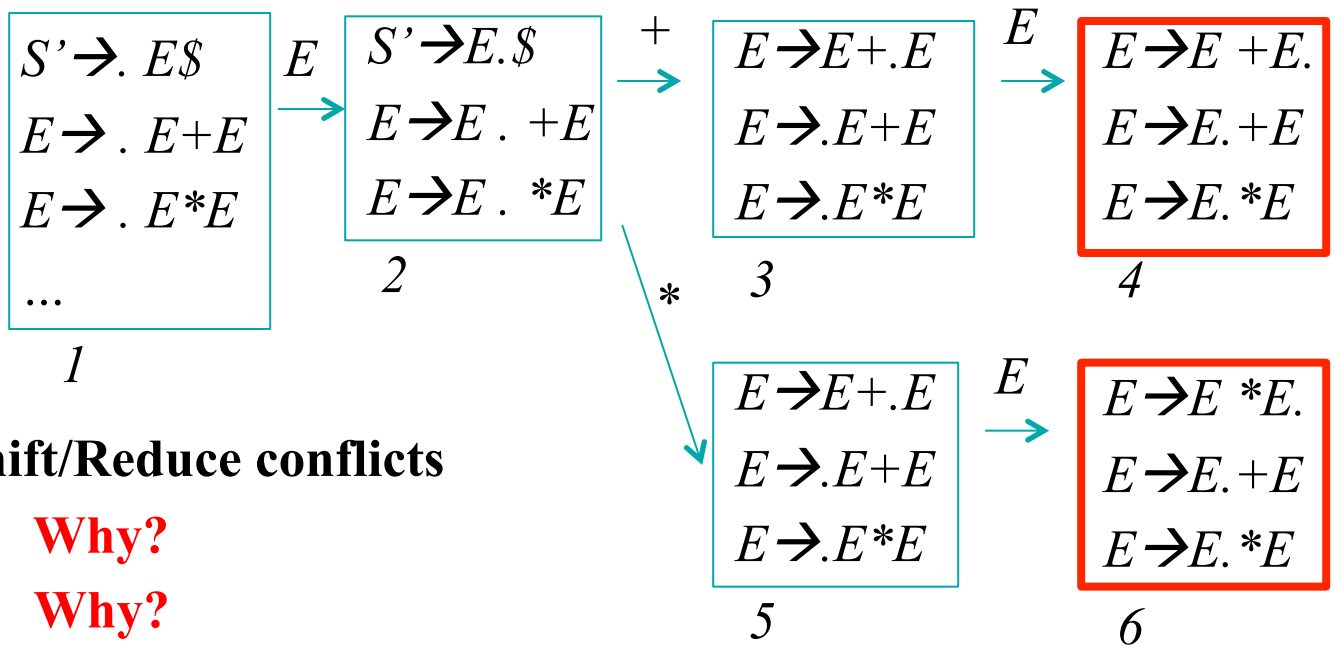
LALR(1) Look ahead LR(1) : state diagram (incomplete)

0: S' → S$ 1: S → V = E 2: S → E 3: E→V 4:V→x 5: V→*E



Shift reduce: ambiguous expression grammar (incomplete)

```
0: S'→ E$ 1: E→ E+E  2: E→ E*E  3: E→ (E)  4: E→ id
```



State 4: two Shift/Reduce conflicts

input +: R1 **Why?**

input *: S **Why?**

State 6: two Shift/Reduce conflicts

input +: R2 **Why?**

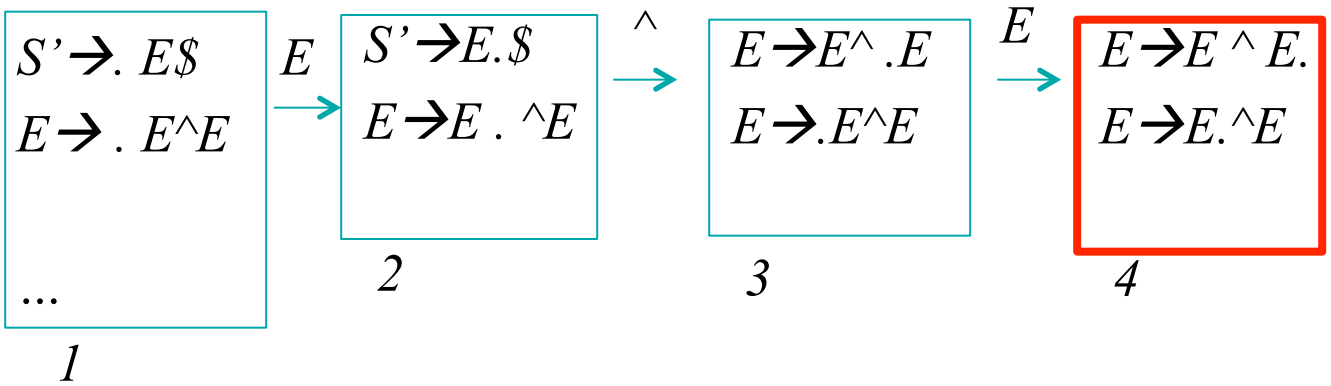
input *: R2 **Why?**

NOTICE:

+ and * are left associative

Shift reduce: ambiguous expression grammar (incomplete)

```
0: S'→ E$ 1: E→ E^E  2: E→id          // ^ for exponent
```



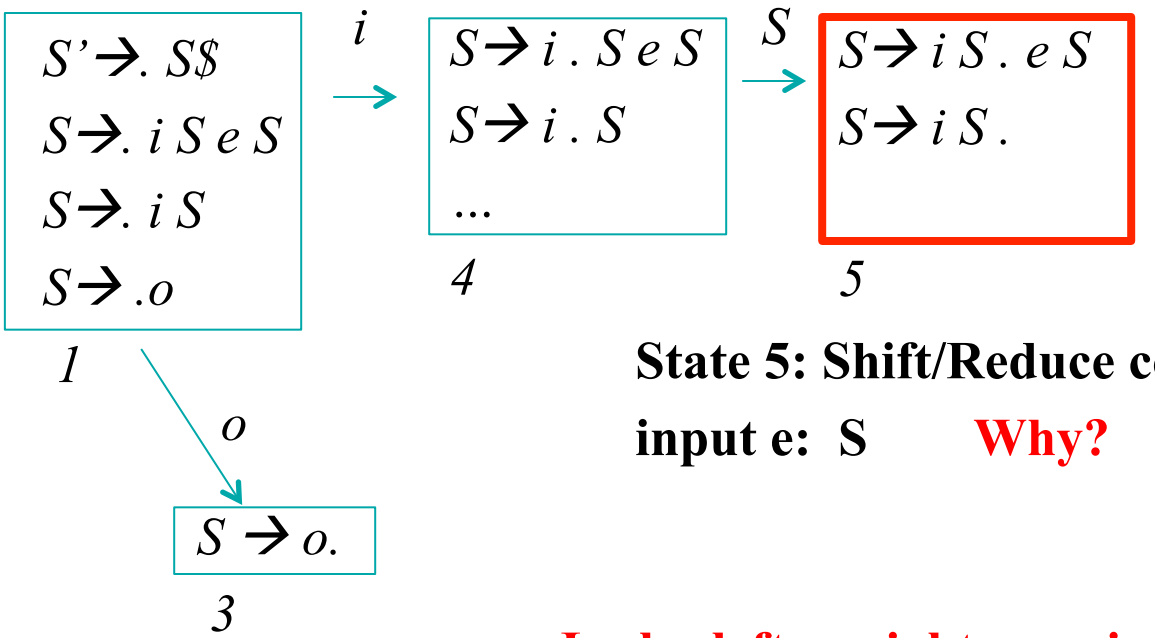
State 4: two Shift/Reduce conflicts

input ^ : S **Why?**

Shift reduce: ambiguous dangling else grammar

```
1: S → if c S else S   2: S → if c S     3: S → other

abstracted to:  1: S→ i S e S  2: i S   3: o
```



State 5: Shift/Reduce conflict

input e: S **Why?**

Is else left or right associative?

Shift reduce: ambiguous cast grammar

```
1: E→E+E  2: E → ( byte ) E  3: E → ( E ) 4: E → id
```

$E \rightarrow .\ E+E$
$E \rightarrow .\ (\ byte\ )\ E$
$E \rightarrow .\ (\ E\ )$
$E \rightarrow .\ id$

→ … →

$E \rightarrow E\ .\ +\ E$
$E \rightarrow (\ byte\ )\ E\ .$

Shift/Reduce conflict

input +: R2 **Why?**

(type) is a right associative cast operator, but consists of three tokens. This is a place where the strict nesting of ( ) does not hold.

The conflict can be resolved by declaring ( ) a right associative unary operator with higher precedence than the other operators.

Ambiguity in JavaCUP

There is no conflict free LR parser for ambiguous grammars

- So we have to fix the problem in other ways

Expressions: operators and productions are given precedence

- Tokens explicitly in precedence declarations
- Productions implicitly based on last operator in the stack

When in shift-reduce conflict

- Shift if look ahead token has higher precedence than production on stack
- Reduce if operator in the stack has higher precedence than look ahead

Within same precedence level

- Left associativity results in a reduce
- Right associativity results in a shift

Dangling else

- Shifting the else will bind else to last unbound if (Java semantics)
- Not shifting can cause a valid program to be rejected