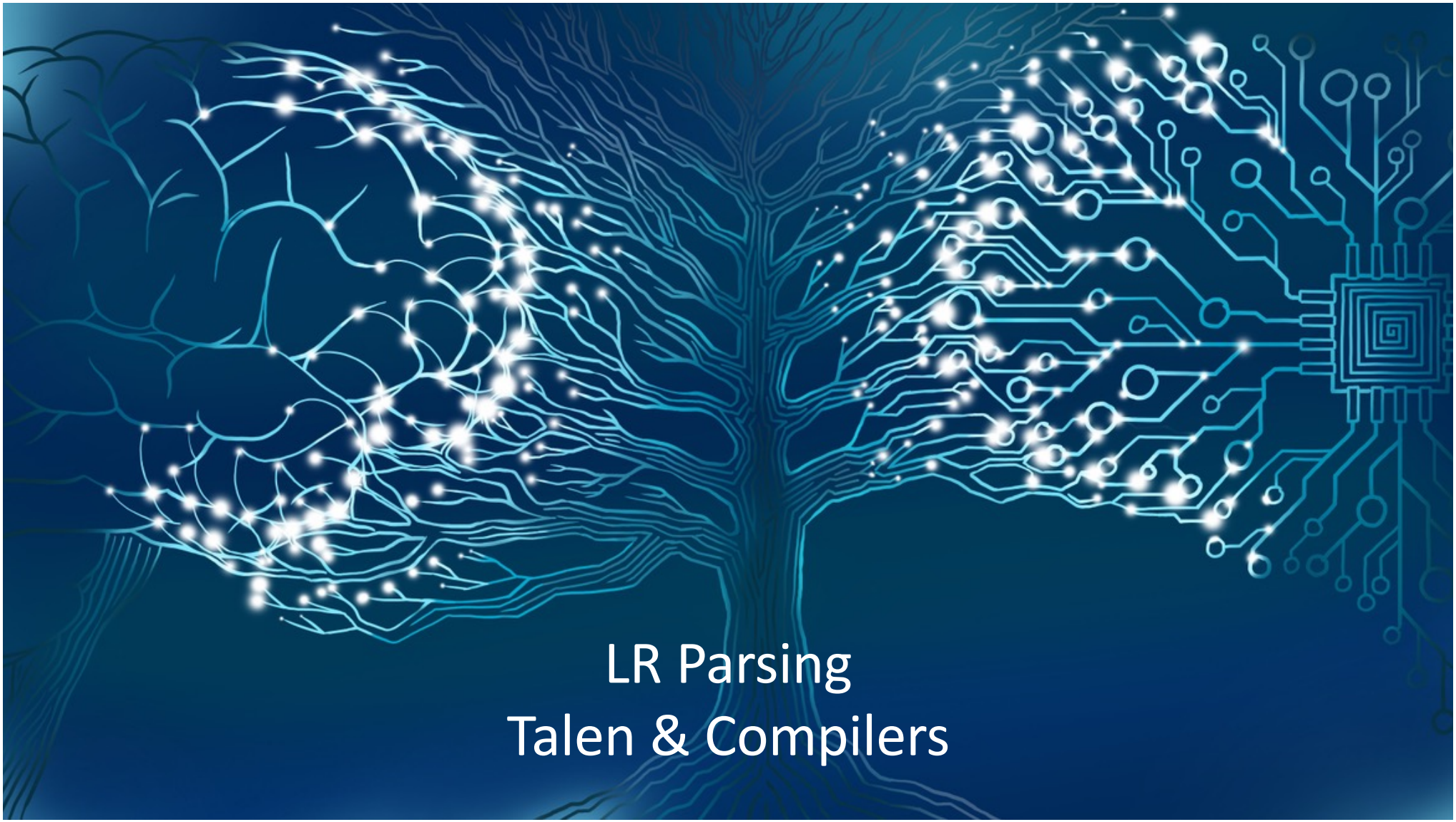

LR Parsing
Talen & Compilers

Utrecht University

The 2nd lab

Uses Happy, a parser that internally makes use of LR parsing; LALR(1) to be precise

Utrecht University

Today

LR parsing

Lecture notes: 9.2-3

Learning goals

Learning goals:

- Parse a sentence bottom-up using a stack and an LR(0) automaton
- Recognize and explain the different kinds of conflicts that can occur in LR parsing

Utrecht University

Parsing

Nondeterministic using parser combinators:

```
[s] → [(r,[s])]
```

Deterministic using a stack:

```
[s] →  Bool
```

together with a single derivation/ AST/ result

Utrecht University

Deterministic parsing

LL parsing:

- Left-to-right input processing
- Leftmost derivation
- Top-down

LR parsing:

- Left-to-right input processing
- Rightmost derivation
- Bottom-up

Utrecht University

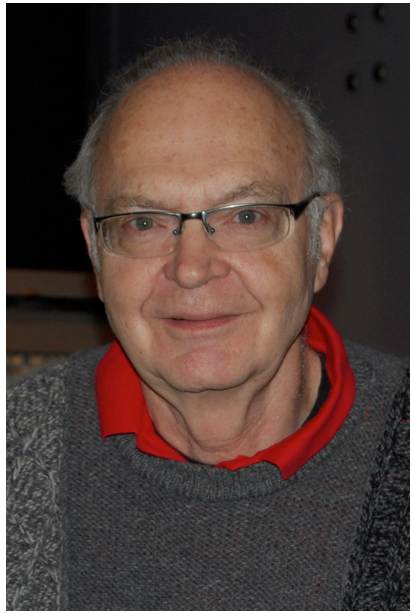
LR parsing idea

LR parsing state:

- A stack consisting of symbols
- The unprocessed part of the input

To parse a string x with a CFG with startsymbol S:

- Start with (ε,x)
- Move/replace symbols, ending with (S,ε)
- If impossible, parsing fails



Donald Knuth
LR: 1965

Utrecht University

An example

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
ε	aab	Shift

Utrecht University

An example

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
ε	aab	Shift
a	ab	Shift

Utrecht University

An example

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
ε	aab	Shift
a	ab	Shift
aa	b	Shift

Utrecht University

An example

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
ε	aab	Shift
a	ab	Shift
aa	b	Shift
aab	ε	Reduce

Utrecht University

An example

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
ε	aab	Shift
a	ab	Shift
aa	b	Shift
aab	ε	Reduce
aaS	ε	Reduce

Utrecht University

An example

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
ε	aab	Shift
a	ab	Shift
aa	b	Shift
aab	ε	Reduce
aaS	ε	Reduce
aS	ε	Reduce

Utrecht University

An example

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
ε	aab	Shift
a	ab	Shift
aa	b	Shift
aab	ε	Reduce
aaS	ε	Reduce
aS	ε	Reduce
S	ε	Parse successful

Utrecht University

Moving or replacing symbols

Shift:

- The first symbol in the input moves to the stack
- (v,ax) ⇒ (va,x)

Reduce:

- top symbols of the stack match a production right-hand side
- (vα,x) ⇒ (vN,x) if N → α

Utrecht University

Acceptance

An LR machine **accepts** a word if there exists a sequence of choices that leads to success.

- It is not always clear when to shift or reduce
- LR parsing is nondeterministic

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Try reduce

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Try reduce
B	ccba	Shift

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Try reduce
B	ccba	Shift
Bc	cba	Shift

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Try reduce
B	ccba	Shift
Bc	cba	Shift
Bcc	ba	Try reduce

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Try reduce
B	ccba	Shift
Bc	cba	Shift
Bcc	ba	Try reduce
BB	ba	Shift

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Try reduce
B	ccba	Shift
Bc	cba	Shift
Bcc	ba	Try reduce
BB	ba	Shift
BBb	a	Try reduce

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Try reduce
B	ccba	Shift
Bc	cba	Shift
Bcc	ba	Try reduce
BB	ba	Shift
BBb	a	Try reduce
BBS	a	Shift

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Try reduce
B	ccba	Shift
Bc	cba	Shift
Bcc	ba	Try reduce
BB	ba	Shift
BBb	a	Try reduce
BBS	a	Shift
BBSa	ε	Reduce

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Try reduce
B	ccba	Shift
Bc	cba	Shift
Bcc	ba	Try reduce
BB	ba	Shift
BBb	a	Try reduce
BBS	a	Shift
BBSa	ε	Reduce
BBSA	ε	Stuck

Utrecht University

Another strategy

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift

Utrecht University

Another strategy

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift

Utrecht University

Another strategy

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Shift

Utrecht University

Another strategy

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Shift
ccc	cba	Shift

Utrecht University

Another strategy

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Shift
ccc	cba	Shift
cccc	ba	Reduce

Utrecht University

Another strategy

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Shift
ccc	cba	Shift
cccc	ba	Reduce
ccB	ba	Shift

Utrecht University

Another strategy

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Shift
ccc	cba	Shift
cccc	ba	Reduce
ccB	ba	Shift
ccBb	a	Shift

Utrecht University

Another strategy

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Shift
ccc	cba	Shift
cccc	ba	Reduce
ccB	ba	Shift
ccBb	a	Shift
ccBba	ε	Reduce (how?)

Utrecht University

Another strategy

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Shift
ccc	cba	Shift
cccc	ba	Reduce
ccB	ba	Shift
ccBb	a	Shift
ccBba	ε	Reduce (how?)
ccBC	ε	Reduce

Utrecht University

Another strategy

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Shift
ccc	cba	Shift
cccc	ba	Reduce
ccB	ba	Shift
ccBb	a	Shift
ccBba	ε	Reduce (how?)
ccBC	ε	Reduce
cA	ε	Reduce

Utrecht University

Another strategy

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
ε	ccccba	Shift
c	cccba	Shift
cc	ccba	Shift
ccc	cba	Shift
cccc	ba	Reduce
ccB	ba	Shift
ccBb	a	Shift
ccBba	ε	Reduce (how?)
ccBC	ε	Reduce
cA	ε	Reduce
S	ε	Parse

Utrecht University

Derivation

```
S => cA => ccBC => ccBba => ccccba
```

```
S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba
```

This is a rightmost derivation

A leftmost derivation:

```
S => cA => ccBC => ccccC => ccccba
```

cannot be constructed with LR parsing (LL parsing!)

Select an action

During the parse process:

- we can only shift if there are symbols left in the input
- we should only shift if there is an option of a later reduction
- we can only reduce if the top of the stack matches a right-hand side

Keep track of where in the productions we might currently be while recognizing the input

Utrecht University

Items

An **item** (also called LR(0) item) is a production together with a marked position on the right-hand side. The position can be either at the beginning, between two symbols, or at the end

A → cBC

leads to the following items

A → ●cBC

A → c●BC

A → cB●C

A → cBC●

Utrecht University

Introducing end of file symbol

S’→ S$
S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Utrecht University

Constructing item sets

When we start parsing, we want to recognize the start symbol

```
S’→ ●S$
```

Which amounts to recognizing the right-hand side of S

```
S’→ S$
S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba
```

```
S’→ ●S$
S → ●cA
S → ●b
```

Keep on doing this until nothing can be done anymore: a fixed point, the **closure** of the item set.

Utrecht University

Constructing item sets

S’→ ●S$
S → ●cA
S → ●b

We cannot reduce in this state: only when ● appears at the end of a right-hand side we have recognized the necessary symbols for the non-terminal

We shift on seeing a c or a b

If the next symbol on the input is an a parsing fails

Utrecht University

Transitioning between item sets

S’→ ●S$
S → ●cA
S → ●b

We shift on seeing a c

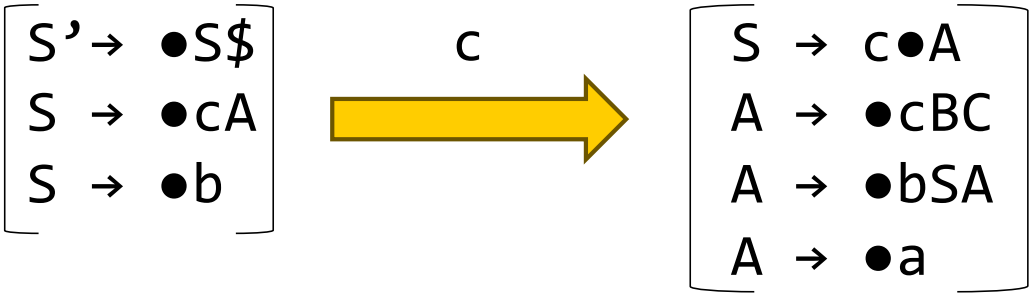
S → c●A
A → ●cBC
A → ●bSA
A → ●a

This is another shift state

S’→ S$
S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Utrecht University

Transitions

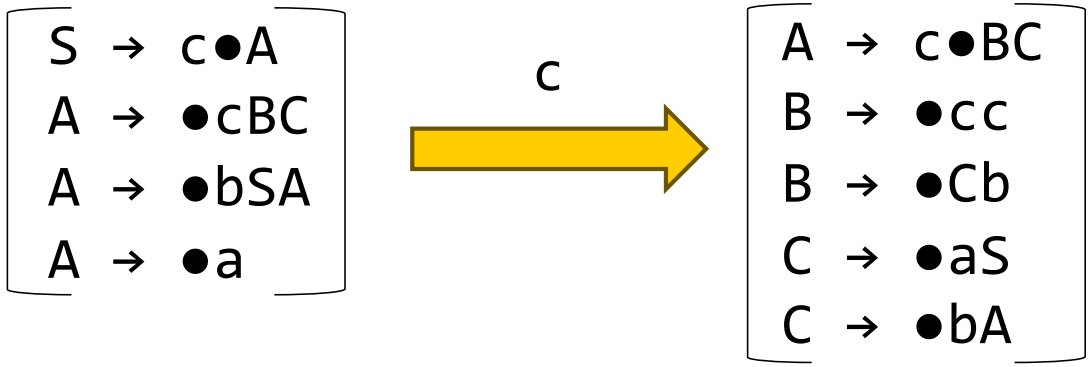


Transitions are labelled with a symbol: a terminal or a non-terminal

In a transition we move the position marker, and compute the new closure

Utrecht University

Transitions continued

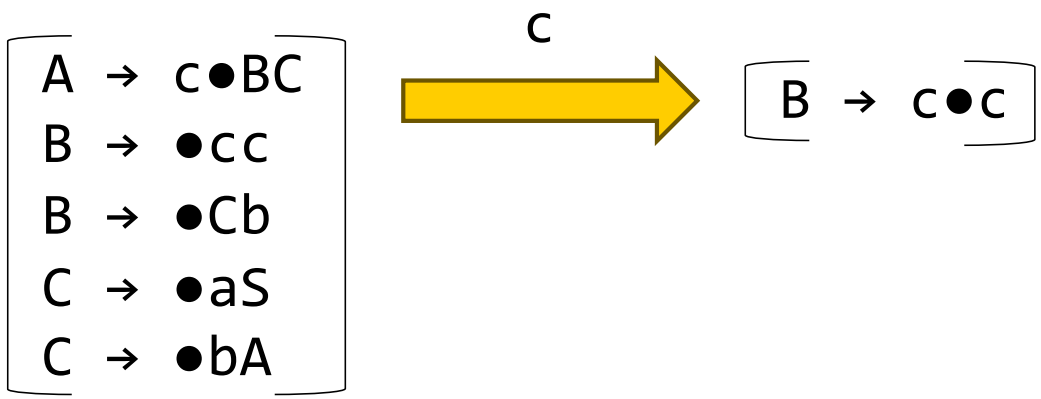



```
S'→ S$
S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba
```

This is another shift state

Utrecht University

Transitions continued

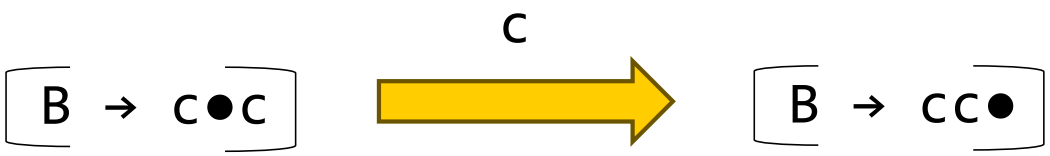


S'→ S$
S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

This is another shift state

Utrecht University

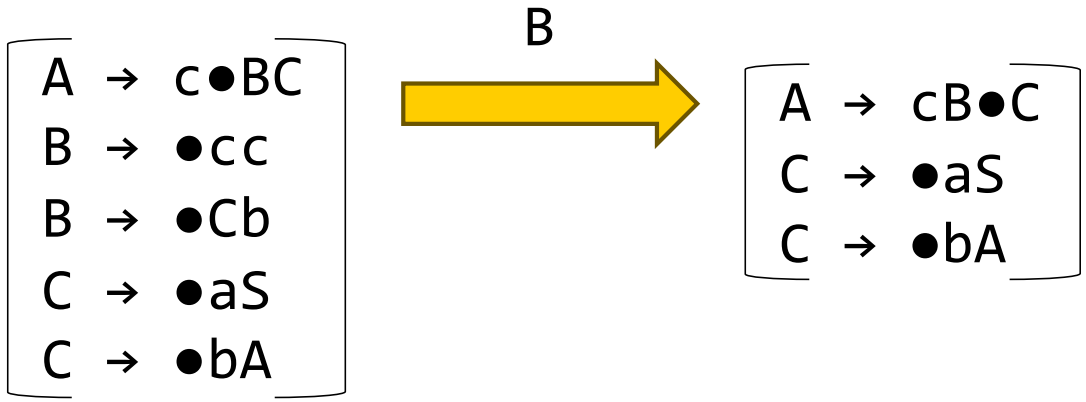
Transitions continued



This is a reduce state

Utrecht University

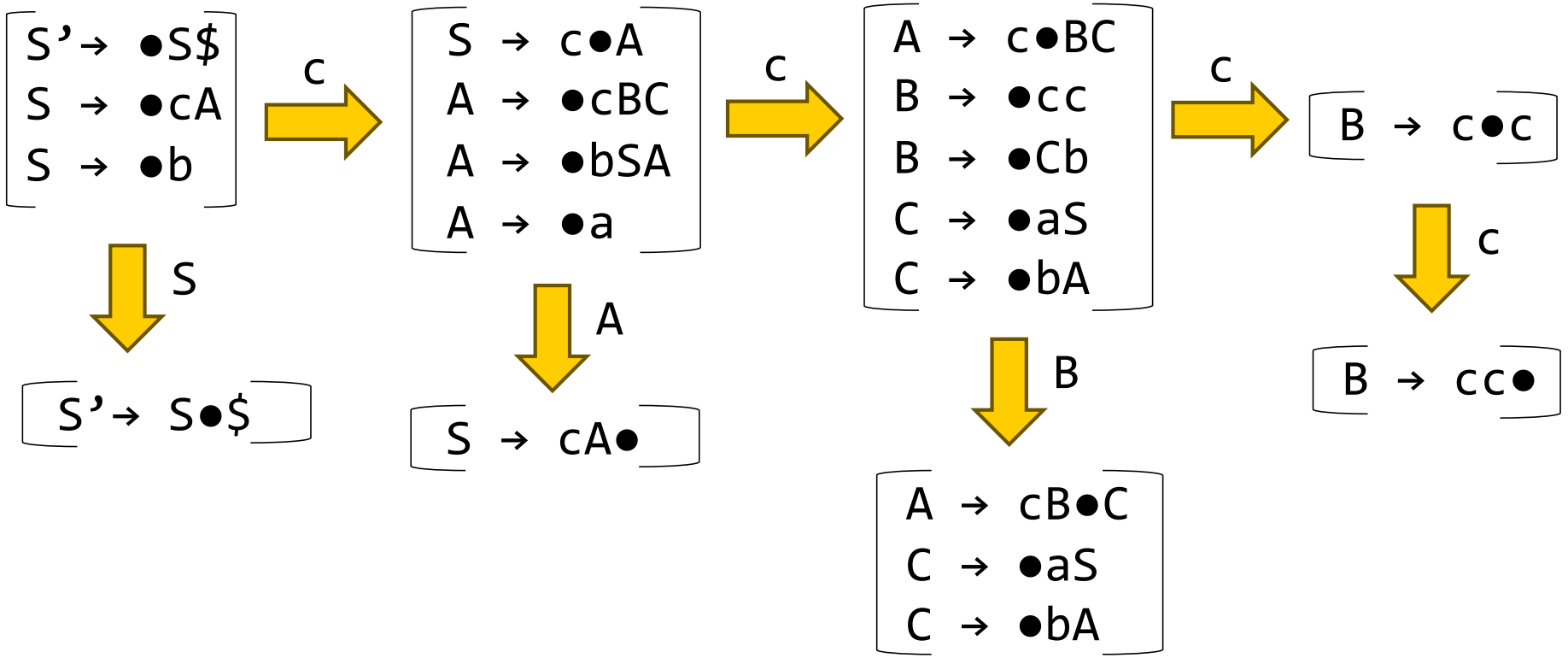
What happens if we reduce?




```
S'→ S$
S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba
```

Utrecht University

Part of an automaton



Utrecht University

The complete automaton

Such an automaton (an **LR(0) automaton**) can become rather large

In the example it has 20 states

Suitable for a generator, not really for manual construction

Utrecht University

1 Go to wooclap.com

2 Enter the event code in the top banner

Event code
FSIJHC

Enable answers by SMS

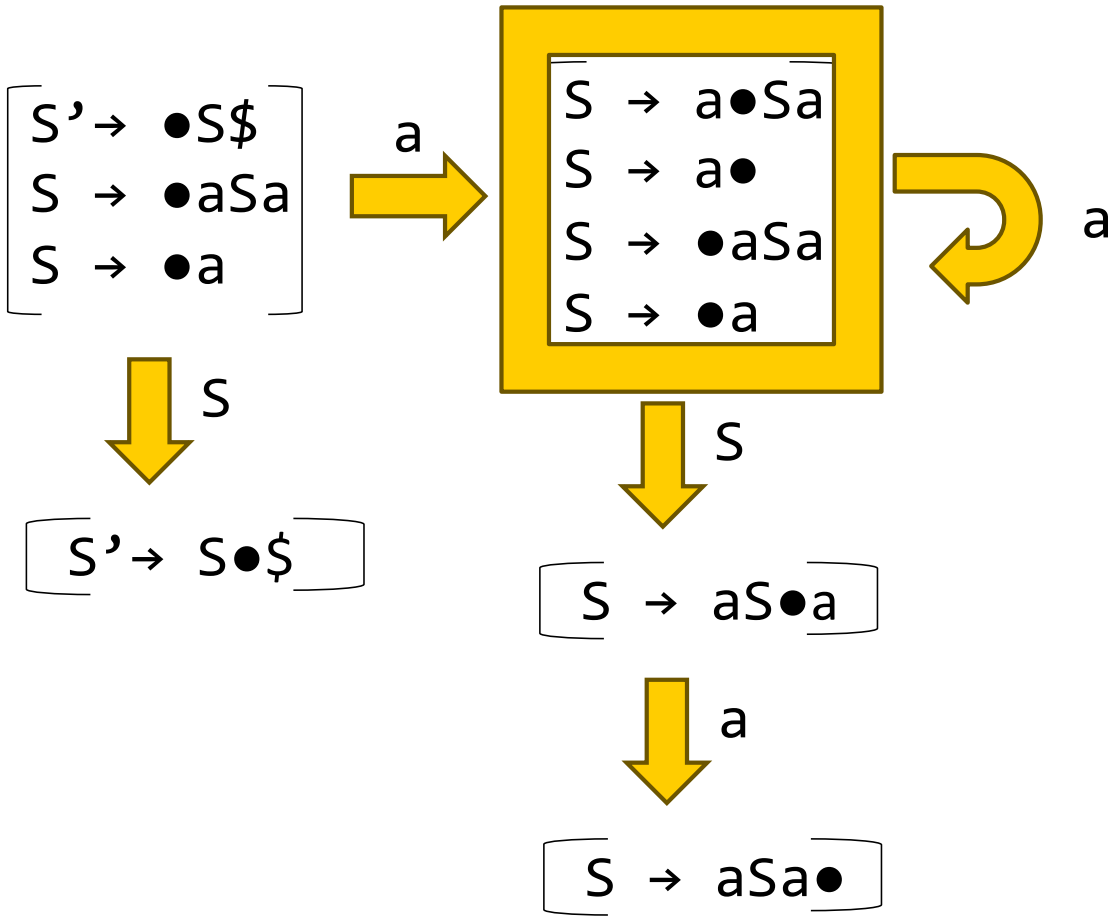
Conflicts

If a state allows both shifting and reducing, there is a **shift-reduce conflict**

If a state allows reducing by more than one production, there is a **reduce-reduce conflict**

Utrecht University

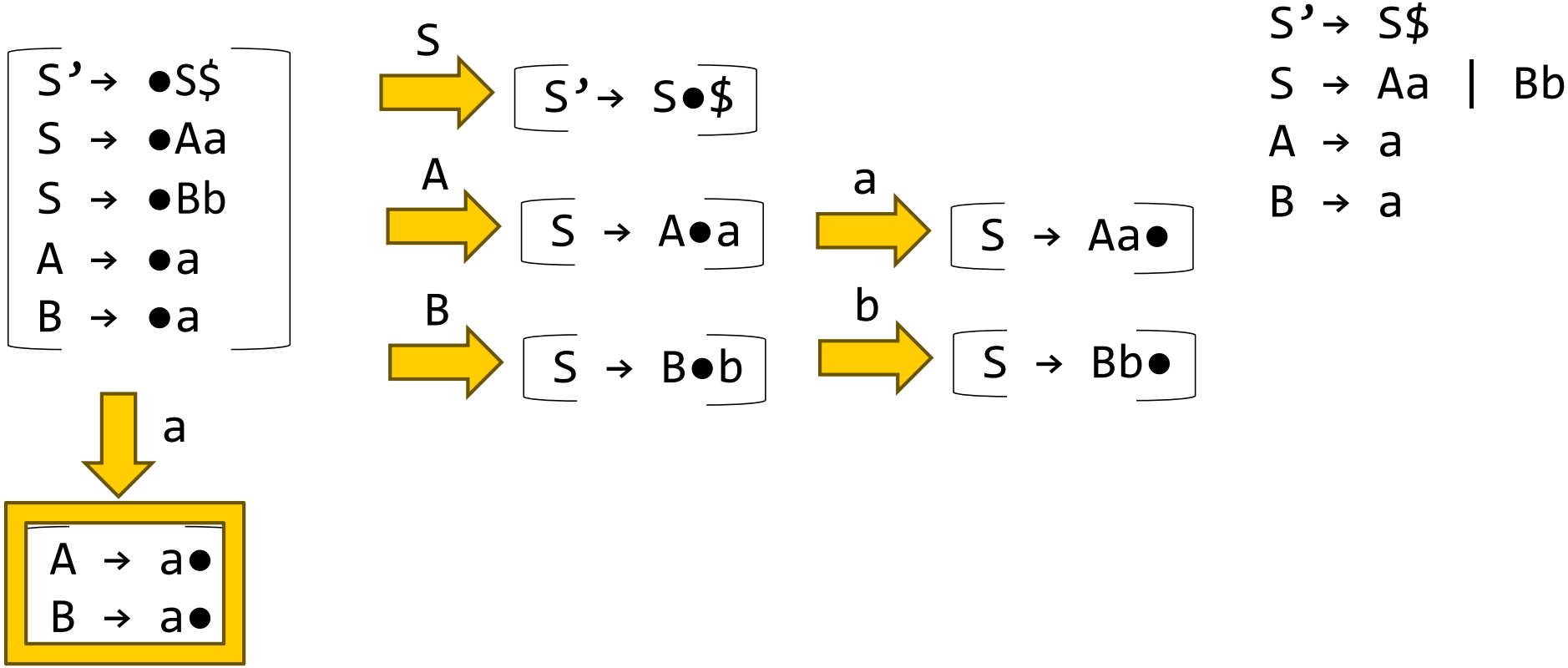
Shift-reduce conflict example



S’→ S$
S → aSa | a

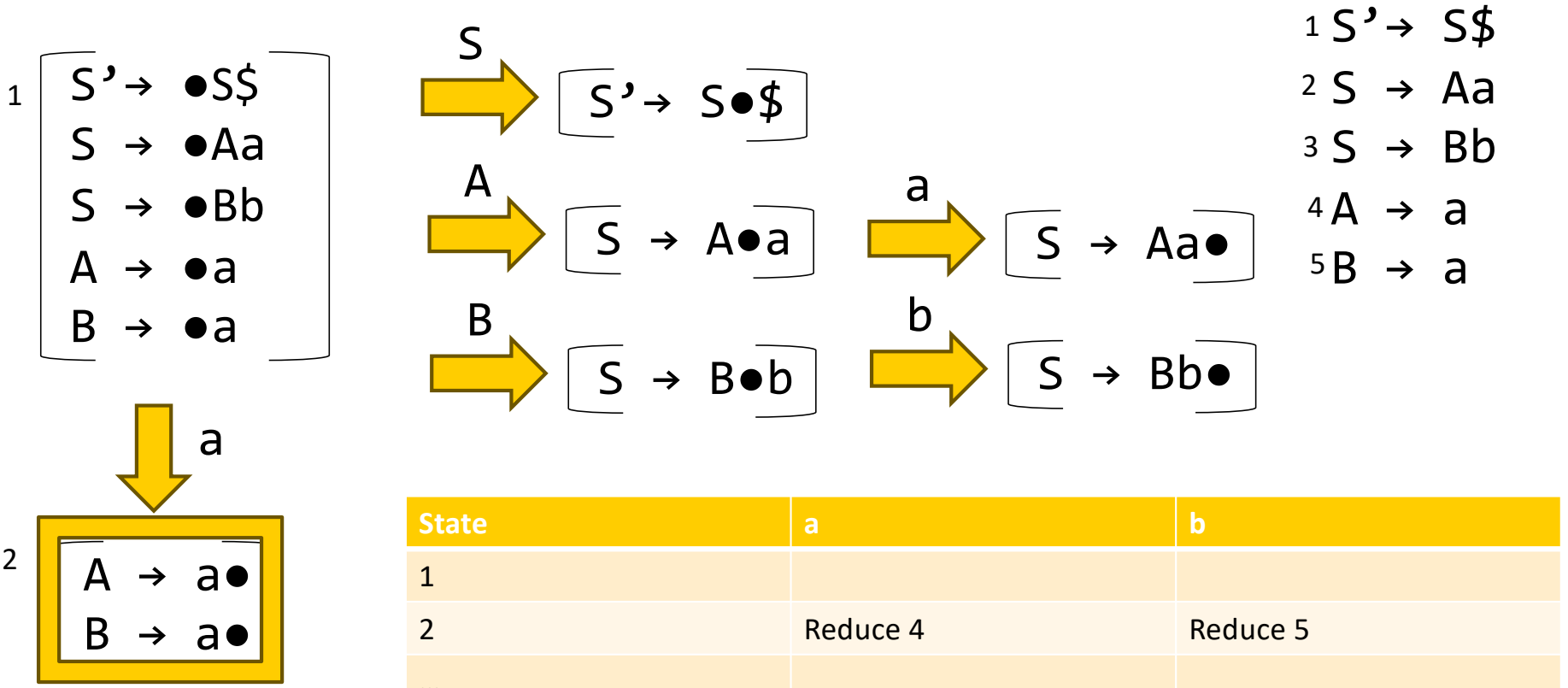
Utrecht University

Reduce-reduce conflict example



Utrecht University

Solving the conflict



State	a	b
1		
2	Reduce 4	Reduce 5
…		

Q2

Utrecht University

What to do with conflicts?

Solving the reduce-reduce conflict example is easy using the lookahead symbols of a non-terminal: SLR(1)

Solving the shift-reduce conflict example is difficult: need infinite lookahead

Had it been `S → ( S )` instead it would have been easy again

Zoo of approaches: SLR(1), LR(1), LR(k), LALR(1), GLR….

Differences in size of automata, tables, parallel processes, lookahead per non-terminal or item, …

Utrecht University

Summary

Bottom-up parsing using LR uses a stack, an automaton, and (often) a shift-reduce table

Probably the most used approach (at least in history) in parser generators