Sequence 2.5 – Simple LR Parser

P. de Oliveira Castro S. Tardieu

SLR Parser

- **S**imple
- **L**eft-to-right: tokens are read from left to right
- **R**ightmost derivation: reductions are always applied from the right

Building an SLR Parser

Grammar

$$(1) S \to T\#$$

$$(2) T \to aTbT$$

$$(3) T \to U$$

$$(4) U \to a$$

- S is the start rule, $\#$ is the EOF marker
- Terminals are $\{a, b, \#\}$

Start State

- Start by adding the start rule
- The . (dot) marks an imaginary cursor in the token flow
  - since we just started parsing; we are at the very start of the rule
  - we expect a T non-terminal

$$\boxed{S \rightarrow .T\#}$$

Figure 1: Start State

Transitive Closure

- We expect a T non-terminal
  - therefore, we include all the rules that produce a T
  - we add the . at the start of each production rule

$$\boxed{\begin{array}{c} S \rightarrow .T\# \\ T \rightarrow .aTbT \\ T \rightarrow .U \end{array}}$$

Figure 2: Start State

Transitive Closure

- Now the . is also before a U non-terminal
  - therefore, we include all the rules that produce a U

$$\boxed{\begin{array}{c} S \rightarrow .T\# \\ T \rightarrow .aTbT \\ T \rightarrow .U \\ U \rightarrow .a \end{array}}$$

Figure 3: Start State (after closure)

Adding Terminal Transitions

- For every **terminal** that follows the . we add a transition
  - Terminals that do not follow the . will not produce a valid derivation
  - The new state includes every rule that expects an *a* after the .
  - In the new state, the . moves after the consumed *a* token

$$\boxed{\begin{array}{c}(0)\\ S \to .T\# \\ T \to .aTbT \\ T \to .U \\ U \to .a\end{array}} \xrightarrow{\text{a}} \boxed{\begin{array}{c}(1)\\ T \to a.TbT \\ U \to a.\end{array}}$$

Figure 4: Terminal transitions

Adding Non Terminal transitions

- For every **non terminal** that follows the . we add a transition

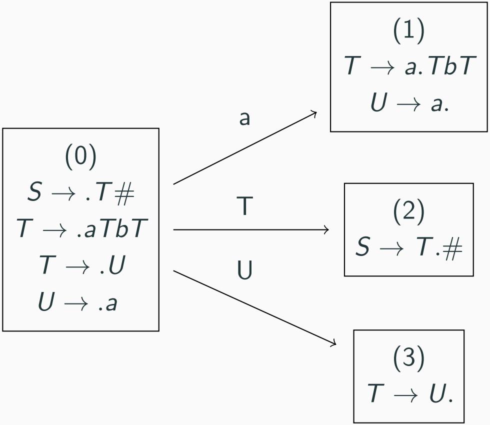

Figure 5: Non Terminal Transitions

Rules transitive Closure

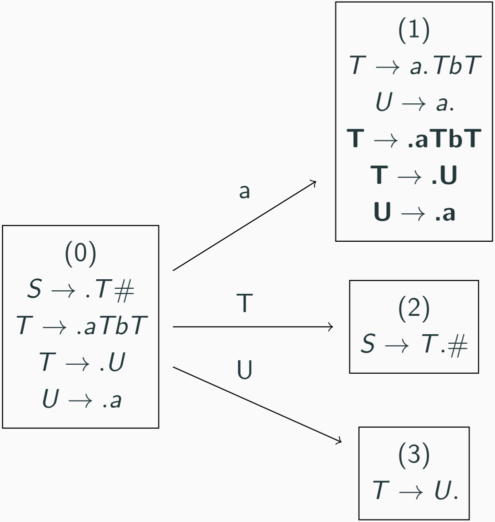

Figure 6: Add new rules transitively in state (1)

Reductions

- When the . is at the end, we add a *reduce* transition
- When we reach with rule (1) the # symbol, we have an *accept* state

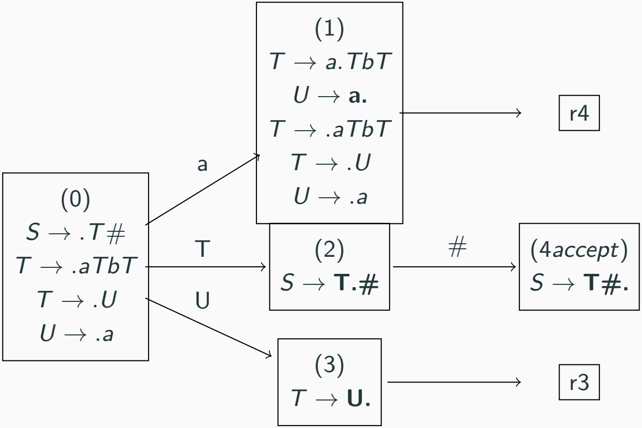

Figure 7: Reduce transitions

Adding transitions to state (1)

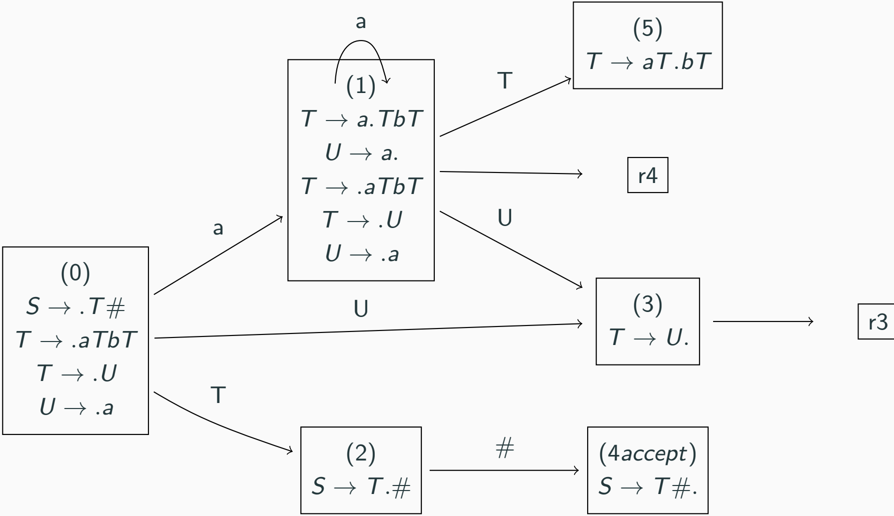

Figure 8: Add transitions to state (1)

Adding transitions to state (6)

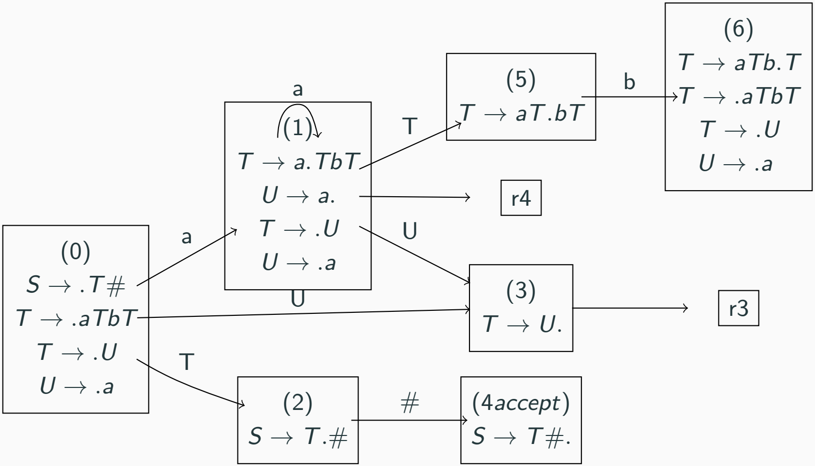

Figure 9: Add transitions to state (6)

Adding transitions to state (7)

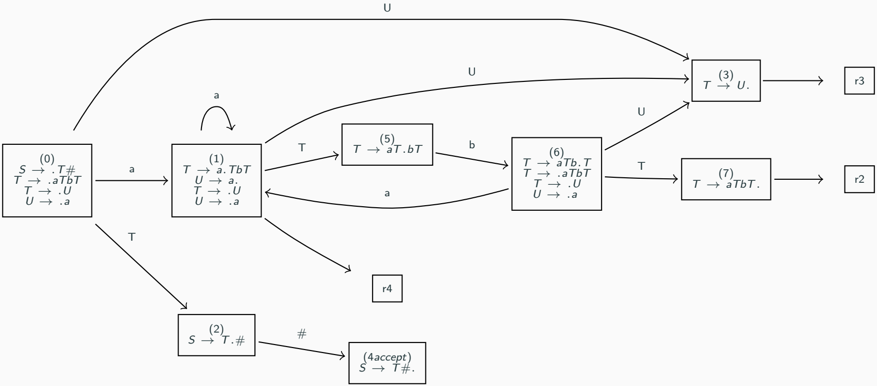

Figure 10: Adding transitions to state (7)

Building Follow Sets

$$(1) S \to T\# \quad (2) T \to aTbT$$
$$(3) T \to U \quad (4) U \to a$$

- The Follow set is the set of terminals that may follow a non-terminal

$$Follow(T) = \{b, \#\}$$
$$Follow(S) = \{\}$$

- Because U is at the end of rule (3), everything that follows T may follow U

$$Follow(U) = Follow(T)$$

Building the parsing table

- Encodes the automaton in table format
- non-terminal transitions are *shifts*
- *reductions* are only affected to the Follow set of the produced terminal

State	a	b	#	S	T	U
0	s1				2	3
1	s1	r4	r4		5	3
2			s4			
3		r3	r3			
4 (accept)						
5		s6				
6	s1				7	3
7		r2	r2			

Shift/Reduce or Reduce/Reduce Conflicts

- A conflict happens when two actions are possible for the same terminal
- By default, bison uses an LALR parser which is an extension of SLR
  - To debug shift/reduce or reduce/reduce conflicts bison outputs the parser automaton to a text file.
  - During the lab look at `src/parser/bison-report.txt`

Example of parsing (aaababa#)

Stack	Input	Action
0	aaababa#	shift 1
0,a,1	aababa#	shift 1
0,a,1,a,1	ababa#	shift 1
0,a,1,a,1,a,1	baba#	shift 1
0,a,1,a,1,a,1	baba#	reduce 4 (pop twice the RHS length)
		(4) U $\rightarrow$ a (here pop 2*1 elements)
		and follow U transition from state 1 $\rightarrow$ 3
0,a,1,a,1,U,3	baba#	reduce 3
		(3) T $\rightarrow$ U (here pop 2*1 elements)
		and follow T transition from state 1 $\rightarrow$ 5
0,a,1,a,1,T,5	baba#	shift 6
0,a,1,a,1,T,5,b,6	aba#	shift 1

Example of parsing (aaabba)

Stack	Input	Action
0,a,1,a,1,T,5,b,6,a,1	ba#	reduce 4
0,a,1,a,1,T,5,b,6,U,3	ba#	reduce 4
0,a,1,a,1,T,5,b,6,T,7	ba#	reduce 3
0,a,1,a,1,T,5,b,6,T,7	ba#	reduce 2
		(2) T $\rightarrow$ aTbT (pop 2*4 elements)
0,a,1,T,5	ba#	shift 6
0,a,1,T,5,b,6	a#	shift 1
0,a,1,T,5,b,6,a,1	#	reduce 4
0,a,1,T,5,b,6,U,3	#	reduce 3
0,a,1,T,5,b,6,T,7	#	reduce 2
0,T,2	#	shift 4
0,T,2,#		accept

Produced Derivation Tree