

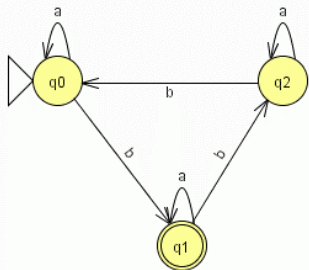
# A Hands-on Approach to FLA with JFLAP

## SLR(1) Parsing

Susan Rodger, Duke University

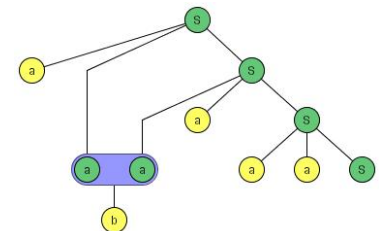
Thomas Finley, Cornell University

Peter Linz, University of California, Davis



SIGCSE 2006

March 4, 2006



# Parsing in JFLAP

- Brute Force Parsing
  - Reg. Grammars, CFG, unrestricted grammars
- LL(1) Parsing
- SLR(1) Parsing
  - Application with
    - DFA
    - Pushdown Automata
  - Can parse grammars with conflicts!

# Example Parsing with SLR

- Ambiguous Grammar
- Will have conflicts in the parse table, but can still parse strings

S	→	SS
S	→	a
S	→	b

# SLR(1) Parsing

1. Define FIRST and Follow sets

2. Build DFA

3. Define parse table

orange is conflict

JFLAP : <untitled7>

File Input Convert Help

Editor Build SLR(1) Parse

Do Selected Do Step Do All Next Parse

Parse table complete. Press "parse" to use it.

	FIRST	FOLLOW
S	{ a, b }	{ \$, a, b }

```

    graph LR
      q0((q0)) -- a --> q2((q2))
      q0 -- b --> q3((q3))
      q0 -- S --> q1((q1))
      q1 -- a --> q2
      q1 -- b --> q3
      q1 -- S --> q4((q4))
      q2 -- a --> q2
      q3 -- b --> q3
      q4 -- a --> q2
      q4 -- b --> q3
      q4 -- S --> q4
  
```

	a	b	\$	S
0	s2	s3		1
1	s2	s3	acc	4
2	r2	r2	r2	
3	r3	r3	r3	
4	r1	r1	r1	4

# Parse of aaba with reduce conflicts

- Parse entry highlighted

- Stack

- Rule used

- Parse tree

The screenshot shows the JFLAP interface for parsing the string 'aaba'. The 'SLR(1) Parsing' tab is active, displaying a table of parse entries, a stack, a rule list, and a parse tree.

	a	b	\$	S
0	s2	s3		1
1	s2	s3	acc	4
2	r2	r2	r2	
3	r3	r3	r3	
4	r1	r1	r1	4

Input: aaba  
Input Remaining: a\$  
Stack: 4S1S0

S'	→	S
S	→	SS
S	→	a
S	→	b

Parse tree diagram showing the derivation of 'aaba' from the start symbol S. The root node S has two children: S and S. The left S has two children: a and a. The right S has one child: b.

Reducing by S→b, S pushed on stack

# Parse of aaba complete

JFLAP : (slr-ambiguous.jff)

File Input Convert Help

Editor Build SLR(1) Parse SLR(1) Parsing

	a	b	\$	S
0	s2	s3		1
1	s2	s3	acc	4
2	r2	r2	r2	
3	r3	r3	r3	
4	r1	r1	r1	4

Start Step Noninverted Tree

Input: aaba  
Input Remaining: \$  
Stack: S0

S'	→	S
S	→	SS
S	→	a
S	→	b

The parse tree for the string 'aaba' is shown. The root node is S. It has two children: S and S. The rightmost S child of the root has a single child 'a'. The left S child of the root has two children: S and S. The right S child of this node has a single child 'b'. The left S child of this node has two children: S and S. Both of these S children have a single child 'a'. The tree structure is: S → (S S) → (S S) a → (S S) b a → a a b a.

String accepted

# Recall the conflicts

- When click on orange entry, can choose a different entry to resolve conflict
- For both, let's choose the shift instead of the reduce

	a	b	
0	s2	s3	
1	s2	s3	acc
2	r2	r2	r2
3	r3	r3	r3
4	r1	r1	r1

The table above shows a conflict at row 4. A dropdown menu is open over the 'r1' entry in column 'b', showing two options: 'r1' and 's3'. The 'r1' option is currently selected.

# Parse of aaba with shift conflicts

- Note tree is a different shape

JFLAP : (slr-ambiguous.jff)

File Input Convert Help

Editor Build SLR(1) Parse SLR(1) Parsing

	a	b	\$	S
0	s2	s3		1
1	s2	s3	acc	4
2	r2	r2	r2	
3	r3	r3	r3	
4	s2	s3	r1	4

Start Step Noninverted Tree

Input aaba

Input Remaining \$

Stack S0

S'	→	S
S	→	SS
S	→	a
S	→	b

```
graph TD; S1((S)) --- S2((S)); S1 --- S3((S)); S2 --- A1((a)); S3 --- S4((S)); S3 --- S5((S)); S4 --- A2((a)); S5 --- S6((S)); S5 --- S7((S)); S6 --- B((b)); S7 --- A3((a));
```

String accepted



# Comparison Reduce vs Shift Conflicts

With Reduce Entrees

With Shift Entrees

JFLAP : (slr-ambiguous.jff)

File Input Convert Help

Editor Build SLR(1) Parse SLR(1) Parsing

	a	b	\$	S
0	s2	s3	1	S
1	s2	s3	acc	4
2	r2	r2	r2	
3	r3	r3	r3	
4	r1	r1	r1	4

Start Step Noninverted Tree

Input: aaba  
Input Remaining: \$  
Stack: S0

S'	→	S
S	→	SS
S	→	a
S	→	b

String accepted

JFLAP : (slr-ambiguous.jff)

File Input Convert Help

Editor Build SLR(1) Parse SLR(1) Parsing

	a	b	\$	S
0	s2	s3	1	S
1	s2	s3	acc	4
2	r2	r2	r2	
3	r3	r3	r3	
4	s2	s3	r1	4

Start Step Noninverted Tree

Input: aaba  
Input Remaining: \$  
Stack: S0

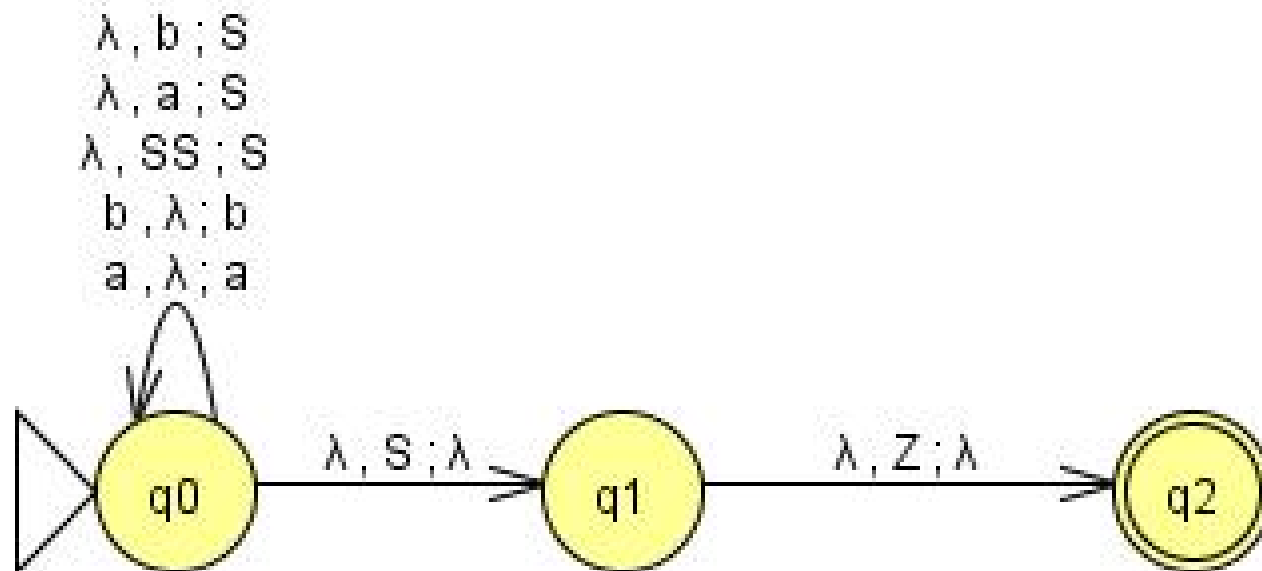
S'	→	S
S	→	SS
S	→	a
S	→	b

String accepted

# Compare SLR(1) with NPDA

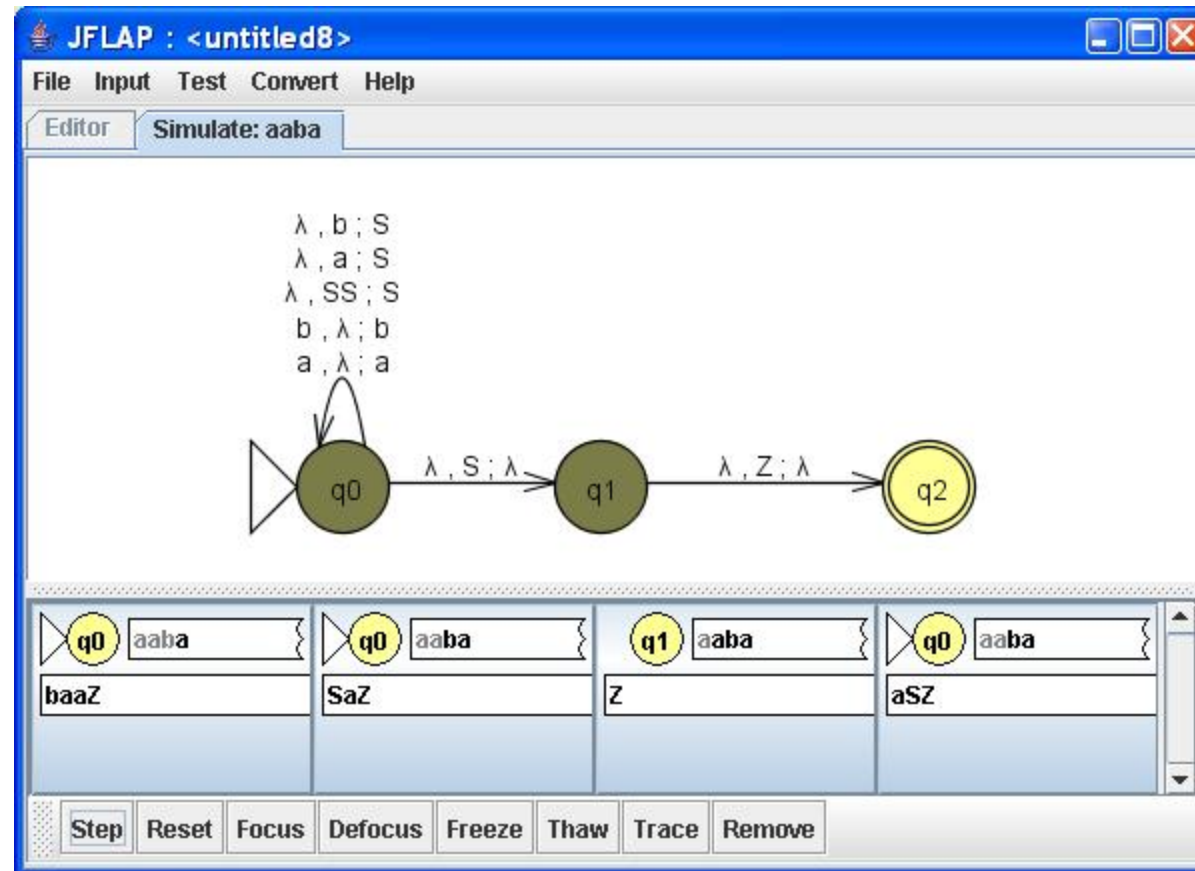
- Convert the CFG to an NPDA

S	→	SS
S	→	a
S	→	b



# Trace same string: aaba

- Note the nondeterminism
- Discuss how lookaheads in SLR(1) make it deterministic



# Finish the trace: aaba

- 5 paths accepted

The screenshot shows the JFLAP software interface with the following components:

- Title Bar:** JFLAP : <untitled8>
- Menu Bar:** File Input Test Convert Help
- Editor Tab:** Editor Simulate: aaba
- Automaton Definition:**

```
λ , b ; S
λ , a ; S
λ , SS ; S
b , λ ; b
a , λ ; a
```
- Automaton Diagram:** A finite automaton with three states: q0 (start state, yellow), q1 (yellow), and q2 (final state, grey). Transitions are: q0 to q0 on λ, q0 to q1 on λ, S; λ, q1 to q2 on λ, Z; λ.
- Simulation Area:** A table of simulation steps. The first row contains four steps, each with a yellow q2 icon and the string 'aaba'. The second row contains one step with a yellow q2 icon and the string 'aaba'. The remaining cells are empty.
- Control Panel:** Step, Reset, Focus, Defocus, Freeze, Thaw, Trace, Remove