Theoretical Computer Science Concepts Come Alive

Susan Rodger
Duke University

IIT Bombay
February 2, 2015
Outline

• Overview and Motivation
• Examples
• Design Goals of JFLAP
• Teaching with JFLAP
• Research Study
• History of JFLAP
Automata Theory

- Foundations of Computer Science (theory)
- Formal Languages
  - Finite state machines, Turing machines, grammars
  - Proofs
  - Motivation
    - How does a compiler work?
    - How does it recognize the syntax errors in your program?
Formal Languages and Automata Theory

• Traditionally taught
  – Pencil and paper exercises
  – No immediate feedback

  – More mathematical than most CS courses
  – Less hands-on than most CS courses
  – No programming? Unlike most other CS courses
Students Ready to learn Automata Theory!
Things start well enough …
But soon, instead of pictures, there are **WORDS**.
Big words! The type with more than one syllable!
VIOLENCE AMONG STUDENTS AS NERVES FRAY!
We only wanted to learn automata theory! Isn’t there a better way?
Try JFLAP …
Students Learning Automata with JFLAP
### Why Develop Tools for Automata?

| Textual  | \([\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\}]\) \\
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[\delta = {(q_0, b, q_0), (q_0, a, q_1), (q_1, a, q_0), (q_1, b, q_2), (q_2, a, q_1)}]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tabular</th>
<th>![Tabular Diagram]</th>
</tr>
</thead>
</table>
|           | \[\begin{array}{c|cc}
|          | a & b \\
| \hline
| q_0 & q_0 & q_0 \\
| q_1 & q_1 & q_2 \\
| q_2 &     &     \\
| \end{array}\] |

| Visual    | ![Visual Diagram] |

| Interactive| ![Interactive Diagram] |
Other Tools for Automata

• Turing’s World (Barwise and Etchemendy)
• Deus Ex Machina (Taylor and Savoiu)
• Theory of Computing Hypertextbook (Ross)
• Many others
  – L-System tools
  – Compiler tools
  – Finite State machine tools
Electronic Textbooks (ebooks) engage students

- OpenDSA (Shaffer, Virgina Tech)
  - Algorithm animations built in
- runestoneinteractive.org (Brad Miller,
  - Several books (Python)
    - Python - try and run code built in
    - Quizzes
- Zyante.com – interactive textbooks
- Track student progress
- Requirements and design strategies for open source interactive computer science eBooks
  - ITiCSE 2013 Working Group (Korhonen, Naps, et al)
Overview of JFLAP

• **Java Formal Languages and Automata Package**

• Instructional tool to learn concepts of Formal Languages and Automata Theory

• Topics:
  – Regular Languages
  – Context-Free Languages
  – Recursively Enumerable Languages
  – Lsystems
  – Many proofs (NFS to DFA to min DFA to RE)

• **With JFLAP your creations come to life!**
Thanks to Students - Worked on JFLAP and Automata Theory Tools

- NPDA - 1990, C++, Dan Caugherty
- JFLAP - 1996-1999, Java version
  Eric Gramond, Ted Hung, Magda and Octavian Procopiuc
- Pâté, JeLLRap, Lsys
  Anna Bilska, Jason Salemme, Lenore Ramm, Alex Karweit, Robyn Geer
- JFLAP 4.0 – 2003, Thomas Finley, Ryan Cavalcante
- JFLAP 6.0 – 2005-2008 Stephen Reading, Bart Bressler, Jinghui Lim, Chris Morgan, Jason Lee
- JFLAP 7.0 - 2009 Henry Qin, Jonathan Su
- JFLAP 8.0? – 2011-14 Julian Genkins, Ian McMahon, Peggy Li, Lawrence Lin, John Godbey

Over 20 years!
Outline

• Overview and Motivation
• Examples
• Design Goals of JFLAP
• Teaching with JFLAP
• Research Study
• History of JFLAP
DFA Example

• Build a deterministic finite automaton (DFA) to recognize even binary numbers with an even number of 1s.

• Only use symbols 0 and 1

• Binary numbers: 0, 1, 10, 11, 100, 101, 110, 111, …

• When is a binary number an even number?
  – Ends in 0

• Which strings should be accepted?

• 11010, 10010, 1111, 10100

  No, odd      Yes      No, ends      Yes
  no. of 1’s    In 1
Simulation on 1101010
Simulation on 1101010
Simulation on 1101010

- State q0 transitions to q1 on input 0.
- State q1 transitions to q0 on input 1.
- State q2 transitions to q3 on input 1.
- State q3 transitions to q4 on input 0.
- State q4 transitions to q2 on input 1.
Simulation on 1101010
Simulation on 1101010
Simulation on 1101010
Simulation on
1101010
Accepts Input!
1101010
Add meaning to states!
The diagram illustrates a finite automaton with four states labeled q0, q1, q2, and q4.

- **q0**: Starts here.
- **q1**: Represents a state with only one 0.
- **q2**: Represents a state where the input is 0, indicating an odd number of 1's.
- **q3**: Represents a state where the input is 1.
- **q4**: Represents a state where the input is 0, indicating an even number of 1's.

Transitions:
- From q0 to q1 on input 0.
- From q0 to q2 on input 1.
- From q2 to q3 on input 1.
- From q3 to q4 on input 0.
- From q4 to q2 on input 0.

The automaton accepts binary numbers with an even number of 1's.
The diagram shows a finite automaton with states labeled as follows:

- **q0**: Start state. Transition on '1' to q2.
- **q1**: Transition on '0' to q2. States with only one 0.
- **q2**: Transition on '0' to q1. States with an odd number of 1's.
- **q3**: Transition on '1' to q4. States with an even number of 1's, ends in 1.
- **q4**: Accepting state. Transition on '0' back to q3.
The diagram represents a finite automaton with states labeled q0, q1, q2, q3, and q4. The transitions are labeled with 0s and 1s:

- From q0:
  - Transition on 1 to q1 labeled "only one 0".
  - Transition on 0 to q2 labeled "odd number of 1's".

- From q1:
  - Transition on 0 to q0 labeled "only one 0".

- From q2:
  - Transition on 0 to q3 labeled "even number of 1's, ends in 1".
  - Transition on 1 to q4 labeled "even number of 1's, ends in 0".

- From q3:
  - Transition on 1 to q4 labeled "even number of 1's, ends in 0".
  - Transition on 0 to q2 labeled "odd number of 1's".

- From q4:
  - Transition on 0 to q3 labeled "even number of 1's, ends in 1".
Test Multiple Inputs
NFA to DFA

1. Start with NFA

2. Construct new DFA
   On q0 with an a, go to q0, q1 and q2

3. Final DFA
DFA to Min DFA

- Start with DFA
DFA to Min DFA(2)

- Start tree of distinguishable states

- Complete tree!
DFA to Min DFA (3)

• Determine states in min DFA

• Add arcs to complete it
Regular Expression (RE) to FA

1. Start with RE

2. Generalized Transition Graph (GTG)

3. De-oring
RE to FA (cont)

4. De-concatenate

5. De-staring
FA to RE

1. Start with FA
2. FA only one final state
3. Convert to GTG
4. Select node to remove, q1
FA to RE (cont)

5. Compute new RE transitions

6. Remove state q1

7. Resulting RE

$a^*(a+ab)(c+bb)^*$
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
SLR(1) Parsing

1. Define FIRST and Follow sets
2. Build DFA
3. Define parse table

orange is conflict
Parse of aaba with reduce conflicts

- Parse entry highlighted
- Stack
- Rule used
- Parse tree

Reducing by $S \rightarrow b$, S pushed on stack
Parse of aaba complete
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
Parse of aaba with shift conflicts

- Note tree is a different shape
Comparison Reduce vs Shift Conflicts

With Reduce Entrees

With Shift Entrees
Compare SLR(1) with NPDA

- Convert the CFG to an NPDA
Trace same string: aaba

- Note the nondeterminism
- Discuss how lookaheads in SLR(1) make it deterministic
Finish the trace: aaba

- 5 paths accepted
Another Example: Grammar

• Grammar – set of replacement rules to define a language

• Examples:
  – Grammar for English
    • defines English sentences
  – Grammar for Python programming language
    • defines syntactically correct programs
  – Grammar for a formal language (simpler)
Grammar for $a^n b^n c^n$

- Unrestricted grammar
- Generates strings with an equal number of a’s, b’s, c’s
- a’s first, then b’s, then c’s
- Example strings can derive:
  - abc
  - aabbccc
  - aaabbbcccc
  - aaaaabbbbbcccccc
  - …
Example Derivation for aabbcc

\[ S \rightarrow AX \]

rule: \[ S \rightarrow AX \]
Example Derivation for aabbcc

S → AX  
  → aAbcX  

rule: S → AX  
rule: A → aAbc
Example Derivation for aabbcc

S → AX          rule: S → AX
   → aAbcX       rule: A → aAbc
   → aaBbcBbcX    rule: A → aBbc

NOTE: We have generated the correct symbols, aabcbc, but they are in the wrong order!
Example Derivation for aabbcce

S → AX  
→ aAAbcX  rule: S -> AX
→ aaBbcbcX  rule: A -> aAbc
→ aabBcbbcX  rule: A -> aBbc
→ aabBcbbcX  rule: Bb -> bB
Example Derivation for aabbcc

S \rightarrow AX \quad \text{rule: } S \rightarrow AX

\rightarrow aAbcX \quad \text{rule: } A \rightarrow aAbc

\rightarrow aaBbcbX \quad \text{rule: } A \rightarrow aBbc

\rightarrow aabBbcX \quad \text{rule: } Bb \rightarrow bB

\rightarrow aabDBCX \quad \text{rule: } Bc \rightarrow D

Note: the D absorbed the c!
Example Derivation for aabbcc

S → AX  
→ aAbcX  
→ aaBbcXcX  
→ aabBbcXcX  
→ aabDbcX  
→ aabDbcX  

rule:  S -> AX  
rule:  A -> aAbc  
rule:  A -> aBbc  
rule:  Bb -> bB  
rule:  Bc -> D  
rule:  Db -> bD
Example Derivation for aabbcc

S → AX

→ aAAbcX  rule: S → AX

→ aaBbcbcX  rule: A → aAbc

→ aabBbcbcX  rule: A → aBbc

→ aabBbcDcX  rule: Bbc → bB

→ aabbDcX  rule: Bc → D

→ aabbDcDX  rule: Db → bD

→ aabbcDX  rule: Dc → cD
Example Derivation for aabbcc

\[
\begin{align*}
S & \rightarrow AX \\
 & \rightarrow aAbcX \\
 & \rightarrow aaBbcbcX \\
 & \rightarrow aabBbcX \\
 & \rightarrow aabDBCX \\
 & \rightarrow aabbDcX \\
 & \rightarrow aabbcDX \\
 & \rightarrow aabbcEXc \\
\end{align*}
\]

rule: \( S \rightarrow AX \)
rule: \( A \rightarrow aAbc \)
rule: \( A \rightarrow aBbc \)
rule: \( Bb \rightarrow bB \)
rule: \( Bc \rightarrow D \)
rule: \( Db \rightarrow bD \)
rule: \( Dc \rightarrow cD \)
rule: \( Dx \rightarrow Exc \)

Eventually \( \ldots \rightarrow \) aabbcc

Note the c spit out on right end!
We could have done this derivation of aabbcc with JFLAP.

Now let’s see how JFLAP visualizes this derivation with a “parse tree”
Input: aabbc
String accepted! 51 nodes generated.

<table>
<thead>
<tr>
<th>LHS</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>AX</td>
</tr>
<tr>
<td>A</td>
<td>aAbc</td>
</tr>
<tr>
<td>A</td>
<td>aBbc</td>
</tr>
<tr>
<td>Bb</td>
<td>bB</td>
</tr>
<tr>
<td>Bc</td>
<td>D</td>
</tr>
<tr>
<td>Dc</td>
<td>cD</td>
</tr>
<tr>
<td>Db</td>
<td>bD</td>
</tr>
<tr>
<td>DX</td>
<td>EXc</td>
</tr>
<tr>
<td>BX</td>
<td>λ</td>
</tr>
<tr>
<td>cE</td>
<td>Ec</td>
</tr>
<tr>
<td>bE</td>
<td>Eb</td>
</tr>
<tr>
<td>aE</td>
<td>aB</td>
</tr>
</tbody>
</table>

Press step to show derivations.
Input: aabbcc
String accepted! 51 nodes generated.

<table>
<thead>
<tr>
<th>LHS</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>AX</td>
</tr>
<tr>
<td>A</td>
<td>aAbc</td>
</tr>
<tr>
<td>A</td>
<td>aBbc</td>
</tr>
<tr>
<td>Bb</td>
<td>bB</td>
</tr>
<tr>
<td>Bc</td>
<td>D</td>
</tr>
<tr>
<td>Dc</td>
<td>cD</td>
</tr>
<tr>
<td>Db</td>
<td>bD</td>
</tr>
<tr>
<td>DX</td>
<td>EXc</td>
</tr>
<tr>
<td>BX</td>
<td>λ</td>
</tr>
<tr>
<td>cE</td>
<td>Ec</td>
</tr>
<tr>
<td>bE</td>
<td>Eb</td>
</tr>
<tr>
<td>aE</td>
<td>aB</td>
</tr>
</tbody>
</table>

Derived AX from S.
Input: aabbcc
String accepted! 51 nodes generated.

LHS | RHS
---|---
S  | AX
A  | aAbc
A  | aBbc
Bb | bB
Bc | D
Dc | cD
Db | bD
DX | EXc
BX | λ
cE | Ec
bE | Eb
aE | aB

Derived aAbc from A.
Note all letters there, but wrong order: aabcbcbc
Input: aabbcc
String accepted! 51 nodes generated.
Absorb the “c”
Input: aabbcc
String accepted! 51 nodes generated.

Derived bD from Db.
Spit out the “c” at the right end.
Input: aabbcc
String accepted! 51 nodes generated.

LHS | RHS
---|---
S  | AX
A  | aAbCc
A  | aBbCc
Bb | bB
Bc | D
Dc | cD
Db | bD
DX | EXc
BX | λ
cE | Ec
bE | Eb
aE | aB

derived Ec from cE.
Input: aabbcc
String accepted! 51 nodes generated.

LHS | RHS
---|---
S  | AX
A  | aAbc
A  | aBbc
Bb | bB
Bc | D
Dc | cD
Db | bD
DX | EXc
BX | λ
cE | Ec
bE | Eb
aE | aB

Derived Eb from bE.
String accepted! 51 nodes generated.

Derived Eb from bE.
Input: aabbcc
String accepted! 51 nodes generated.

LHS | RHS
---|---
S  → AX
A  → aAbc
A  → aBbc
Bb → bB
Bc → D
Dc → cD
Db → bD
DX → EXc
BX → λ
cE → Ec
bE → Eb
aE → aB

Derived aB from aE.
Input: aabbcc
String accepted! 51 nodes generated.

Derived bB from Bb.
Input: aabbcc
String accepted! 51 nodes generated.

LHS | RHS
---|---
S  | AX
A  | aAbc
A  | aBbc
Bb | bB
Bc | D
Dc | cD
Db | bD
DX | EXc
BX | λ
cE | Ec
bE | Eb
aE | aB

Derived bB from Bb.
Absorb second “c”
Spit the “c” out at right end
Input: aabbcc
String accepted! 51 nodes generated.

LHS | RHS
---|---
S → AX
A → aAbc
A → aBbc
Bb → bB
Bc → D
Dc → cD
Db → bD
DX → EXc
BX → λ
cE → Ec
bE → Eb
**aE → aB**

Derived Eb from bE.
Input: abbc
String accepted! 51 nodes generated.

Derived Eb from bE.
Input: aabbcc
String accepted! 51 nodes generated.

LHS | RHS
---|---
S  | AX
A  | aAbc
A  | aBbc
Bb | bB
Bc | D
Dc | cD
Db | bD
DX | EXc
BX | λ
cE | Ec
bE | Eb
aE | aB

Derived aB from aE.
Input: aabbcc
String accepted! 51 nodes generated.

LHS | RHS
---|---
S  | AX
A  | aAbc
A  | aBbc
Bb | bB
Bc | D
Dc | cD
Db | bD
DX | EXc
BX | λ
cE | Ec
bE | Eb
aE | aB

Derived bB from Bb.
Input: aabbcc
String accepted! 51 nodes generated.

<table>
<thead>
<tr>
<th>LHS</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>AX</td>
</tr>
<tr>
<td>A</td>
<td>aAbc</td>
</tr>
<tr>
<td>A</td>
<td>aBbc</td>
</tr>
<tr>
<td>Bb</td>
<td>bB</td>
</tr>
<tr>
<td>Bc</td>
<td>D</td>
</tr>
<tr>
<td>Dc</td>
<td>cD</td>
</tr>
<tr>
<td>Db</td>
<td>bD</td>
</tr>
<tr>
<td>DX</td>
<td>EXc</td>
</tr>
<tr>
<td>BX</td>
<td>λ</td>
</tr>
<tr>
<td>cE</td>
<td>Ec</td>
</tr>
<tr>
<td>bE</td>
<td>Eb</td>
</tr>
<tr>
<td>aE</td>
<td>aB</td>
</tr>
</tbody>
</table>

Derived bB from Bb.
Input: aabbcc
String accepted! 51 nodes generated.

Derived A from BX. Derivations complete.
What else can JFLAP do?

• Create other machines
  – Moore and Mealy
  – Pushdown Automaton
  – Turing machine

• Parsing of grammars
  – regular, context-free grammars
  – Unrestricted grammar

• Conversions for proofs
  – NFA to DFA to minimal DFA
  – NFA $\leftrightarrow$ regular expression
  – NFA $\leftrightarrow$ regular grammar
  – CFG $\leftrightarrow$ NPDA
JFLAP - L-Systems

- L-Systems may be used to model biological systems and create fractals.
- Similar to Chomsky grammars, except all variables are replaced in each derivation step, not just one!
- Commonly, strings from successive derivations are interpreted as strings of render commands and are displayed graphically.
### Axiom: \( R \sim \### B \)

<table>
<thead>
<tr>
<th>LHS</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B )</td>
<td>( \sim ### TL - B ++ B )</td>
</tr>
<tr>
<td>( L )</td>
<td>( \sim { \text{angle}=15 { -g ++ g % -- g } } )</td>
</tr>
<tr>
<td>( R )</td>
<td>( !@@ R )</td>
</tr>
<tr>
<td>( T )</td>
<td>( T g )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>15</td>
</tr>
<tr>
<td>color</td>
<td>brown</td>
</tr>
<tr>
<td>polygonColor</td>
<td>forestGreen</td>
</tr>
</tbody>
</table>

**L-System** = \((A, \Sigma, R)\)
Expansion contains 264 Symbols!
Expansion contains 545 Symbols!
Add second T rule

<table>
<thead>
<tr>
<th>LHS</th>
<th>RHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>[ \sim ## T L - B + + B ]</td>
</tr>
<tr>
<td>L</td>
<td>[ \text{angle}=15 { - g + + g % - - g } ]</td>
</tr>
<tr>
<td>R</td>
<td>!@@@ R</td>
</tr>
<tr>
<td>T</td>
<td>T g</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>15</td>
</tr>
<tr>
<td>color</td>
<td>brown</td>
</tr>
<tr>
<td>polygonColor</td>
<td>forestGreen</td>
</tr>
</tbody>
</table>

L-System = (A, Σ, R)
Expansion contains 121 Symbols!
L-Systems

The same stochastic L-system, rendered 3 different times all at the 9th derivation.
Students love L-Systems
Outline

• Overview and Motivation
• Examples
• Design Goals of JFLAP
• Teaching with JFLAP
• Research Study
• History of JFLAP
Goal – JFLAP defined generally

• Want JFLAP to fit with all textbooks
• Generalities
  – Empty string: preference - lambda or epsilon
  – FA
    • Single or multiple symbols
    • [1-9] on label to mean characters 1-9
  – PDA
    • Single or multiple symbols (restricted)
    • Accept by empty stack or final state
Goal – JFLAP defined generally (cont)

• Generalities (cont)
  – Turing machine preferences
    • Allow transitions from Final states
    • Accept by final state
    • Accept by halting
    • Allow Stay option (R, L, S) for moving tape head
  – Turing machine other
    • Single or multi tape
    • Building blocks – generalized transitions
      a – read a, write a, do not move tape head
Other Features of JFLAP

• Curve transitions
• Save images in several formats (png, jpg, gif, bmp, svg)
• Undo/redo capability
• Can zoom many of the panes
DFA - Add Trap state and curve the arcs
Example: Build an NFA for valid integers

- Example:
  - Valid integers \{-3, 8, 0, 456, 13, 500, \ldots\}
  - Not valid: \{006, 3-6, 4.5, \ldots\}
Example: NFA for all valid integers
NFA annotated and shortcut

• New feature: [1-9] on labels
Outline

• Overview and Motivation
• Examples
• Design Goals of JFLAP
• Teaching with JFLAP
• Research Study
• History of JFLAP
Using JFLAP during Lecture

- Use JFLAP to build examples of automata or grammars
- Use JFLAP to demo proofs
- Load a JFLAP example and students work in pairs to determine what it does, or fix it if it is not correct.
JFLAP’s use Outside of Class

• Homework problems
  – Turn in JFLAP files
  – OR turn in on paper, check answers in JFLAP
• Recreate examples from class
• Work additional problems
  – Receive immediate feedback
Some Example for Lectures
Example: Show the power of transforming grammars

- Brute Force Parser
  - Give a grammar with a lambda-production and unit production
  - Run it in JFLAP, see how long it takes (LONG)
    - Is $aabbab$ in $L$?
    - Transform the grammar to remove the lambda and unit-productions
  - Run new grammar in JFLAP, runs much faster!
Example (cont)

Parse Tree Results

- First Grammar – 1863 nodes generated
- Second Grammar – 40 nodes generated
- Parse tree is the same.
Example: Nondeterminism

• NPDA for palindromes of even length
• Exercise during class.
• Students have trouble thinking nondeterministically
Example (cont)

- Run input strings on the NPDA
  - Shows the nondeterminism
Outline

• Overview and Motivation
• Examples
• Design Goals of JFLAP
• Teaching with JFLAP
• Research Study
• History of JFLAP
JFLAP Study

• Study of JFLAP’s effectiveness in learning
  – Two year study
  – Fourteen Faculty Adopters
  – Two 2-day faculty Adopter Workshops – June 2005, June 2006
  – Collect data 2005-06 and 2006-07 Academic years
  – Pretest/Posttest
  – Interviews
  – Team of three evaluators
    • Eric Weibe – Education
    • Rocky Ross – Computer Science Theory
    • Joe Bergin – Computer Science Tools
Fourteen Faculty Adopter Participants

- small, large
- public, private
- includes minority institutions

- Duke
- UNC-Chapel Hill
- Emory
- Winston-Salem State University
- United States Naval Academy
- Rensselaer Polytechnic Institute
- UC Davis
- Virginia State University
- Norfolk State University
- University of Houston
- Fayetteville State University
- University of Richmond
- San Jose State University
- Rochester Institute of Technology
Goals of the JFLAP Study - Formal Languages and Automata (FLA)

• Present FLA in a visual and interactive manner in addition to the more traditional approach
  – Integrated

• Present Applications of FLA

• Provide a tool for allowing students to explore FLA in a computational manner

• Provide Materials for instructors to integrate this approach in their courses
Running a Study is hard!

• Hit by the drop in enrollments in after dot-com burst
• IRBs are different process at every institution
  – One page writeup ok’d (simplest)
  – Full medical IRB (many pages)
• One institution shut down all IRB research projects – we could not use data already collected.
• One University - Control Group – different times means different types of students, different professors.
• Some faculty came to workshop and did not follow through
• There were also some fantastic faculty!
Conclusions From Study

• Results of Study showed
  – All the faculty used JFLAP in their courses, mostly for homework, some in lecture
  – Students had a high opinion of JFLAP
  – Majority of students felt access to JFLAP
    • Made learning course concepts easier
    • Made them feel more engaged
    • Made the course more enjoyable
  – Over half the students used JFLAP to study for exams
  – Over half the student thought time and effort using JFLAP helped them get a better grade.
Outline

• Overview and Motivation
• Examples
• Design Goals of JFLAP
• Teaching with JFLAP
• Research Study
• History of JFLAP
How did JFLAP get started?

Back in 1989

• New Assistant Professor at Rensselaer Polytechnic Institute

• New Course: Combined automata theory with CS1 and CS 2 (data structures)

• Student wanted feedback on all the answers in the book!
JFLAP Evolved 1990-present

• Many tools were developed …
• Many students …
• Many years…
• Java came along… made our life easier…
NPDA

- Dan Caugherty – Rensselaer, 1990
- In C++ and X Windows
- DIMACS (1992)
Running NPDA
LR Parser

- Rensselaer
- Mike James 1992
- Steve Blythe 1993
- In C++ and X Windows
- SIGCSE 1994
LR Parser had lots of windows!
• LR parse table

• LR Parser

String to parse: aacbba

Rest of input: aacbba$
Symbol Stack: A$
State Stack: 2 0

Start       | Step

Reducing using the production
A → Aa
JFLAP was born!

- Modified NPDA to add menu, DFA and TM
- Mark LoSacco – 1991
- EdMedia 1993
TuBB- TM Building Blocks

- Rensselaer
- Eric Luce
- VL 1993
JFLAP moves to Java!

- 1996 – Madga and Octavian Procopiuc
- FIE 1996
JFLAP Evolves

• L-systems - Wolfman (1993), Ramm (1998)
• Pumping Lemma – Leider (1996)
• Pate – Salemme and Bilska (1996)
• JeLLRap – Geer and Karweit (1997)
• Proving Theorems to JFLAP – Grammond (1998)
• Regular Expressions to JFLAP – Hung (1999)
JFLAP Moves to Swing!

• 2001 and on – Finley, Cavalcante
  – Complete rewrite
  – Changed some of the algorithms
  – Incorporating many of the previous tools
  – Including L-Systems
  – SIGCSE 2004
JFLAP 5.0 adds Building Blocks

• 2005 – Stephen Reading and Bart Bressler
• SIGCSE 2006
JFLAP book
Rodger and Finley, 2006

• Includes chapters on:
  – Parsing – LL and LR
  – Lsystems
JFLAP 6.0 – later 2006

- Added Moore and Mealy machines
- Pumping lemmas
- Batch run mode
2007 – added CYK

• Convert CNF to CYK, parse, then convert back to show parsing of string.

• CYK algorithm was not Visible!
JFLAP 7.0 - 2009

• Adding zooming
• Undo/redo
• More work on Turing Machines
• Saving in various image file formats
JFLAP 8.0 Beta – Where we are now!

- Formal definition
- Variables and terminals with multiple symbols
- Students: Genkins, McMahon, Godbey, Lin
JFLAP 8.0 Beta (cont)

- Language generator

Grammar: \( V = \{ S \} \)
- \( T = \{ a, b \} \)
- \( S = S \)
JFLAP 8.0 Beta (cont)

- CYK animation
Other ways to make theoretical computer science come alive…
Automata Theory
Interaction in Class – Props
Edible Turing Machine

- TM for $f(x) = 2x$ where $x$ is unary
- TM is not correct, can you fix it? Then eat it!
- States are blueberry muffins
Students building DFA with cookies and icing
Future work
Future Work

- JFLAP to HTML5
- Electronic textbook (ebook) for Automata Theory with JFLAP integrated in
- More curriculum materials
Questions?

JFLAP is free

www.jflap.org

JFLAP tutorial
JFLAP – Regular Languages

• Create
  – DFA and NFA
  – Moore and Mealy
  – regular grammar
  – regular expression

• Conversions
  – NFA to DFA to minimal DFA
  – NFA ↔ regular expression
  – NFA ↔ regular grammar
JFLAP – Regular languages (more)

• Simulate DFA and NFA
  – Step with Closure or Step by State
  – Fast Run
  – Multiple Run
• Combine two DFA
• Compare Equivalence
• Brute Force Parser
• Pumping Lemma
JFLAP – Context-free Languages

• Create
  – Nondeterministic PDA
  – Context-free grammar
  – Pumping Lemma

• Transform
  – PDA \( \rightarrow \) CFG
  – CFG \( \rightarrow \) PDA (LL \& SLR parser)
  – CFG \( \rightarrow \) CNF
  – CFG \( \rightarrow \) Parse table (LL and SLR)
  – CFG \( \rightarrow \) Brute Force Parser
JFLAP – Recursively Enumerable Languages

• Create
  – Turing Machine (1-Tape)
  – Turing Machine (multi-tape)
  – Building Blocks
  – Unrestricted grammar

• Parsing
  – Unrestricted grammar with brute force parser
This L-System renders as a tree that grows larger with each successive derivation step.