SIGCSE – Passionate Educators on the Teaching and Research of Computer Science Education

Susan H. Rodger
Duke University
July 23, 2016
Peking University, Beijing, China

Supported by NSF Grants DRL-1031351, DUE 1044191, CRA-W, and IBM Faculty Awards
About Me

• Professor of the Practice of Computer Science
• Area: Visualization and Animation, Computer Science Education

• Passionate about education/diversity
  • Past SIGCSE Chair
  • ACM Education Policy Committee
  • CRA-W Board Member
About Me - Hobby – Baking Shape cakes

The Wiggles magazine Issue No. 42
How do you make those cakes?
CS 1
Sorting
Cookies
Outline

• SIGCSE
  • History, Goals and Programs

• SIGCSE – Some Communities of Research
  • Algorithm Visualization
    • Automata Theory with JFLAP
  • Peer Led Team Learning
  • Integrating Computing into Secondary Schools
ACM Special Interest Groups (SIGS)

• Over 30 SIGs
  • SIGCSE, SIGITE, SIGAI, SIGIR, SIGDD, SIGEVO, SIGAPP, SIGBio, SIGMOBILE, SIGHPC, SIGSAM, SIGSPATIAL, SIGecom, SIGKDD, SIGWEB, SIGMETRICS, SIGMM, SIGMOD, SIGSAC, SIGCAS, SIGUCCS, SIGBED, SIGDA, SIGMICRO, SIGCHI, SIGACCESS, SIGDOC, SIGGRAPH, SIGCOMM, SIGOPS, Sigmis, SIGARCH, SIGPLAN, SIGAda, SIGSIM, SIGSOFT, SIGACT, SIGLOG
ACM SIGCSE History – The Beginning

• Special Interest Group on Computer Science Education (SIGCSE)

• SIGCSE Board started in 1968
  • Chair – Elliott Organick (‘68–’70)
  • Lots of Board members:
    • Robert Aiken, David Matula, Jack Belzer, Peter Calingaert, Thomas Keenen, Earl Schwepppe, William Vivant, William Atchinson, S. D. Conte
SIGCSE Mission Statement

SIGCSE is to provide a forum for problems common among educators working to develop, implement and/or evaluate computing programs, curricula, and courses, as well as syllabi, laboratories, and other elements of teaching and pedagogy.
SIGCSE has 3 Conferences

- SIGCSE Symposium
- ITiCSE
- ICER
Created First SIGCSE Conference
SIGCSE 1970

• November 16, 1970
• Houston, Texas
• Over 40 papers submitted, 18 accepted
• Conference Chairs:

Peter Calingaert       Edward Feustel
SIGCSE 2016 Symposium

• 47th SIGCSE Symposium!
• 297 papers submitted, 105 accepted
• Around 1300 attendees
What is the SIGCSE Symposium like?

• Papers
• Panels
• Keynote Speakers
• Workshops (3 hours)

• Exhibit Hall
Pre-events

Posters

BOFs
SIGCSE’s 2\textsuperscript{nd} Conference - ITiCSE

• First ITiCSE in 1996 – Barcelona, Spain
  • 46 papers accepted
  • Boots Cassel and Jim Hightower - Chairs
• European version of SIGCSE Symposium
• Smaller – 100-300 attendees
• Working Groups
• Tour on middle day
• Papers, one panel
• Posters and Demos
21\textsuperscript{st} ITiCSE – 2016
Arequipa, Peru

• 53 papers
• 7 Working Groups
• Conference banquet in Monastary
• Great Tours after conference

• Next year back to Europe
SIGCSE’s 3rd Conference - ICER

• First ICER in 2005

• Chairs: Richard Anderson, Sally Fincher, Mark Guzdial

• Smaller – 60-100 attendees

• Focus: Computer Science Education Research

• One track papers – with discussions

• Doctoral Consortium day before
SIGCSE Projects

- New Educator’s Workshop
  - Mentoring on teaching, research, promotion
  - Graduate Students about to finish, Assistant Professors

- Chairs Workshop
  - Advice to Dept Chairs (current and future)

- Special Projects
  - Proposals due November and May
  - Up to $5000

- Speaker’s Fund
  - $1000 for a speaker for a conference on topic presented at a SIGCSE conference.
SIGCSE Projects (cont)

• Two Awards
  • Outstanding Educator, and Lifetime Service

• Travel Grants
  • For faculty who have not attended a SIGCSE Symposium - $500 plus registration

• In-Cooperation with SIGCSE
  • For conferences in Computer Science Education – pathway to papers in ACM Digital Library
SIGCSE Chapters

- Australasian ACM SIGCSE Chapter
- Bilkent ACM SIGCSE Chapter
- India iSIGCSE
- Spain ACM SIGCSE
SIGCSE Member Benefits

- Conference discounts
SIGCSE – Current Issues

• Focus on growing our youngest conference, ICER.
  • 20 DC participants at ICER 2015
  • Around 120 participants
• Working with ACM and IE on more of a presence in CS Education Research in Europe
• Investigating an ITiCSE-like conference that can go around the world and be held at different times of the year.
Outline

• SIGCSE
  • History, Goals and Programs
• SIGCSE – Some Communities of Research
  • Algorithm Visualization
    • Automata Theory with JFLAP
  • Peer Led Team Learning
  • Integrating Computing into Secondary Schools
All Forms of Algorithm Visualization

• What data structure is this?
YARN, in the shape of a binary tree. Subtrees made with molecule kit. What is it?
2D-range tree

• Search in x-y plane
• Main tree organized by x-values
• Subtree organized by y values
Binary Search tree of points in the plane – sorted by X-value

In the x-range

Each subtree organized by y-value

Search each subtree by y-value
Different Types of Learners

• Learning Styles
  – Visual Learners
    • Learn through seeing
    • Learn best from visual displays
  – Auditory Learners
    • Learn through listening
    • Learn best through verbal lectures, discussions
  – Kinesthetic Learners
    • Learn through moving, doing and touching
    • Learn best through hands-on approach
How do you reach all three types?

• You must do all three!
  – Provide pictures, diagrams
  – Discuss what you are doing
  – Provide activities for trying it
Learner Engagement Taxonomy with visualization software

• Different forms of Learner engagement
  – No Viewing
  – Viewing
  – Responding
  – Changing
  – Constructing
  – Presenting

• ITiCSE Working Group Report 2002 (Naps et al.)
Algorithm Visualization/Animation Software/Aps/Videos

• Tango, Xtango, Samba, JSamba - Stasko (Georgia Tech)
• AnimalScript – Roessling (Darmstadt Univ of Tech, SIGCSE 2001)
• JHAVE – Naps (U. Wisc. Oshkosh, SIGCSE 2000)
• TRAKLA2 – Software Visualization Group – TKK Finland
• Lots of animations and systems on the web!
• Lots of videos of algorithm animations on the web!
Use of Algorithm Animation in CS 1/2

- **Instructor**
  - Make/Use animations for lecture
  - Stop/Pause – ask what will happen next
  - must be interactive

- **Student**
  - Create animations
  - Replay animations from lecture with same or new inputs
Python Tutor
Compute reverse of a list

```python
def reverse(numbers):
    answer = []
    for num in numbers:
        answer.insert(0, num)
    return answer

myList = [4, 7, 8, 3]
reversed = reverse(myList)
```

Edit code
Python Tutor
Compute reverse of a list

```python
def reverse(numbers):
    answer = []
    for num in numbers:
        answer.insert(0, num)
    return answer

myList = [4, 7, 8, 3]
reversed = reverse(myList)
```
Python Tutor
Compute reverse of a list

```python
def reverse(numbers):
    answer = []
    for num in numbers:
        answer.insert(0, num)
    return answer

myList = [4, 7, 8, 3]
reversed = reverse(myList)
```

Edit code
Python Tutor
Compute reverse of a list

```python
1 def reverse(numbers):
2     answer = []
3     for num in numbers:
4         answer.insert(0, num)
5     return answer
6
7 myList = [4, 7, 8, 3]
8 reversed = reverse(myList)
```

```
Frames
Global frame
  reverse
  myList
```
```
Objects
  list
    4 7 8 3
  empty list
  numbers
  answer
  num
    4
```
Python Tutor
Compute reverse of a list

```python
1 def reverse(numbers):
2     answer = []
3     for num in numbers:
4         answer.insert(0, num)
5     return answer

6 myList = [4, 7, 8, 3]
7 reversed = reverse(myList)
```

Edit code
Python Tutor
Compute reverse of a list

```python
def reverse(numbers):
    answer = []
    for num in numbers:
        answer.insert(0, num)
    return answer

myList = [4, 7, 8, 3]
reversed = reverse(myList)
```
Python Tutor
Compute reverse of a list

```python
def reverse(numbers):
    answer = []
    for num in numbers:
        answer.insert(0, num)
    return answer

myList = [4, 7, 8, 3]
reversed = reverse(myList)
```

Edit code
Python Tutor
Compute reverse of a list

```python
1    def reverse(numbers):
2        answer = []
3        for num in numbers:
4            answer.insert(0, num)
5        return answer
6
7    myList = [4, 7, 8, 3]
8    reversed = reverse(myList)
```

Edit code

Frames
- Global frame
  - reverse
  - myList
- function reverse(numbers)
- list
  - 4
  - 1
  - 7
  - 8
  - 3
- list
  - 0
  - 1
  - 2
  - 3
  - 4

Objects
AlgoViz – Repository of Algorithm Visualizations
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)
Index Operator: Working with the Characters of a String

The indexing operator (Python uses square brackets to enclose the index) selects a single character from a string. The characters are accessed by their position or index value. For example, in the string shown below, the 14 characters are indexed left to right from position 0 to position 13.

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13
L u t h e r  C o l l e g e
```

It is also the case that the positions are named from right to left using negative numbers where -1 is the rightmost index and so on. Note that the character at index 6 (or -8) is the blank character.

```python
school = "Luther College"
m = school[2]
print(m)
lastchar = school[-1]
print(lastchar)
```
Run and edit code in the book

```python
1 school = "Luther College"
2 m = school[2]
3 print(m)
4
5 lastchar = school[-1]
6 print(lastchar)
```
Integrates in Python Tutor

```python
1 school = "Luther College"
2 m = school[2]
3 print(m)
4 lastchar = school[-1]
5 print(lastchar)
```
Questions for feedback

Check your understanding

strings-4-1: What is printed by the following statements?

```
s = "python rocks"
p = print(s[3])
```

- t
- h
- c
- Error, you cannot use the [ ] operator with a string.

Check Me  Compare me

Incorrect. Index locations do not start with 1, they start with 0.

Strings-4-2: What is printed by the following statements?

```
s = "python rocks"
p = print(s[2] + s[-5])
```

- tr
- ps
- nn
- Error, you cannot use the [ ] operator with the + operator.

Check Me  Compare me

Correct! Yes, indexing operator has precedence over concatenation.
Outline

• SIGCSE
  • History, Goals and Programs

• SIGCSE – Some Communities of Research
  • Algorithm Visualization
    • Automata Theory with JFLAP
  • Peer Led Team Learning
  • Integrating Computing into Secondary Schools
Java Program to count number of words in a phrase

```java
public static void main() {
    String phrase = "baby fish should be fed three times a day";
    int numWords = 0;
    int pos = phrase.indexOf(" ");
    while (pos > 0) {
        numWords += 1;
        phrase = phrase.substring(pos+1);
        pos = phrase.indexOf(" ");
    }
    numWords += 1;
    System.out.println("Number of words in phrase is " + numWords);
}
```

• Is this program syntactically correct?
First identify each word/token

```
public static void main() {
    // Code goes here
}
```
First identify each word/token (cont)

String phrase = "baby fish should be fed three times a day";

<table>
<thead>
<tr>
<th>TOKEN</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>keyword</td>
</tr>
<tr>
<td>phrase</td>
<td>variable</td>
</tr>
<tr>
<td>=</td>
<td>equals</td>
</tr>
<tr>
<td>&quot;</td>
<td>doublequote</td>
</tr>
<tr>
<td>baby fish ... day</td>
<td>string</td>
</tr>
<tr>
<td>&quot;</td>
<td>doublequote</td>
</tr>
<tr>
<td>;</td>
<td>semicolon</td>
</tr>
</tbody>
</table>
First identify each word/token (cont)

```c
int numWords = 0;
```

<table>
<thead>
<tr>
<th>TOKEN</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>keyword</td>
</tr>
<tr>
<td>numWords</td>
<td>variable</td>
</tr>
<tr>
<td>=</td>
<td>equals</td>
</tr>
<tr>
<td>0</td>
<td>int</td>
</tr>
<tr>
<td>;</td>
<td>semicolon</td>
</tr>
</tbody>
</table>
Define Rules for a valid program (Grammar)

<program> := <procDef> ( ) { <body> }
<program> := <procDef> (<arglist> ) { <body> }
<procDef> := public static void main
<procDef> := <pubtype> <returntype> <variable>
<body> := <decllist> <stmtlist>
<declist> := <decl> ; <decllist>
<declist> := <decl> ;
<stmtlist> := <stmt> ; <stmtlist>
<stmtlist> := <stmt> ;
Define Rules for a valid program (Grammar)

<decl> ::= int <variable> = <integer> ;
<decl> ::= String <variable> = " <string> " ;
<stmt> ::= while ( <cond> ) { <stmtlist> }
<stmt> ::= <variable> += <integer> ;
<stmt> ::= <variable> = <integer> ;

etc
Now derive the program using the rules (grammar)

\[
\text{<program> := <procDef> ( ) { <body> } }
\]

\[
:= \text{public static void main () { <body>}}
\]

\[
:= \text{public static void main () { <declist> <stmtlist> }}
\]

\[
:= \text{public static void main()} { <decl> ; <declist> <stmtlist> }
\]
And so on until you derive the program

```java
public static void main() {
    String phrase = "baby fish should be fed ...";
    int numWords = 0;
    int pos = phrase.indexOf(" ");
    while (pos > 0) {
        numWords += 1;
        ...
    }
}
```
Determining if a Java program is syntactically correct

- Finite state machine (or determinisitic finite automaton - DFA) – to identify the words or tokens of the program
- Context-free grammar – to write the rules of the programming language
- LR Parsing determining if the program fits the rules – trying to derive the program.

- This area is known as Formal languages and Automata theory
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

• **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 8.0Beta – 2011-14 Julian Genkins, Ian McMahon, Peggy Li, Lawrence Lin, John Godbey
- JFLAP in OpenDSA – 2015 Sung-Hoon Kim and Martin Tamayo

Over 25 years!
### Why Develop Tools for Automata?

| Textual   | $\langle \{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\} \rangle$
<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>

| Tabular    | ![](tabular.png) 
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\begin{array}{c</td>
</tr>
</tbody>
</table>

| Visual     | ![](visual.png) 
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="visual.png" alt="Visual Diagram" /></td>
</tr>
</tbody>
</table>

| Interactive| ![](interactive.png) 
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="interactive.png" alt="Interactive Diagram" /></td>
</tr>
</tbody>
</table>
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 no. of 1’s | Yes | No, ends In 1 | Yes |
Build with JFLAP

-start

-q0

-q1

-q2

-q3

-q4

-finish
Simulation on 1101010
Simulation on $1101010$
Simulation on 1101010
Simulation on 1101010
Simulation on 1101010
Simulation on 1101010
Simulation on
1101010
Accepts Input!

11010110
Add meaning to states!
The automaton in the image represents a finite automaton (FA) that recognizes strings with an odd number of 1's. The states are labeled as follows:
- q0: only one 0
- q1: odd number of 1's
- q2: 0
- q3: 1
- q4: 0

The transitions are:
- From q0 to q1 on input 1
- From q1 to q2 on input 0
- From q2 to q4 on input 1
- From q4 to q4 on input 0
The diagram illustrates a finite state machine (FSM) designed to recognize binary numbers with an even number of 1s, ending in 1.

- **q0**: Start state.
- **q1**: Transition on 0 to itself, labeled "only one 0".
- **q2**: Transition on 1 to **q3**, labeled "odd number of 1's".
- **q3**: Transition on 0 to **q4**, labeled "even number of 1's, ends in 1".
- **q4**: Transition on 1 back to **q2**.
- **q4**: Transition on 0 back to itself.
The diagram depicts a finite automaton with states labeled q0, q1, q2, q3, and q4. The transitions are labeled with 0 and 1, and the states are connected as follows:

- q0 transitions to q1 on 0, and to q2 on 1.
- q1 has a transition to q0 on 0, labeled "only one 0".
- q2 transitions to q3 on 0, labeled "odd number of 1's".
- q3 transitions to q4 on 0, labeled "even number of 1's, ends in 1".
- q4 transitions to q0 on 1, labeled "even number of 1's, ends in 0".

The automaton accepts a binary number if it ends in q3 or q4.
Test Multiple Inputs

The diagram shows a finite state machine (FSM) with multiple states and transitions. The states are labeled as follows:
- q0: odd number of 1's
- q1: only one 0
- q2: odd number of 1's
- q3: even number of 1's, ends in 1
- q4: even number of 1's, ends in 1

The table lists various inputs and their corresponding results:
- 10: Reject
- 111: Reject
- 1010: Accept
- 10110: Reject
- 101: Reject
- 1100: Accept
- 110110: Accept
Example: Build an NFA for valid integers

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

• Shortcut: [1-9] on labels
Back to Recognizing whether a Java Program is syntactically correct or not...

• You would need a DFA to recognize all valid words in a program
  • An integer
  • A variable name
  • All keywords
  • All special symbols ; + - ( ) {}
  • etc
Another Example: Grammar

• Grammar – set of replacement rules to define a language
• Previously looked at grammar for Java (very small part of it!)
• Grammar for a formal language (simpler)
• Consider representing underlined words in a text file (to be interpreted later):
  • cookie&&&&&&______                   cookie
    & = go back one
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
  - aabbcc
  - aaabbbccc
  - aaaaabbbbbcccccc
  - ...
Example Derivation for aabbcc

S → AX

rule: $S \rightarrow AX$
Example Derivation for aabbcc

\[ S \rightarrow AX \]
\[ \rightarrow aAbcX \]

rule: \( S \rightarrow AX \)
rule: \( A \rightarrow aAbc \)
Example Derivation for aabbcc

S → AX          rule: S -> AX
→ aAbcX         rule: A -> aAbc
→ aaBbcbcX      rule: A -> aBbc

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

S → AX
→ aAbcX
→ aaBbcbcX
→ aaBbBcbbcX

rule: S -> AX
rule: A -> aAbc
rule: A -> aBbc
rule: Bb -> bB
Example Derivation for aabbcc

S → AX     rule: S -> AX
   → aAbcX   rule: A -> aAbc
   → aaBbcbcX rule: A -> aBbc
   → aabBcbcX rule: Bb -> bB
   → aabDbcX  rule: Bc -> D

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

\[ S \rightarrow AX \]  
\[ \rightarrow aAbcX \]  \text{rule: } S \rightarrow AX  
\[ \rightarrow aaBbcbcX \]  \text{rule: } A \rightarrow aAbc  
\[ \rightarrow aabBcbcX \]  \text{rule: } A \rightarrow aBbc  
\[ \rightarrow aabDbcX \]  \text{rule: } Bb \rightarrow bB  
\[ \rightarrow aabbDcX \]  \text{rule: } Bc \rightarrow D  
\[ \rightarrow aabbbDcX \]  \text{rule: } Db \rightarrow bD  

Example Derivation for aabbcc

S → AX
→ aAbcX
→ aaBbcbcX
→ aabBbcbcX
→ aabDBCbcX
→ aabbDcX
→ aabbcDX
Example Derivation for aabbccc

S → AX
→ aAbcX
→ aaBbcbcX
→ aabBbcbcX
→ aabDbcX
→ aabbbDcX
→ aabbbcDc
→ aabbcDX
→ aabbcEXc

Eventually ... → aabbcc

rule: S -> AX
rule: A -> aAbc
rule: A -> aBbc
rule: Bb -> bB
rule: Bc -> D
rule: Db -> bD
rule: Dc -> cD
rule: Dx -> Exc

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

Now let’s see how JFLAP visualizes this derivation with a “parse tree”

Parse DAG
<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>

String accepted! 51 nodes generated.
Input: aabbcc
String accepted! 51 nodes generated.

Derived AX from S.
String accepted! 51 nodes generated.

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

LHS | RHS
--- | ---
S   | AX
A   | aA\text{BbC}
A   | aABbC
Bb  | bB
Bc  | D
Dc  | cD
Db  | bD
DX  | EXc
BX  | \lambda
cE  | Ec
bE  | Eb
aE  | aB

Derived bD from Db.
Input: aabbcc
String accepted! 51 nodes generated.

Derived cD from Dc.
Spit out the “c” at the right end
String accepted! 51 nodes generated.

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 | \( \lambda \)
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.
### 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  |

String accepted! 51 nodes generated.

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

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

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

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

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

Derived aB from aE.
Input: abbc
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.

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

Derived bB from Bb.
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>$[\text{angle}=15 { -g + + g % - - g }]$</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 = $(A, \Sigma, R)$
R ~ ## B
!@@ R ~ ## [ ~ ## TL - B ++ B ]
! @@! @@ R ~ ## [ ~ ## T g { angle=15 \{ - g + + g \% - g \}} - [ ~ ## T 2
Expansion contains 545 Symbols!
Add second T rule

<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)
JFLAP v8.0 (Beta) (ex10-tree-thick-fall-leaves.jff)

File   Help
L System   L-S Render

!@@! @@ R~##[~ ##T[angle=15{-g++g%--g}]-##TL

Pitch 0 Roll 0 Yaw 0
Expansion contains 121 Symbols!
Expansion contains 4350 Symbols!
L-Systems

The same stochastic L-system, rendered 3 different times all at the 9th derivation.
Students like L-systems
Two-year JFLAP Study 2005-2007

Fourteen Faculty Adopter Participants

- 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

-small, large
-public, private
-includes minority institutions
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 students thought time and effort using JFLAP helped them get a better grade.
What happens when your hobby and your career collide?

It is now time for engaging students with edible CS
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
JFLAP is free

www.jflap.org

JFLAP tutorial
Outline

• SIGCSE
  • History, Goals and Programs
• SIGCSE – Some Communities of Research
  • Algorithm Visualization
    • Automata Theory with JFLAP
• Peer Led Team Learning
• Integrating Computing into Secondary Schools
Motivation for Peer-Led Team Learning

• Goals
  • Increase number of women and underrepresented groups
  • Increase retention and enthusiasm

• Approaches
  • Active Recruiting of Incoming First-year students
  • Optional/Required of registered students
8 NSF-FUNDED SCHOOLS
What is PLTL?

• Related to a course
  – Students solve problems in small groups (4-8 students) weekly in addition to regular class meeting
  – Interesting exercises to be solved as a group
  – Led by trained undergraduate student leaders who facilitate group learning

• Used in Chemistry, www.pltl.org

• Beneficial to both students and student leaders
Why PLTL?

• Factors affecting intellectual development in college
  • Student/faculty interaction outside the classroom
  • Involvement on campus through various forms of community-building activities
  • Involvement with student peer groups
  • “peer group – the most potent source of influence on growth and development during the undergraduate years.”
Cone of Learning
(Edgar Dale)

After 2 weeks
we tend to remember...

10% of what we read
20% of what we hear
30% of what we see
50% of what we hear and see
70% of what we say
90% of what we say and do

Reading
Hearing words
Looking at pictures
Watching a movie
Looking at an exhibit
Watching a demonstration
Seeing it done on location
Participating in a discussion
Giving a talk
Doing a dramatic presentation
Simulating the real experience
Doing the real thing

Nature of involvement
Passive
Active

Defining PLTL in CS (also called ESP-PLTL)

• Small groups meet related to a course
  • Not everyone from the course
  • Build friendships to help support you through major

• Active recruiting

• Aim for gender balance

• Undergraduate peer leaders

• Solve challenging problems
Peer-Led Team Learning in CS (PLTL in CS)

• Combines components from PLTL and ESP
• Eight Universities – Fall 2005 – Spring

Beloit College (WI)                              Purdue University (IN)
Duke University (NC)                             Rutgers University (NJ)
Georgia Tech (GA)                                University of Wisconsin Madison (WI)
Loyola College (MD)                              University of Wisconsin Milwaukee (WI)

• www.pltlcs.org

Supported by the National Science Foundation collaborative Grants CNS-0420436, 0420343, 0419340, 0420433, 0420358, 0420312, 0420368, 0420337, 0638510 and 0638499 and a donation from Microsoft.
Duke University - “PLTL in CS” version
Emerging Scholars Program (DES)

• One year program – four courses total
  – First semester
    • Main course: Non-majors course: CPS 4 (Alice) (1 credit)
    • Problem Solving Seminar course: CPS 18S (1/2 credit)
  – Second Semester
    • Main Course: CS 1 course: CPS 6 (Java)
    • Problem Solving Seminar course: CPS 18S (1/2 credit)
  – Active Recruiting (email to targeted groups, accepted student fairs, invite students in main course)
  – Gender balanced
  – Outside Speaker/Field Trip
  – Undergraduate Peer Leaders in Problem Solving Seminar
CompSci 18S: Problem Solving Seminar

• 2 peer leaders, about 12 students, (1 professor)
• Solve problems in groups of 4
• Either general computer science problems or related to the main course
• Subset of students from main course, those who want the group experience
• Peer leaders trained in workshop, meet weekly
2 Main Courses: Non-majors (Alice) and CS 1 (Java)

• Workshop format
  – Lecture 10-20 minutes
  – Students program rest of class
  – Students work in pairs (“pair programming”)
    • Two people, two laptops, consult a lot
  – Assigned seats and pairs, change every 2-3 weeks

• About 35-50 students
2 Main Courses: Undergraduate role

• About 8-10 undergraduate teaching assistants

• Roles:
  • Attend the “workshop lecture” to assist
  • Meet weekly
  • Grade and hold consulting hours
  • Includes the two peer leaders from the problem solving seminar
Example of Problem Solving: Be A Robot

• Group of 4 – brain, eyes, 2 hands
• Only brain knows what you are building
• Only eyes can see
• Must work together precisely like a robot
Results from Study

Results - Why enroll in main course?

31 female/49 male responses 2005 (select all that apply)

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>M</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>71.0%</td>
<td>22.5%</td>
<td>I received an invitation</td>
<td></td>
</tr>
<tr>
<td>67.7%</td>
<td>28.6%</td>
<td>To see whether I enjoy CS</td>
<td></td>
</tr>
<tr>
<td>29.0%</td>
<td>40.8%</td>
<td>Meets requirement for my major</td>
<td></td>
</tr>
<tr>
<td>25.8%</td>
<td>79.6%</td>
<td>I know I am interested in CS</td>
<td></td>
</tr>
<tr>
<td>19.4%</td>
<td>18.4%</td>
<td>Programming is useful job-market skill</td>
<td></td>
</tr>
<tr>
<td>16.1%</td>
<td>57.1%</td>
<td>I plan to major in CS</td>
<td></td>
</tr>
</tbody>
</table>
# Retention Data

<table>
<thead>
<tr>
<th>Retention Data, All Institutions Combined (2005 - 2007)</th>
<th>ESP-PLTL</th>
<th>Non ESP-PLTL</th>
<th>Total (All Students)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#</td>
<td>%</td>
<td>#</td>
</tr>
<tr>
<td>Completed</td>
<td>383</td>
<td>93.2%</td>
<td>2363</td>
</tr>
<tr>
<td>Dropped</td>
<td>28</td>
<td>6.8%</td>
<td>323</td>
</tr>
<tr>
<td>Total</td>
<td>411</td>
<td>100.0%</td>
<td>2686</td>
</tr>
</tbody>
</table>
Final Grade Data, all Institutions 2005-2007

<table>
<thead>
<tr>
<th></th>
<th>All ESP-PLTL</th>
<th></th>
<th>All Non-ESP-PLTL</th>
<th></th>
<th>Total (All Students)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#</td>
<td>%</td>
<td>#</td>
<td>%</td>
<td>#</td>
</tr>
<tr>
<td>B or better</td>
<td>219</td>
<td>80.2%</td>
<td>1130</td>
<td>68.4%</td>
<td>1349</td>
</tr>
<tr>
<td>Less than B</td>
<td>54</td>
<td>19.8%</td>
<td>522</td>
<td>31.6%</td>
<td>576</td>
</tr>
<tr>
<td>Total</td>
<td>273</td>
<td>100.0%</td>
<td>1652</td>
<td>100.0%</td>
<td>1925</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>ESP-PLTL Female</th>
<th></th>
<th>Non-ESP-PLTL Female</th>
<th></th>
<th>Total (All Females)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#</td>
<td>%</td>
<td>#</td>
<td>%</td>
<td>#</td>
</tr>
<tr>
<td>B or better</td>
<td>70</td>
<td>83.3%</td>
<td>295</td>
<td>70.1%</td>
<td>365</td>
</tr>
<tr>
<td>Less than B</td>
<td>14</td>
<td>16.7%</td>
<td>126</td>
<td>29.9%</td>
<td>140</td>
</tr>
<tr>
<td>Total</td>
<td>84</td>
<td>100.0%</td>
<td>421</td>
<td>100.0%</td>
<td>505</td>
</tr>
</tbody>
</table>
Advantages for Peer Leaders (telephone interview)

• Common themes emerged
  • Improved Leadership skills
  • Opportunity to try out educator role
  • Reinforcement of understanding CS concepts
  • Increased confidence to continue in field
  • Friendships with students
  • Would recommend experience to others
Summarizing results

• Active Recruiting increased number of women
  • Email/mailed invitation was most effective
• Retention of PLTL in CS students was higher
• Grades of PLTL in CS students was higher
• Friendships and Bonding occurred with students
• Advantages for Peer Leaders too
• PLTL in CS workshop April 2007 at Duke
Outline

• SIGCSE
  • History, Goals and Programs

• SIGCSE – Some Communities of Research
  • Algorithm Visualization
    • Automata Theory with JFLAP
  • Peer Led Team Learning
    • Integrating Computing into Secondary Schools
Alice Programming Language

• Create interactive stories or games
• Learn programming in an easy way, drag-and-drop your code
• Problem solving with visual feedback
  • Objects are visual!
• Alice is free: www.alice.org
• Developed by Randy Pausch
More on ... Alice Programming Language

• Has libraries of 3D objects

• Keeps Track of objects you select
Objects Have Multiple Parts that are moveable
Alice Code is Easy to Learn

Select Code, Drag-and-Drop code in program
Play Alice Animation

• Chicken rises, cow turns head and talks
Why Alice?

• Lots of other great tools for teaching programming

• Alice is easy to use, drag-and-drop, objects already exist

• Storytelling - Attractive to both girls and boys
Success - Alice attracts diverse group

- At Duke
  - CompSci 4 Spring 2005
    - 22 preregister, 30 enroll (12 female + 3 African Amer.)
  - CompSci 4 Fall 2005
    - 20 preregister, 31 enroll (17 female + 1 African Amer.)
  - CompSci 4 Fall 2006 – 2 sections
    - 64 students, 33 female, 7 African Amer.
  - CompSci 4 Fall 2007 – 2 sections
    - 84 students -> 50% female
  - CompSci 4 Fall 2008 – 2 sections
    - 100 students -> 50% female
  - Same for Spring 2009, Fall 2009...
  - Advertised in school paper
    - picture of ice skater
    - Web site of animations
  - This course is now CompSci 94
Success - Alice Excites 4\textsuperscript{th}-6\textsuperscript{th} Grade Girls

- Duke Femmes Event, April 07
- 60 girls – 4 groups of 15
- Taught them Alice for an hour
- Handout to take home
- Event again in 2008, almost every year since
Adventures in Alice Programming
www.cs.duke.edu/csed/alice/aliceInSchools

• 2-week Teacher workshops
  • Over 200 teachers, middle school, high school, some elementary
  • First week Teach Alice, Practice
  • Second week - Develop Lesson Plans
  • All disciplines: math, science, history, language arts, foreign language, art, music, business
• Summers 2008-2016, funding for lodging

• Main Sites:
  • Duke University, Durham, NC
  • Charleston/Columbia, SC
  • UNL - Lincoln, NE
Curriculum Materials
www.cs.duke.edu/csed/alice/aliceInSchools

• Over 90 tutorials available for free
• Beginner, advanced, challenges, projects
• Paper handouts and video
• Over 200 Teacher lesson plans
  • Organized by discipline and grade level
Example: Objects are visible

Getting Started Tutorial teaches

• Placing objects
• Moving objects
• Setting up Camera tripods and moving between views
• Using built in methods and writing your own
  • Dragon flapWings
• Gluing objects together
• Adding sound, 2D pictures to enhance world
Getting Started Tutorial – 3 part
Science Example
How a volcano is formed
Tutorial for Adventure Game – Find objects in order
What a 6\textsuperscript{th} grader can do with Alice - teacher Chari Distler
No Superheros in Alice
Thank You

• Questions?