Computational Thinking

Computational Thinking for Computer Science (CT4CS) Students

Virginia Tech

Background

- An undergraduate course at Virginia Tech
- Offered twice as an alternative to a required problem-solving class
- Experience report at SIGCSE, 2011 in paper co-authored with Deborah Tatar

Summary

- Motivation
  - Aspirational: help computer science students develop intuitions, mental models, and patterns of thinking about computation ("think like a computer scientist")
  - Pragmatic: engage students in learning experiences related to recurring, fundamental concepts about computation
- Means
  - A non-programming entry level CS course
  - An array of editing/visualization/simulation tools and physical simulations
- Results
  - Survey of first offering (N=17) and experiential evidence
  - Sufficiently encouraging to pursue a (current) second offering
  - Able to deal with many core computing concepts
  - No good approach (yet) to algorithmic concepts
- URL: www.cs.vt.edu/~kafura/ComputationalThinking

Overview

- Motivation
- Class Outline
- An example
- Evaluation
- Conclusions

Motivation

Computational Thinking

- conveys essential thought processes about computation
- usually to non-CS students
- informs discipline-specific ways of looking at the world
- elevates computational sophistication
- improves collaboration with computer scientists

Computer Science

- it is “not just programming”
- accessible regardless of background
- approaches
  - contextualized programming
  - problem-solving
  - great ideas/principles
  - survey of discipline

Computational Thinking for Computer Science

- conveys essential thought processes about computation
- to computer science students
- without requiring or using programming
- in concrete, tangible forms

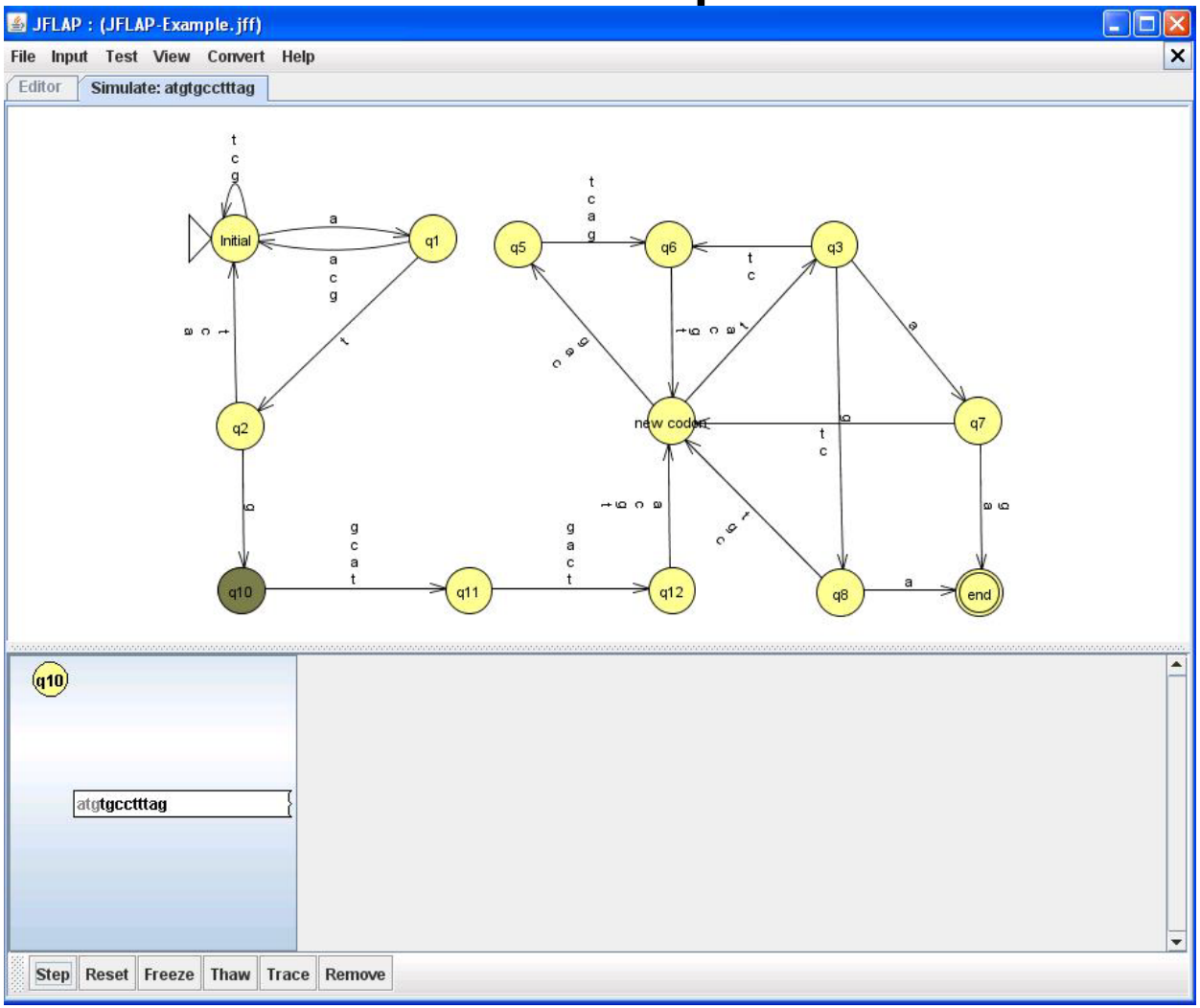
Class outline

Wks	Topic		Concepts/Tools
.5	Modeling	Definition of CS	Guided discussion
2		State,behavior	Finite state machines, acceptors, grammars; **Tools: JFLAP, ANTLR**
2		Abstraction	Abstraction, generalization, composition using Venn/tree/UML diagrams, XML; **Tools: XMLSpear, physical simulation**
1.5		Relationships	Representing, inferring, visualizing relationships, ontologies; **Tool: Protege**
1.5	Engineering	Concurrency	Race conditions, synchronization, Petri nets; **Tool: Snoopy, physical simulation**
1		Abstraction	Layered systems/protocols; **Tool: Snoopy**
2		Binding, scope	Lambda calculus; **Tool: Lambda Teacher**
1		Testing	Developing test cases, coverage; **Tools: applet, WebCAT**
1		Debugging	Puzzle solving with backtracking; **Tool: Sodoku system**
1		Data structures	Mapping complex structures to memory; **Tool: physical simulation**

Example (1)

- Concepts
  - Finite states --- transitions --- inputs/events
- Assignment
  - Develop a finite state acceptor to recognize if a DNA sequence is a possible gene
- Tool
  - JFLAP

Gene Acceptor in JFLAP



Example (2)

- Concepts
  - Languages – structure – grammar
- Assignment
  - Develop a BNF grammar for US Currency
- Tool
  - ANTLR

Grammar in ANTLR

ANTLRWorks 1.4.2

File Edit Find Go To Grammar Refactor Generate Run Window Help

C:\Documents and Settings\Dennis\Desktop\SIGCSE\Presentation\ANTLR-Example.g

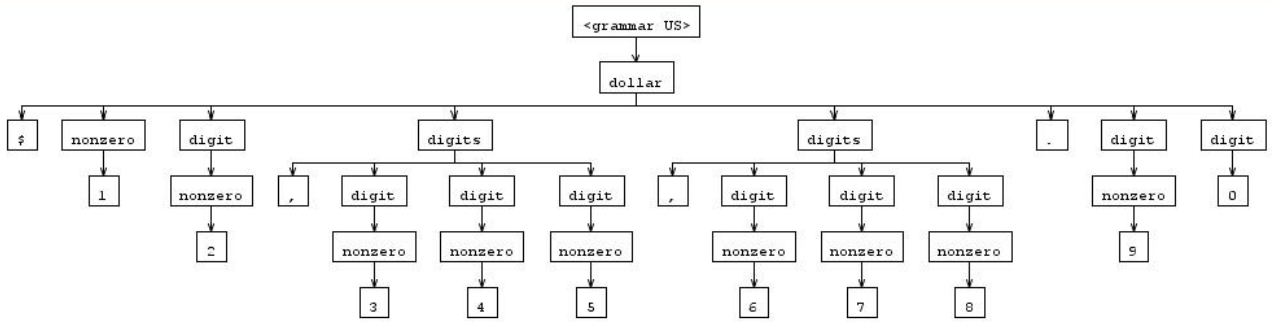
ANTLR-Example.g
- nonzero
- digit
- digits
- dollar

```
//10 points
//very slick

grammar US;
nonzero :       '1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9' ;
digit   :       nonzero | '0' ;
digits  :       ',' digit digit digit ;
dollar  :       '$' (nonzero digit? digit? digits* | nonzero? digit) '.' digit digit ;
```

dollar | Line Endings: Unix (LF) | Ignore rules: - | Guess

$12,345,678.90



Zoom

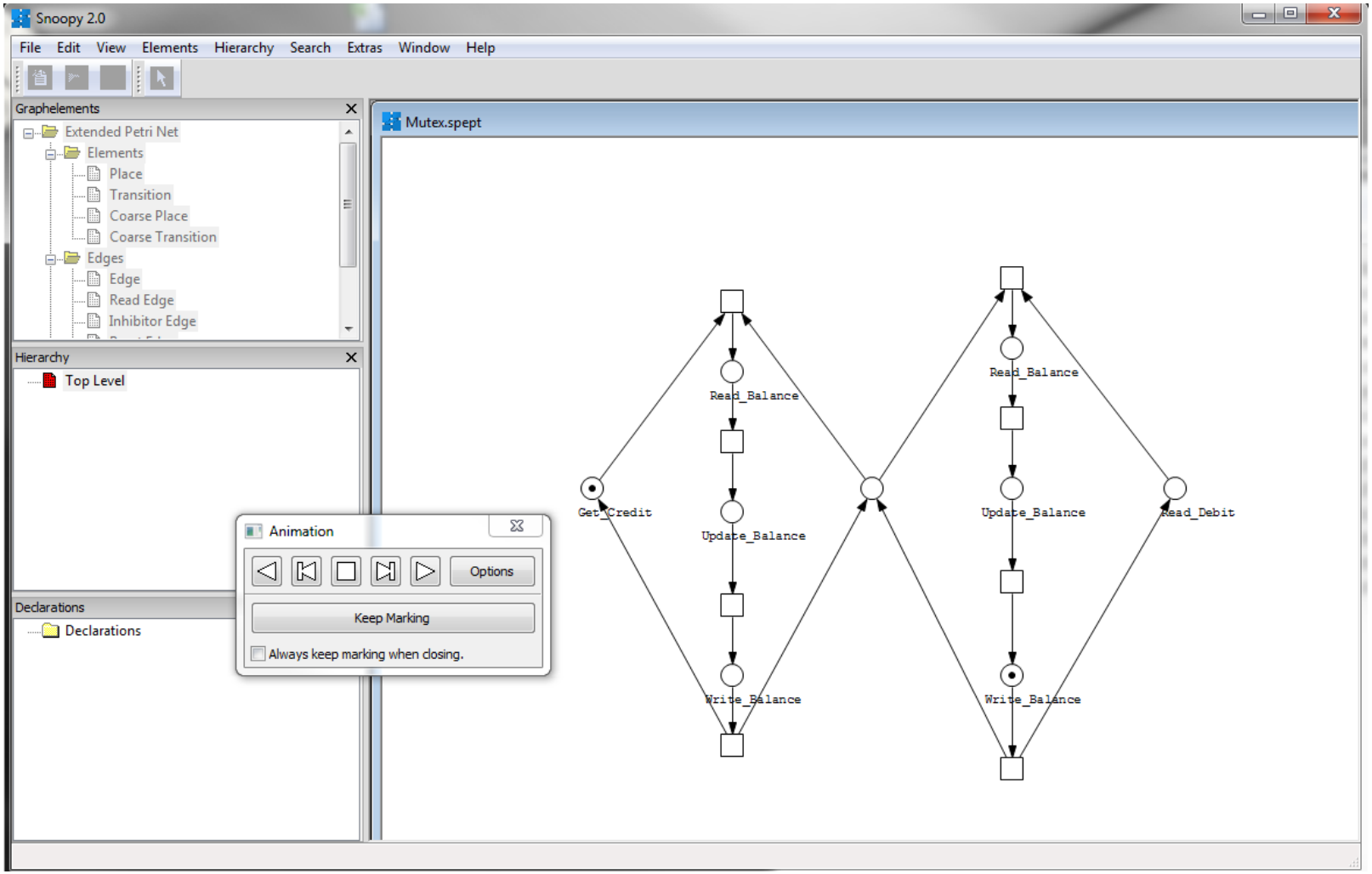
Syntax Diagram | Interpreter | Console | Debugger

4 rules (3 warnings) | 8:1 | Writable | Warnings reported in console | 19M of 259M

Example (3)

- Concepts
  - Concurrency – synchronization– asynchrony
  - Petri nets
- Assignment
  - Develop solutions for simple mutual exclusion and more complex traffic intersection
- Tools
  - Physical simulation
  - Snoopy (Petri net simulator)

Mutex in Snoopy



Discussion

- Finite state acceptor
  → junior level computational biology course
- Grammars/languages
  → senior level compiler course
- Concurrency
  → junior level systems course
- Question: What is the relationship between acceptors and grammars?
  → senior level formal languages course

Evaluation

- An end of term reflections/survey (N=17)
- Key observations - The students reported that the course/topics…
  - *deepened their knowledge and perspective on computer science.*
  - *offered a number of new (to them) concepts and/or improved their understanding of concepts they had already seen.*
  - *helped them develop a better vocabulary for explaining computer science issues.*
- Place in curriculum
  - *The students expressed divided opinions on the ordering of this course with respect to an introductory programming course in computer science.*
  - Room for adoption flexibility

Conclusions

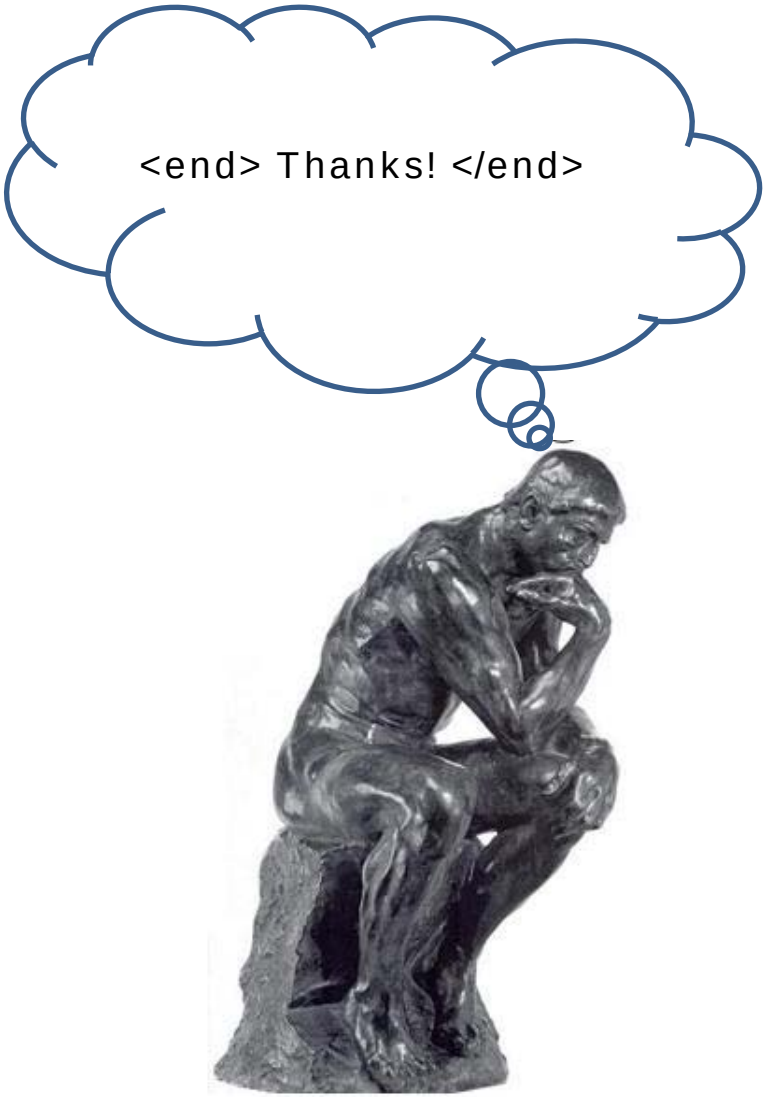
- Concretely engage students in a wide variety of sophisticated computing concepts without the entanglements of programming
  - We need to *show* students that CS is more than programming
  - Tools are available
- CS is about ideas
  - Students need the opportunity to struggle with deep(er) aspects of representation and process
  - Appealing to students for deeper reasons

? is there a set of “right” topics

? possibilities for continuation (at VT) and/or adoption (elsewhere)

? <your question here>


100111010110011...


<end> Thanks! </end>