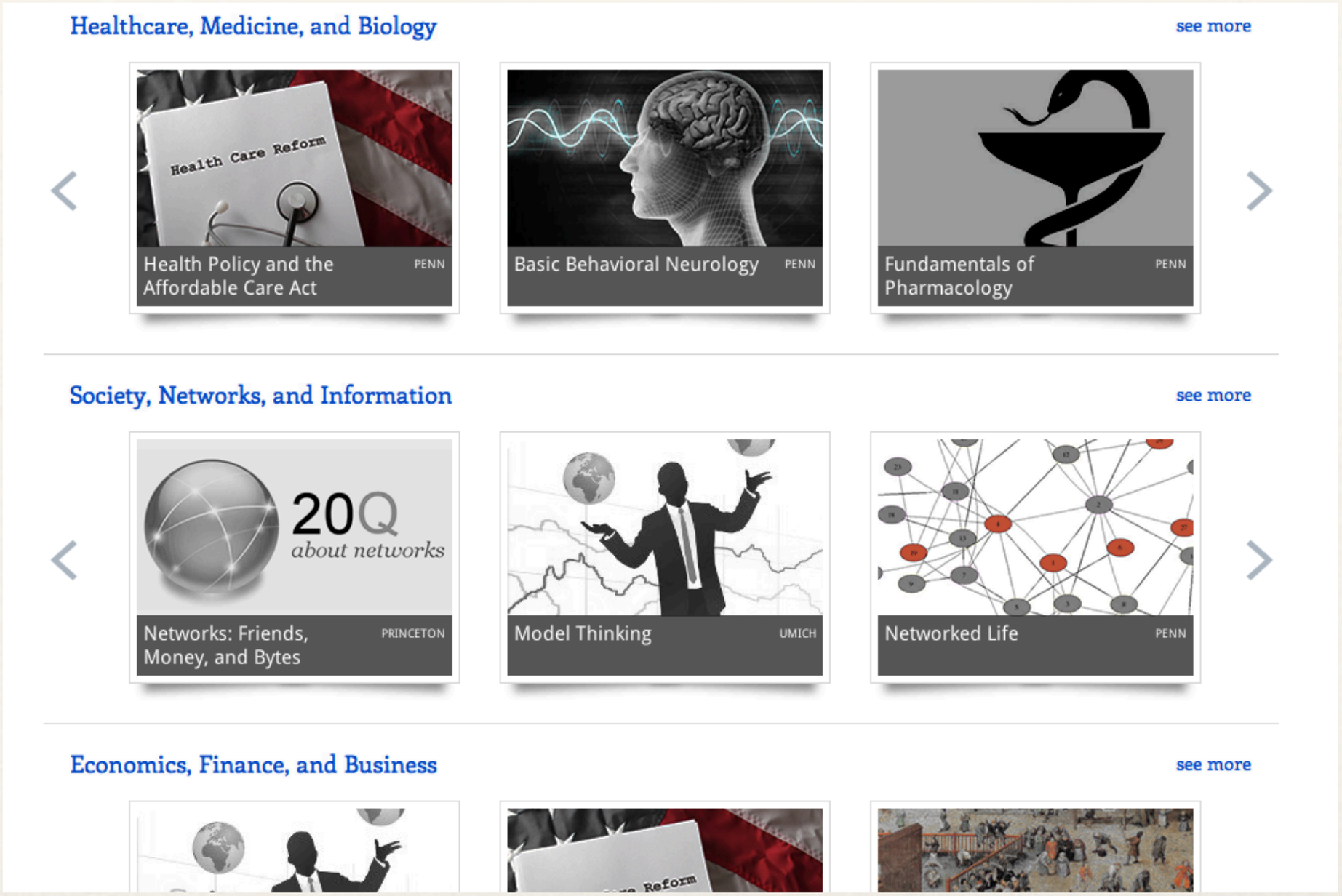
A Canonicalizing Model for Building Programming Tutors

Kelly Rivers and Kenneth R. Koedinger

6/17/2012 - 11th International Conference on Intelligent Tutoring Systems

Time Constraints in Authoring



Proposed Fix: Large Sets of Data

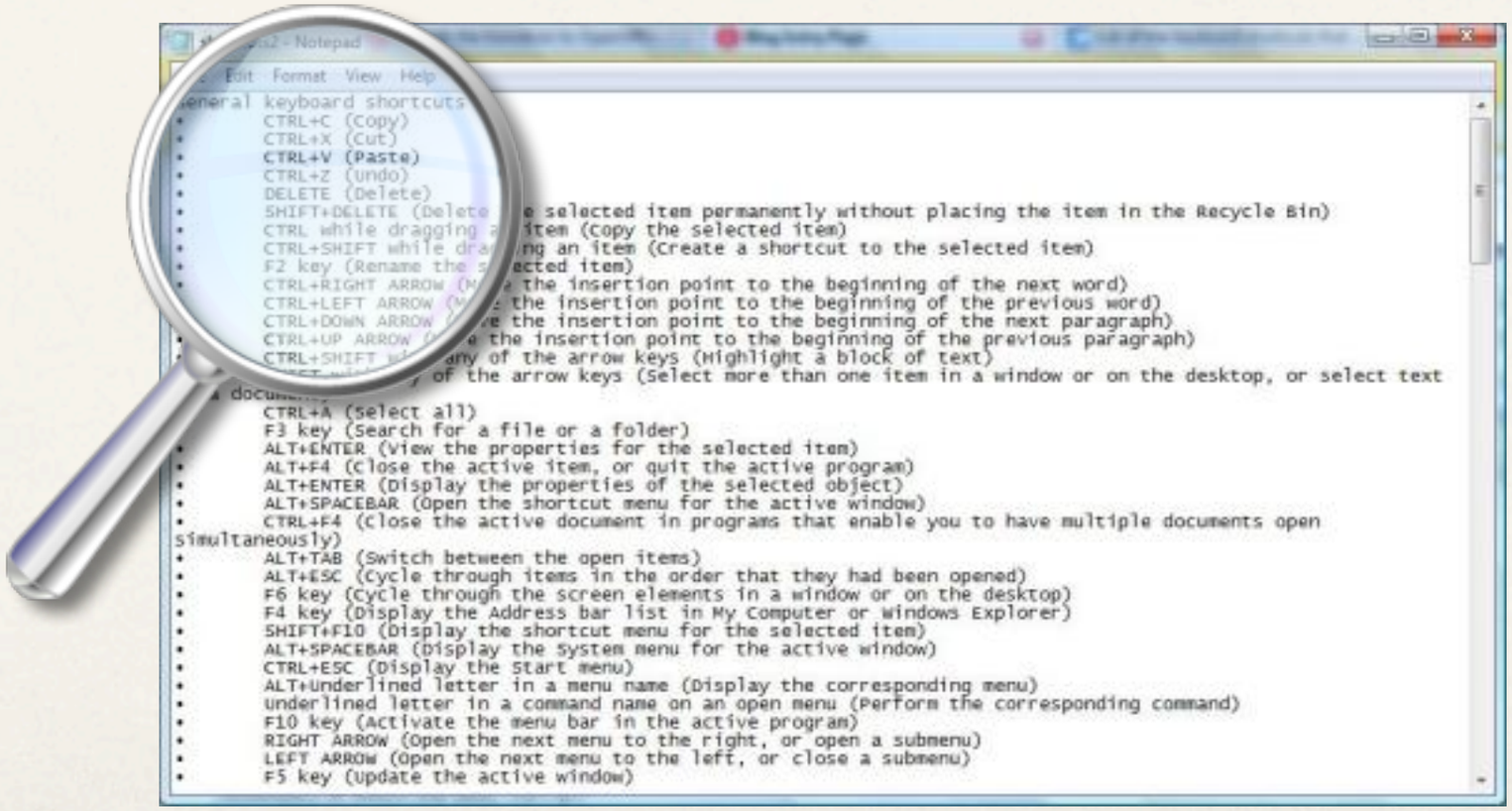
A Complex Problem

Solution

Instead of making more complex tutors, make the student submissions more condensed!

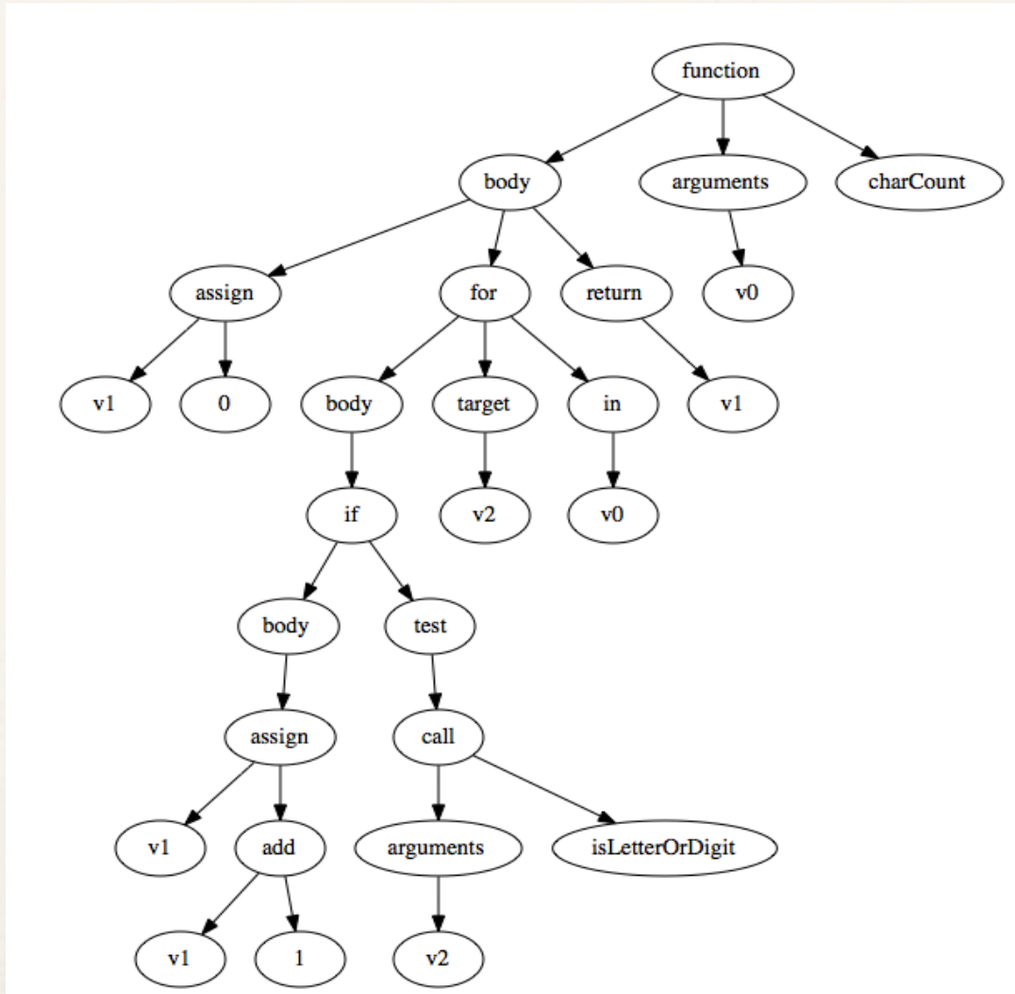
Canonicalization

- A process for converting data that has more than one possible representation into a "standard", "normal", or canonical form



Abstract Syntax Trees (ASTs)

- A tree representation of the abstract syntactic structure of source code written in a programming language



Program Transformations

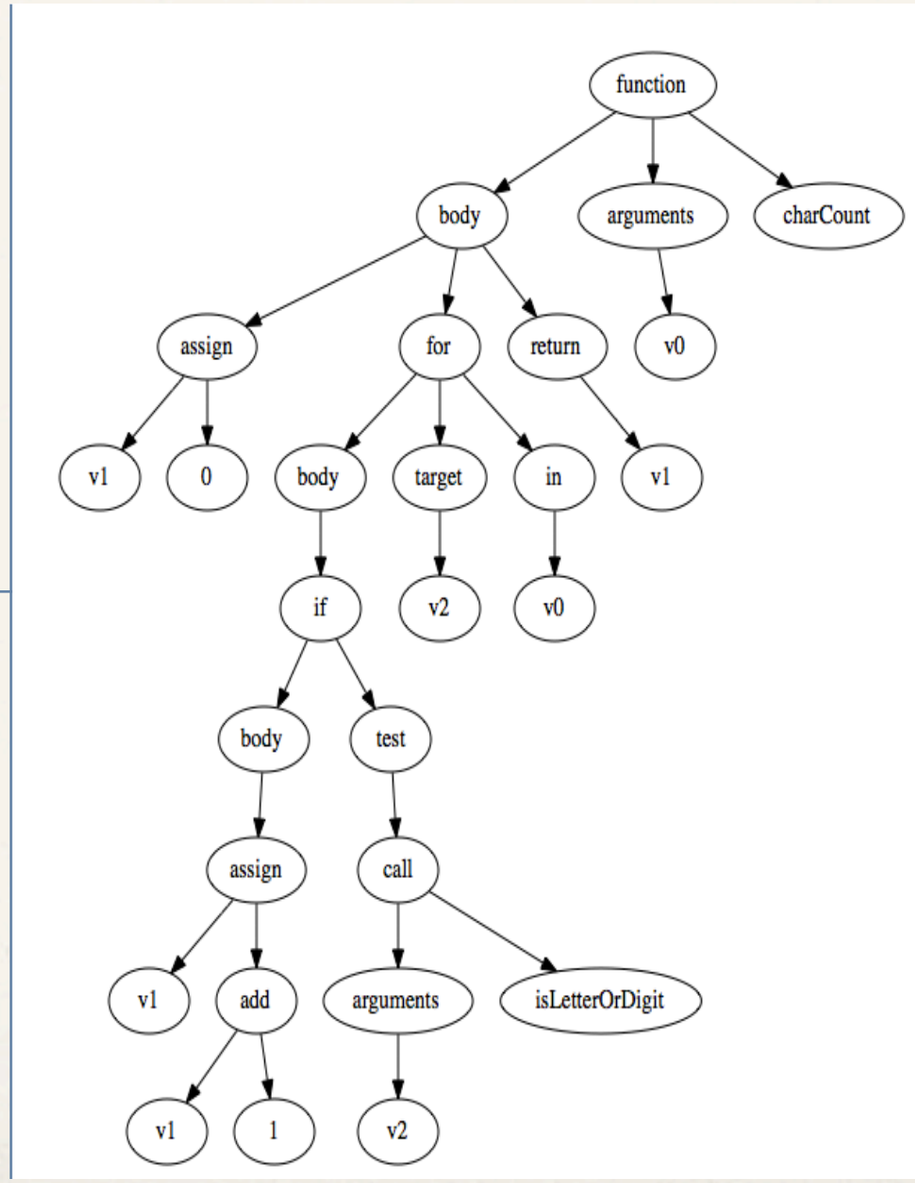
- Functions which are run on computer programs to change their content
- These usually guarantee that the result will be *semantically* equivalent

The Canonicalization Process

Step 1: Student Code -> AST

```
def charCount(text):
    count= 0
    for c in text:
        if isLetterOrDigit(c):
            count +=1
    return count
```

```
def charCount(text):
    charcount = 0
    for char in text:
        if isLetterOrDigit(char): charcount += 1
    return charcount
```



Step 1: Student Code -> AST

```
def charCount(text):
    count= 0
    for c in text:
        if isLetterOrDigit(c):
            count +=1
    return count
```

```
def charCount(text):
    charcount = 0
    for char in text:
        if isLetterOrDigit(char): charcount += 1
    return charcount
```

```
def charCount(v0):
    v1 = 0
    for v2 in v0:
        if isLetterOrDigit(v2):
            v1 += 1
    return v1
```

Step 2: Run Canonicalization Suite on AST

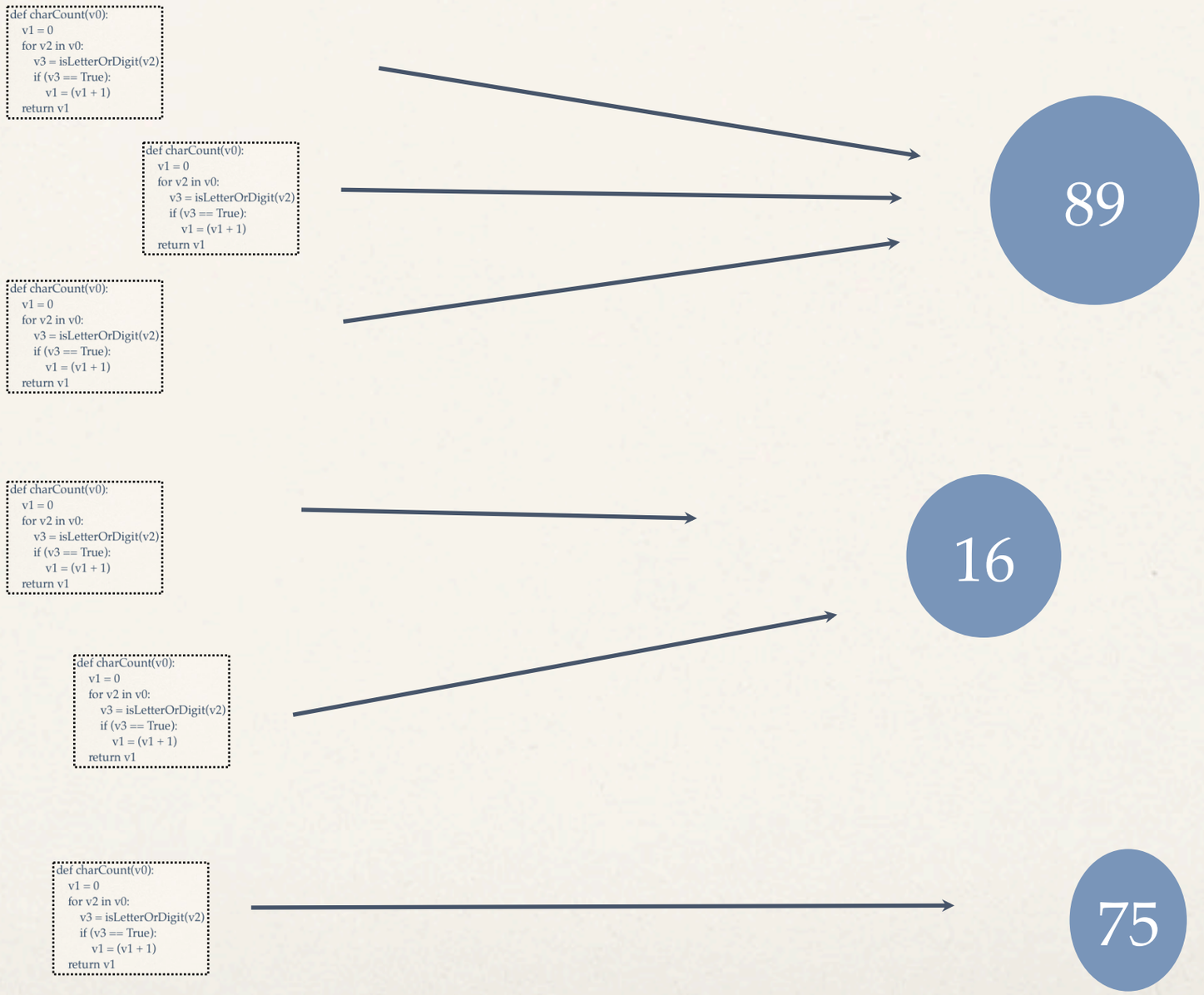
```
def charCount(v0):
    v1 = 0
    for v2 in v0:
        v3 = isLetterOrDigit(v2)
        if (v3 == True):
            v1 = (v1 + 1)
    return v1
```



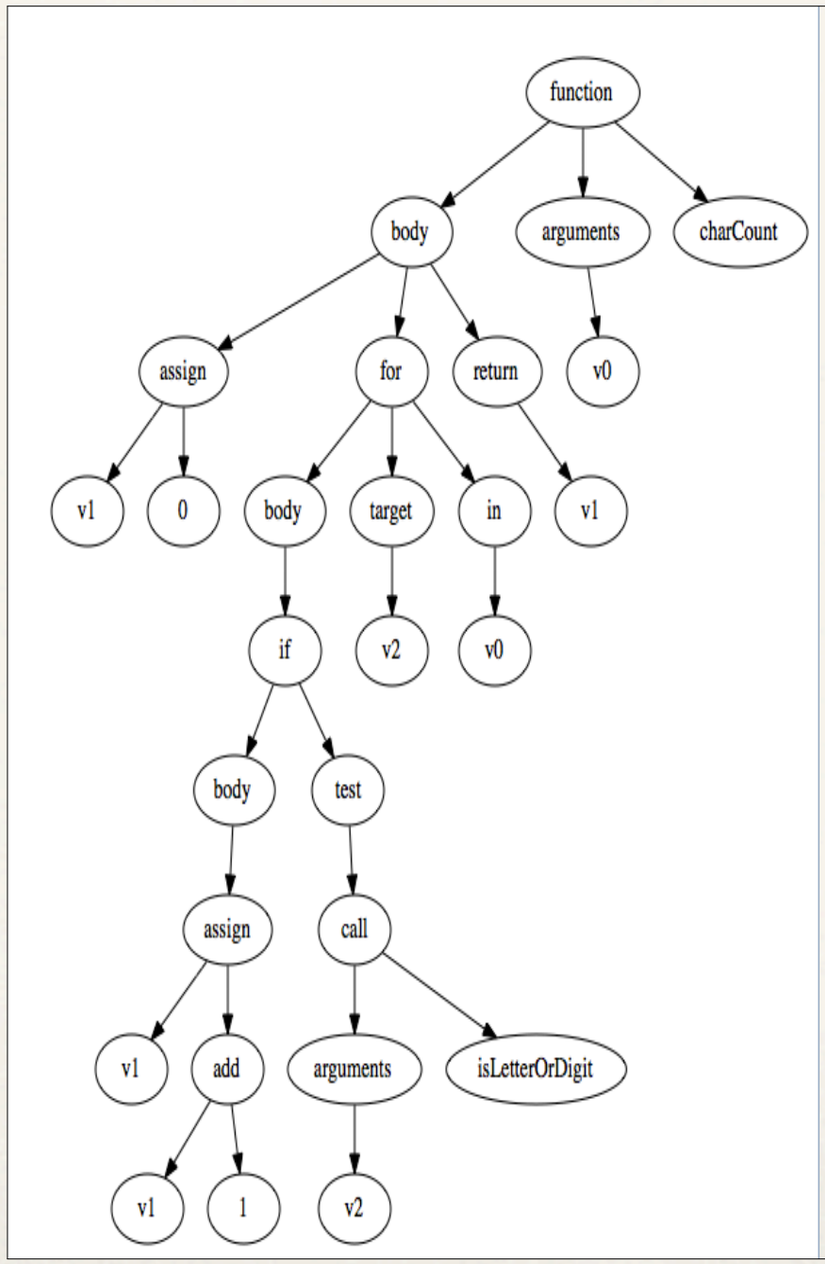
```
def charCount(v0):
    v1 = 0
    for v2 in str(v0):
        if (isLetterOrDigit(v2) == True):
            v1 += 1
    return v1
```

```
def charCount(v0):
    v1 = 0
    for v2 in v0:
        if isLetterOrDigit(v2):
            v1 = (1 + v1)
    return v1
```

Step 3: ASTs -> Canonicalized Models

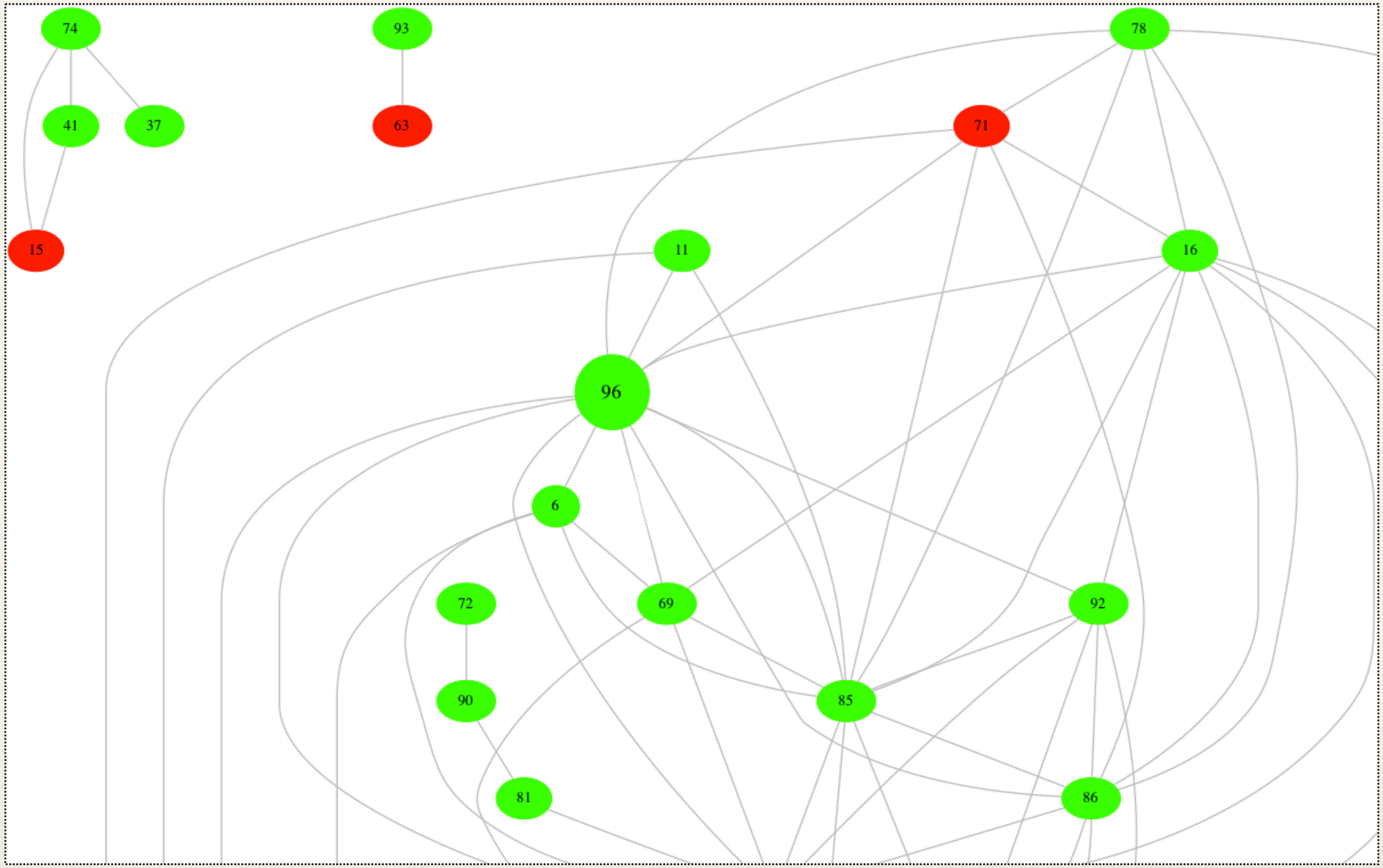


Step 4: Canonicalized Models -> Source Code




```
def charCount(v0):
    v1 = 0
    for v2 in v0:
        if isLetterOrDigit(v2):
            v1 = 1
    return v1
```

Step 5: Cluster Source Code



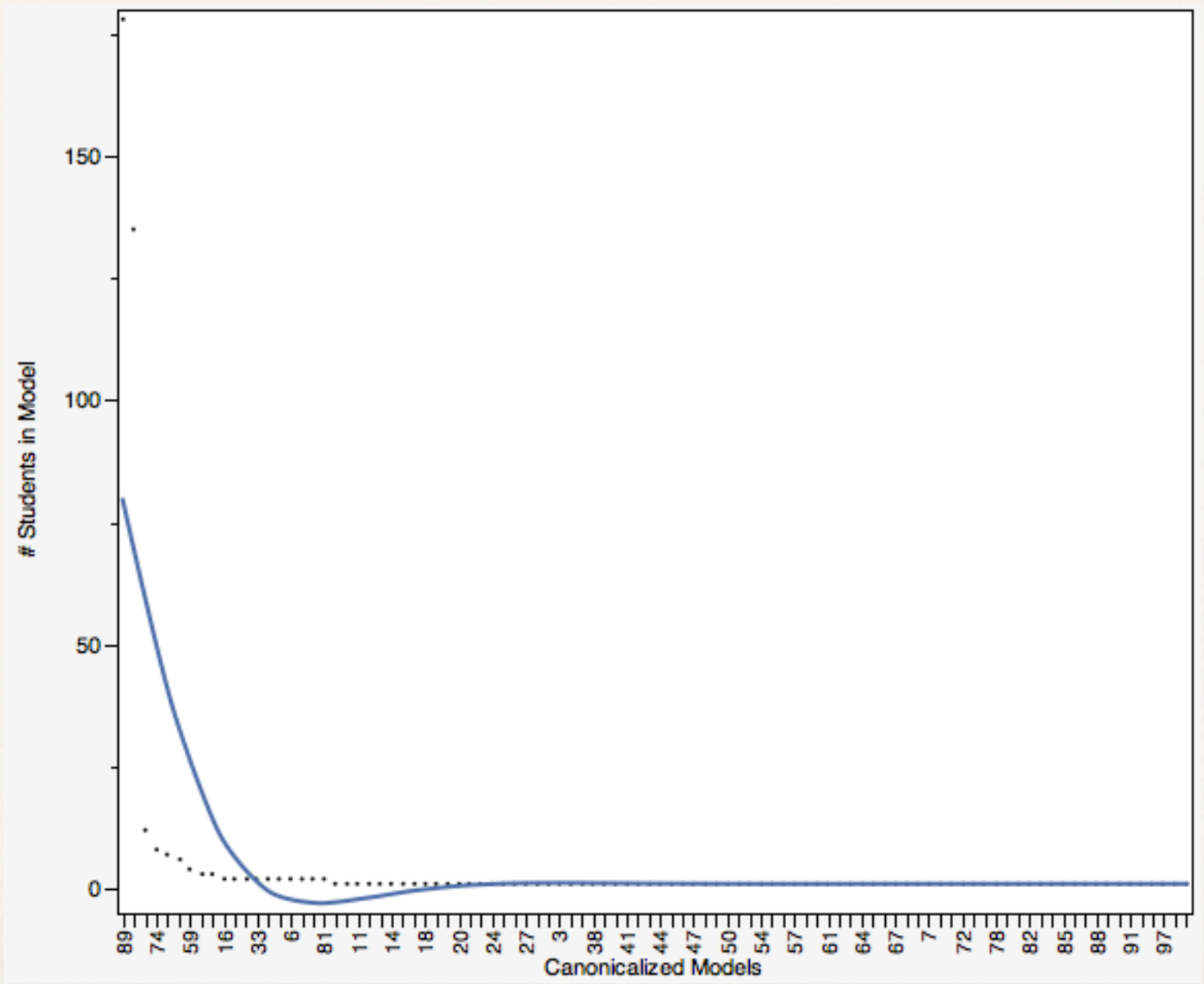
Preliminary Results

Observe changes from Program Text -> AST -> Canonicalized Models

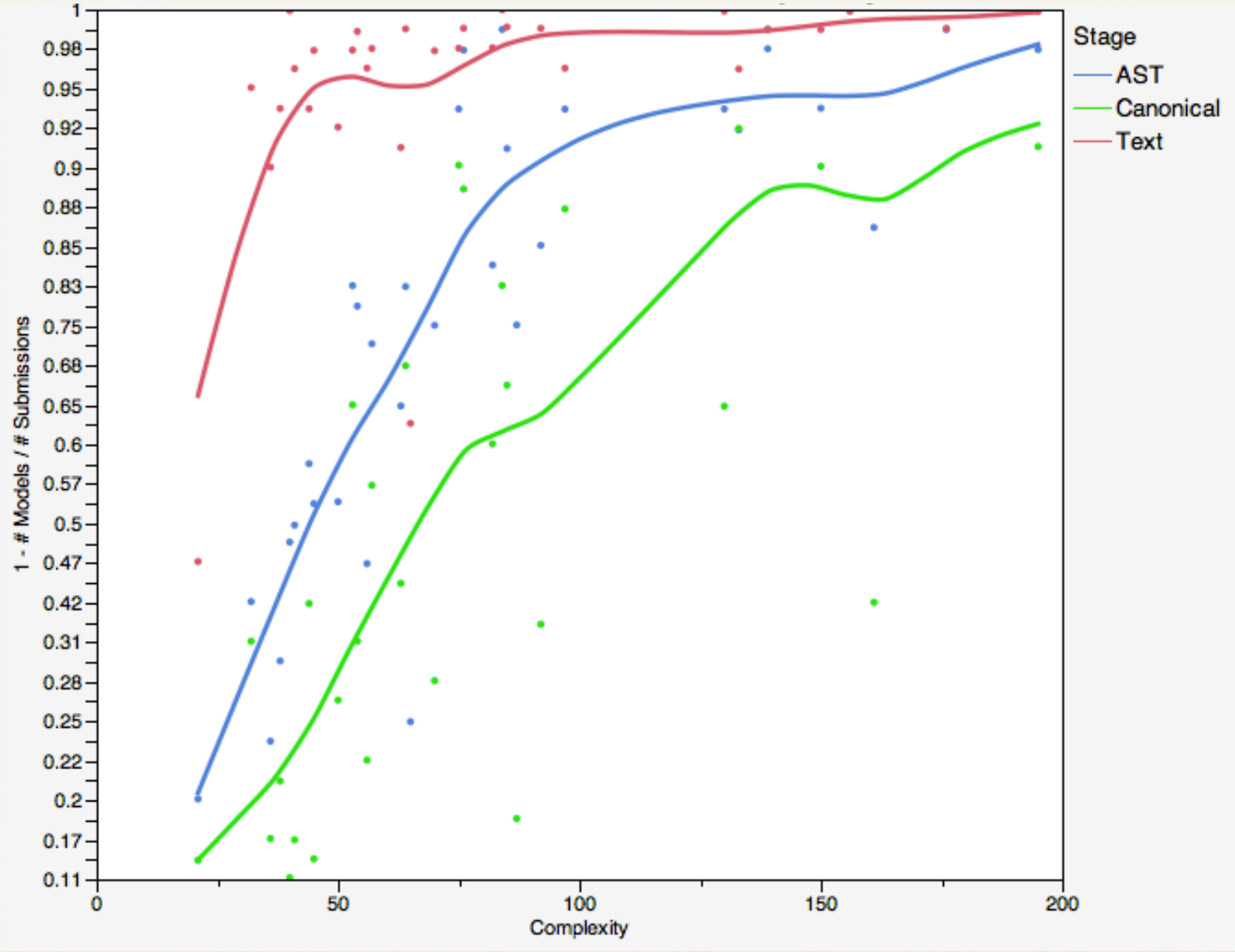
Data Set

- Principles of Computing, Carnegie Mellon University, Spring 2011
- Language: Python (dynamic typing, multiple paradigms)
- Programming Assignments 1-3 (out of 6)
- 33 problems (14 solo, 17 collaborative, 2 bonus)
- Average of 22% incorrect submissions

The Long Tail



Pre-Post Canonicalization



Overall Statistics

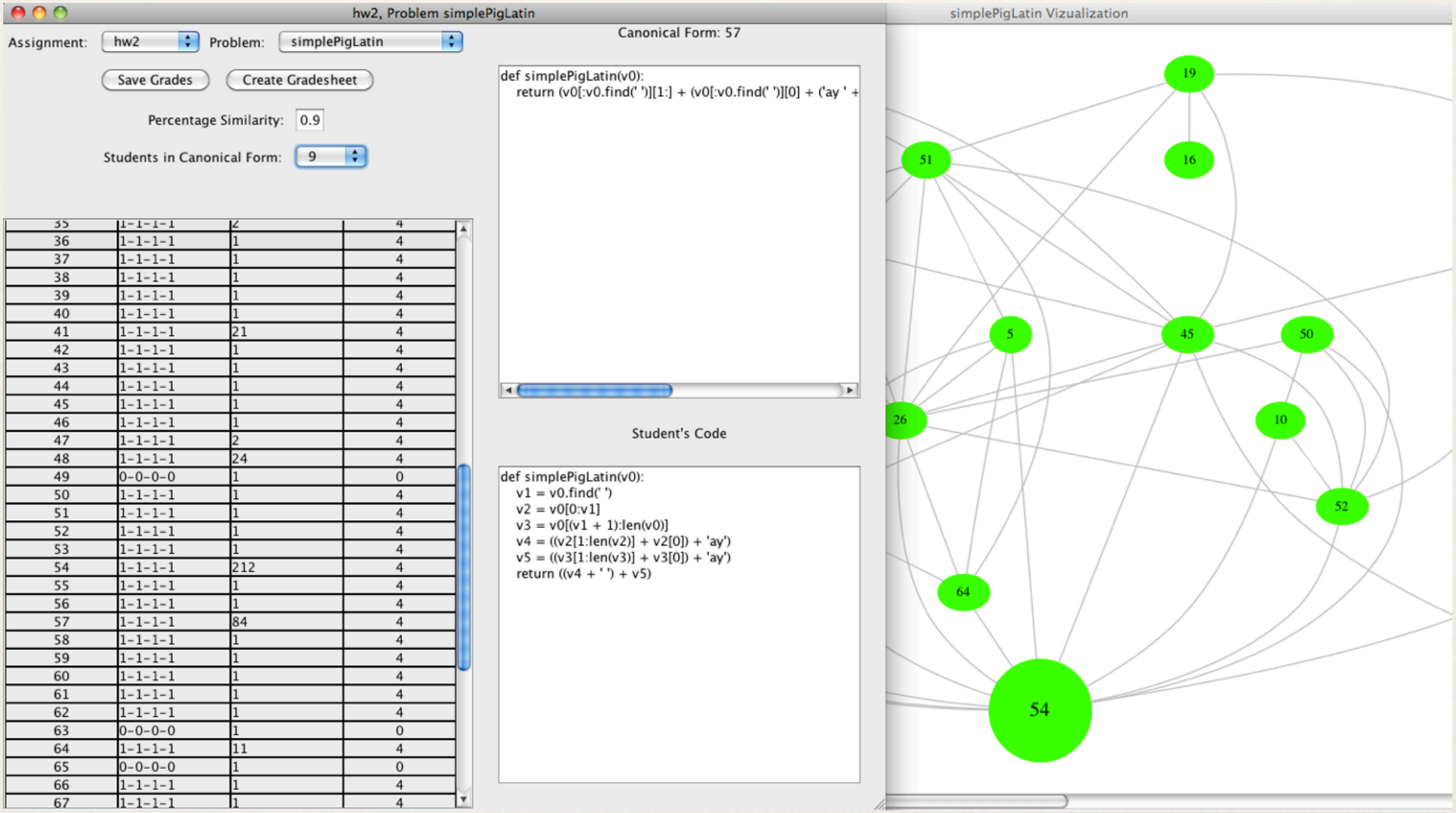
	%Reduction	Δ Max Group Size	# Groups / # Individual
Text	5%*	-	9 / 387
AST	28%**	26**	29 / 263
Canonical Models	48%**	94**	26 / 176

T-test of means, * < 0.01, ** < 0.001

Future Work

Use the simplified, canonicalized models to create tutors per problem

Traditional Hint Construction



35	1-1-1-1	2	4
36	1-1-1-1	1	4
37	1-1-1-1	1	4
38	1-1-1-1	1	4
39	1-1-1-1	1	4
40	1-1-1-1	1	4
41	1-1-1-1	21	4
42	1-1-1-1	1	4
43	1-1-1-1	1	4
44	1-1-1-1	1	4
45	1-1-1-1	1	4
46	1-1-1-1	1	4
47	1-1-1-1	2	4
48	1-1-1-1	24	4
49	0-0-0-0	1	0
50	1-1-1-1	1	4
51	1-1-1-1	1	4
52	1-1-1-1	1	4
53	1-1-1-1	1	4
54	1-1-1-1	212	4
55	1-1-1-1	1	4
56	1-1-1-1	1	4
57	1-1-1-1	84	4
58	1-1-1-1	1	4
59	1-1-1-1	1	4
60	1-1-1-1	1	4
61	1-1-1-1	1	4
62	1-1-1-1	1	4
63	0-0-0-0	1	0
64	1-1-1-1	11	4
65	0-0-0-0	1	0
66	1-1-1-1	1	4
67	1-1-1-1	1	4

Automatic Hint Construction: Model Based

```
def simplePigLatin(v0):
    v1 = v0.find(' ')
    v2 = v0[0:v1]
    v3 = v0[v1::None]
    v4 = (v2[0] + 'ay')
    v5 = (v3[1] + 'ay')
    v6 = ((((v2[1::None] + v4) + ' ') + v3[2::None]) + v5)
    return v6
```

When using splicing, default terms can be successfully left blank!

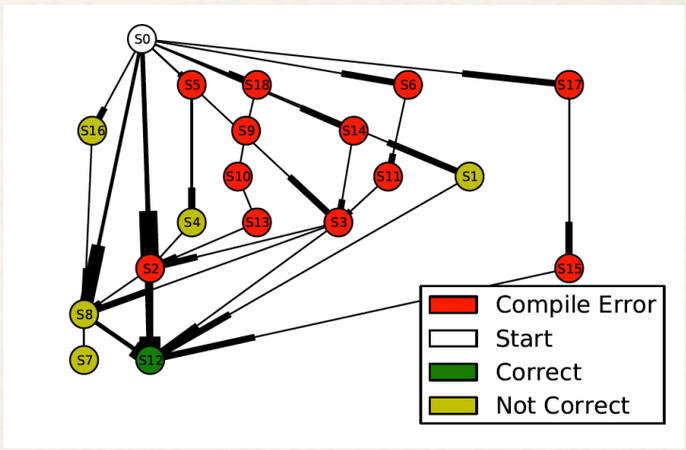
```
def simplePigLatin(v0):
    v1 = v0.find(' ')
    v2 = v0[:v1]
    v3 = v0[v1:]
    v4 = (v2[0] + 'ay')
    v5 = (v3[1] + 'ay')
    v6 = ((((v2[1:] + v4) + ' ') + v3[2:]) + v5)
    return v6
```

Automatic Hint Construction: Data-Driven

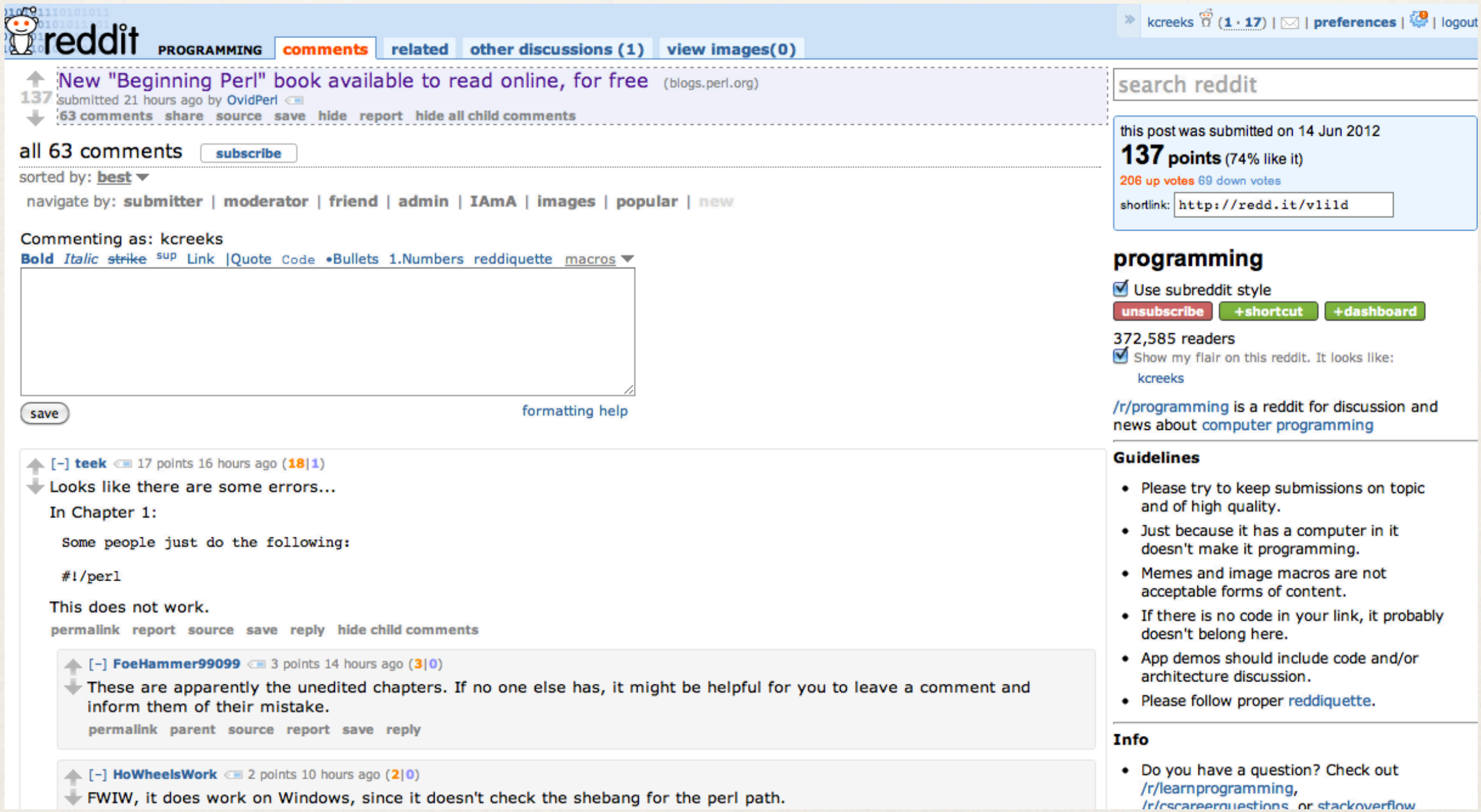
- "Program Representation for Automatic Hint Generation for a Data-Driven Novice Programming Tutor" (Jin et al, ITS 2012)
- "Calculating Probabilistic Distance to Solution in a Complex Problem Solving Domain" (Sudol-DeLyser, Rivers, and Harris, EDM 2012)

```
def charCount(v0):
    v1 = 0
    for v2 in range(len(v0)):
        if isLetterOrDigit(v0[v2]):
            v1 = (1 + v1)
    return v1
```

```
def charCount(v0):
    v1 = 0
    for v2 in range((len(v0) - 1)):
        if isLetterOrDigit(v0[v2]):
            v1 = (1 + v1)
    return v1
```



Automatic Hint Construction: Crowdsourced



Conclusion

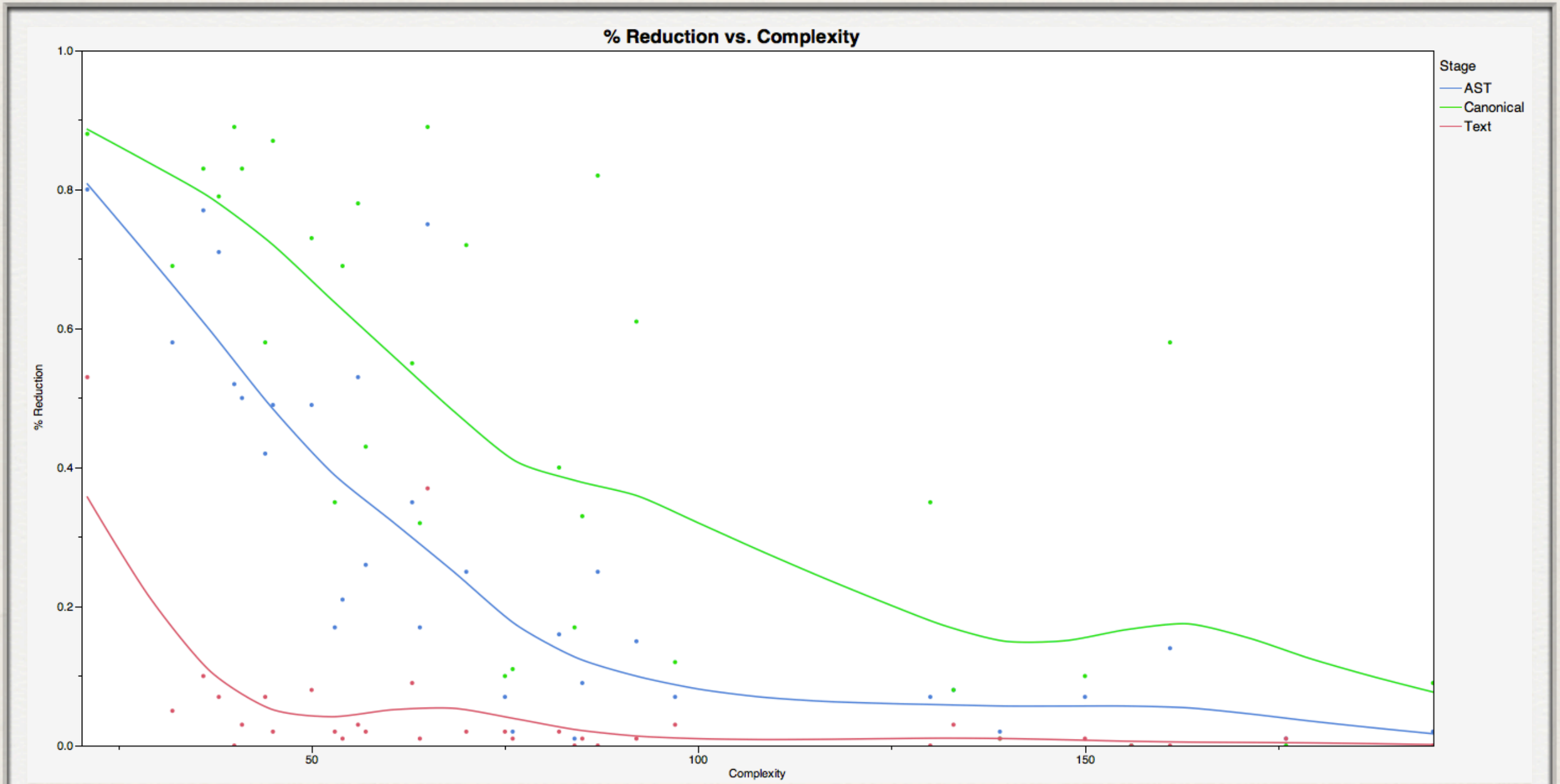
- **What was done:** Simplifying input in complex domains with canonicalization
- **What comes next:** Automatic hint generation (through various techniques)
- **A question for you:** Can canonicalization be extended to further complex domains?

Acknowledgements and Questions

- Ken Koedinger
  - For always providing excellent advice
- David Kosbie and Leigh Ann Sudol-DeLyser
  - For their never-ending help with developing new ideas
- Gregory Dyke
  - For drawing the picture of a tree
- This work was supported in part by Graduate Training Grant awarded to Carnegie Mellon University by the Department of Education (# R305B090023).

Canonicalization Suite

```
functions = [lambda x: helperFolding(x, tree, problems),
                    lambda x: simplify(x, argTypes, {}, [0]),
                    constantFolding,
                    lambda x : copyPropagation(x, tree, problems, {}),
                    lambda x: collapseConditionals(x, tree, problems),
                    constantFolding,
                    lambda x: deadCodeRemoval(x, {}, tree, problems),
                    lambda x : crashableCopyPropagation(x, [], tree, problems),
                    deMorganize, removeUnneccessaryEquals, conditionalRedundancy,
                    combineConditionals,
                    lambda x: collapseConditionals(x, tree, problems),
                    orderCommutativeOperations,
                    constantFolding,
                    cleanupBoolOps, cleanupRanges, cleanupSlices, cleanupTypes,
cleanupInverts, cleanupNegations,
                    deMorganize, orderCommutativeOperations
                   ]
```


% Reduction vs. Complexity
1.0
0.8
0.6
0.4
0.2
0.0
% Reduction
50
100
150
Complexity
Stage
AST
Canonical
Text

We are not breaking Rice's Theorem