Intelligent Programming Tutors

Johan Jeuring
Technologies for learning
Core Lecture 3
Computer Science

Where the Jobs Will Be:

Source: Degree data from National Science Foundation, Science and Engineering Degrees, 1964-2006. Jobs data is reflected as an average per year and is from Bureau of Labor Statistics, November 2007 Monthly Labor
Programming in our education

- 10 programming intensive courses attracted ~2000 students in 2017
- ~10,000 lab submission
- ~40 TAs
Learning programming

❖ When learning how to program, you need help.
❖ Beginner programmers experience a lot of problems.
❖ Individual support is difficult to arrange.
❖ Assessment takes a lot of time.
❖ Feedback often comes too late.
❖ Can intelligent programming tutors (IPTs) help?
Programming tutors

- Offer or link to learning materials
- Offer multiple programming tasks
- Analyse not just a final product (automatic assessment), but also steps students take when programming
- Give feedback on student programs
- Give hints about how to proceed
History

AUTOMATIC GRADING OF STUDENTS’
ALGOL PROGRAMMING
PETERNAUR

Abstract.
The report describes an experiment on automatic grading of student algorithms, using an ALGOL compiler. The experiment is based on an evaluation of the efficiency and logical completeness of the algorithms, not on their formal correctness, which is supposed to be checked in advance by the individual student. The technique used is to embed the student algorithms within a larger grading program structure, which supplies test cases and performs checks and evaluation. The complete text of such a grading program is given. The experience gained through the experiment, and suggestions for further developments, are discussed.

472
ANDERSON, CONRAD, AND CORBETT

<table>
<thead>
<tr>
<th>TABLE 1 (Continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6. (defun create-list (num)</td>
</tr>
<tr>
<td>(let (count num)</td>
</tr>
<tr>
<td>(numlist (list num)))</td>
</tr>
<tr>
<td>(loop</td>
</tr>
<tr>
<td>(cond ((equal count 1) (return numlist)))</td>
</tr>
<tr>
<td>(&lt; LOOP-ACTIONS2 &gt;)))</td>
</tr>
<tr>
<td>In numeric iteration the first thing you should do after the exit test is update the counter variable. You can use the function SETQ to do this.</td>
</tr>
</tbody>
</table>

7. (defun create-list (num) |
| (let (count num) |
| (numlist (list num))) |
| (loop |
| (cond ((equal count 1) (return numlist))) |
| (setq count (+ count 1)) |
| (cons numlist))) |
| You’ve got the right idea, since you’ll need to use the function “cons”, but remember you are trying to insert the new value of the counter into the list. What LISP function allows you to assign the results of this function call to a local variable?

8. (defun create-list (num) |
| (let (count num) |
| (numlist (list num))) |
| (loop |
| (cond ((equal count 1) (return numlist))) |
| (setq count (+ count 1)) |
| (setq numlist (list))) |
| If you LIST together arguments like 7 and 8 you get (7 8)). What you want is (7 8). Try another function that combines things together.

9. (defun create-list (num) |
| (let (count num) |
| (numlist (list num))) |
| (loop |
| (cond ((equal count 1) (return numlist))) |
| (setq count (+ count 1)) |
| (setq numlist (cons count numlist)))) |
| --- YOU ARE DONE. TYPE NEXT TO GO ON AFTER --- |
| --- TESTING THE FUNCTION YOU HAVE DEFINED --- |
| THE LISP WINDOW |

⇒ (create-list 10)
(1 2 3 4 5 6 7 8 9 10)
What do intelligent programming tutors do?
Hour of Code
FIT Java Tutor

```java
public class Recursion {
    public static void main(String[] args) {
        Recursion recursion = new Recursion();
        int result = recursion.recursive(5);
        System.out.println(result);
    }

    public int recursive(int num) {
        return recursive(num - 1) + num;
    }
}
```

Exception in thread "main" java.lang.StackOverflowError
    at Recursion.recursive(Recursion.java:9)
    at Recursion.recursive(Recursion.java:9)
    at Recursion.recursive(Recursion.java:9)
    at Recursion.recursive(Recursion.java:9)
    at Recursion.recursive(Recursion.java:9)

Feedback: request hide
The system has determined a certain similarity between your program and the program shown above. Compare the two programs and modify your program if necessary.
Please rate the feedback (helpful, fair, not helpful):
CheckPoint

Complete the function below to multiply the two numbers a and b without using the normal multiplication operator * in your solution. You should do this in the smallest number of steps you can manage.

```c
int multiply (int a, int b) {
    int total = 0;
    for (int i = 0; i < a; i++) {
        total += b;
    }
    return total;
}
```

Comments:

- Test 1: Compile answer (0.0 out of 0)
- Test 2: Test answer (2.0 out of 2)
  
  All tests completed successfully!
- Test 3: Check for use of * (2.0 out of 2)
- Test 4: Efficiency check (3.9 out of 6)

409 * 351 solved in 409 steps, but it could have been done in 351 steps
351 * 409 solved in 351 steps, but there is also a solution which only uses 9 steps
Ask-Elle

Write a function that reverses a list: myreverse :: [a] -> [a]. For example:

Data.List> myreverse "A man, a plan, a canal, panama!"
"A man a plan a canal apanam A"

Data.List> myreverse [1,2,3,4]
[4,3,2,1]
What kind of help? (Narciss)

<table>
<thead>
<tr>
<th>KTC</th>
<th>Knowledge about task constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR</td>
<td>Hints on task requirements</td>
</tr>
<tr>
<td>TPR</td>
<td>Hints on task-processing rules</td>
</tr>
<tr>
<td>KC</td>
<td>Knowledge about concepts</td>
</tr>
<tr>
<td>EXP</td>
<td>Explanations on subject matter</td>
</tr>
<tr>
<td>EXA</td>
<td>Examples illustrating concepts</td>
</tr>
<tr>
<td>KM</td>
<td>Knowledge about mistakes</td>
</tr>
<tr>
<td></td>
<td>( ○ basic or ● detailed)</td>
</tr>
<tr>
<td>TF</td>
<td>Test failures</td>
</tr>
<tr>
<td>CE</td>
<td>Compiler errors</td>
</tr>
<tr>
<td>SE</td>
<td>Solution errors</td>
</tr>
<tr>
<td>SI</td>
<td>Style issues</td>
</tr>
<tr>
<td>PI</td>
<td>Performance issues</td>
</tr>
<tr>
<td>KH</td>
<td>Knowledge about how to proceed</td>
</tr>
<tr>
<td></td>
<td>( ○ hint, ● solution or ● both)</td>
</tr>
<tr>
<td>EC</td>
<td>Bug-related hints for error correction</td>
</tr>
<tr>
<td>TPS</td>
<td>Task-processing steps</td>
</tr>
<tr>
<td>IM</td>
<td>Improvement hints</td>
</tr>
<tr>
<td>KMC</td>
<td>Knowledge about meta-cognition</td>
</tr>
</tbody>
</table>
# A review

<table>
<thead>
<tr>
<th>Name</th>
<th>KTC</th>
<th>KC</th>
<th>KM</th>
<th>KH</th>
<th>KMC</th>
<th>Name</th>
<th>KTC</th>
<th>KC</th>
<th>KM</th>
<th>KH</th>
<th>KMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT Progr. Tutor (APT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Jackson00)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADAPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Java Sensei</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Ala-mutka04)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Jin12)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Alleman91)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Jin14)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AnalyseC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IITS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>APPOS2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Keuning14)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ask-Elle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Kim98)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASSYST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Koh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>At(x)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LAURA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoGrader</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Lazar14)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoLEP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LISP tutor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoStyle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Ludwig</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoTeach</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Mandal07)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MarmoSet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bridge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MENO-HI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C-tutor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Naur64)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camus</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Online Judge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ceilidh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PASS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Chang60)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PATTIE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checkpoint</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Pex4Fun</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHIRON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PHP ITS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COALA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PRAM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Hunt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ProgramCritic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CourseMarker/-Master</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ProgTest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSTutor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ProPAT_deBUG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cteacher</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ProPL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A review: results

- 101 tools that support students while programming
- Feedback mainly about identifying mistakes
- Few tools give hints on how to proceed
What is a good program?

- Functionally correct
- Satisfies time and space efficiency requirements
- Quality code, good style (many aspects)
Limitations

❖ Rice’s theorem: All non-trivial, semantic properties of programs are undecidable
❖ So we cannot determine correctness and performance properties for all programs
❖ Some code quality aspects can always be determined
❖ Is undecidability a big problem for student programs?
How do intelligent programming tutors do what they do?
Giving feedback

- Testing
- Proving
- Analyzing
- Comparing
Testing

- Unit testing — $\text{isPrime } 15 = \text{False}$
- Property testing — $y = \text{sort } x \implies \text{isAscending } y$
- Requires running a program
- Also called dynamic analysis
- Uses the “operational semantics” of a program
Testing — evaluation

- Huge field: thousands of tools (besides ITSs)
- Well suited to point out errors (true negatives)
- Can point to properties that are not satisfied
- Confidence in correctness of programs (not complete though)
- Risk for false positives
Proving

sort :: \( \forall x . x \rightarrow \{ y | \text{isAscending } y \land y \text{ isPermutation } x \} \)

- Dependently typed programming
- Automated correctness proving (SMT solvers)
- Uses the “denotational semantics” of a program
Proving — evaluation

❖ Since the Intel chip disaster a big field
❖ Hard to achieve automatically
❖ Returns true positives (property-wise), risk for false negatives
❖ Clear scaling issues (but for beginners maybe less problematic?)
❖ Might be used to focus on program parts that are not automatically verifiable
❖ Not aware of any usage for IPTs
Analyzing

\[ \text{sort} :: (\text{Ord } x) \Rightarrow \forall x. [x] \rightarrow [x] \]

- For determining typing, dependency, time complexity or space complexity properties
- Also called static analysis, abstract interpretation
- Uses typing rules, or space properties, or…
- Is an approximation (simplification)
Analyzing — evaluation

❖ Usually returns a simpler result than a program (types, memory usage, etc)
❖ Huge field, in particular in the context of compilers
❖ Comparing is an instance, but deserves separate treatment
Comparing

parseTable = map words

parseTable [] = []
parseTable (x : xs) = words x : parseTable xs

- against expert solutions or previous solutions from other students
- rewriting student solutions to normal forms (name-independent, application-independent, …)
Comparing — evaluation

- True true positives when comparing against expert solutions
- Comparing against previous student solutions has advantages (search space wise) and disadvantages (suboptimal paths)
- Hints are better when using multiple expert solutions
- Easier to catch quality issues compared to testing and proving
- Not often studied outside the field of IPTs, only for plagiarism detection and code duplication
Giving hints

- About incomplete programs: comparing, proving?
- About complete programs: comparing, analyzing
Ask-Elle
Write a function that reverses a list: `myreverse :: [a] -> [a]`. For example:

```
Data.List> myreverse "A man, a plan, a canal, Panama!"
"A man, a plan, a canal, Panama!"
```
```
Data.List> myreverse [1,2,3,4]
[4,3,2,1]
```

```haskell
myreverse = ?
where
  reverse' acc ? = ?
```

You can follow one of the following strategies:

- Introduce a helper function that uses an accumulating parameter.

**Hint 1**

Introduce the constructor pattern `[]`.

**Hint 2**

Refine the current term to

```
myreverse = ?
where
  reverse' acc [] = ?
```
Characteristics

❖ Type and dependency analysis to detect errors, using an existing compiler for beginners (analysing)
❖ Property testing to detect errors (testing)
❖ Expert solutions to guide a student (comparing)
❖ Normalisations to rewrite student programs
No diagnosis?

Your solution passed all tests! However, it is different from the solutions we have constructed.

Opportunity: proving to accept programs that cannot be compared
Why?

- Program correct, but implements a different solution
- Program correct, similar to a model solution, but Ask-Elle’s program transformations do not recognise this
- Program is incorrect, but property checking finds no counterexamples, because:
  - program is undefined in many places
  - the property has a strong precondition
More diagnoses?

We need numbers!
Experiment

❖ 116 2nd year computer science students learning functional programming
❖ Use Ask-Elle in a lab session (and at home)
❖ 3466 diagnose requests (interactions) received by our server
❖ Subsequent interactions on the same exercises are collected in attempts
Classification categories

- Compiler error
- Matches model solution (Model)
- Counterexample

Tests passed  Undecided
Discarded
## The diagnoses

<table>
<thead>
<tr>
<th></th>
<th>Interactions</th>
<th>Attempts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler error</td>
<td>1920 (55.4%)</td>
<td>142 (21.8%)</td>
</tr>
<tr>
<td>Model</td>
<td>754 (21.8%)</td>
<td>221 (33.9%)</td>
</tr>
<tr>
<td>Counterexample</td>
<td>201 (5.8%)</td>
<td>33 (5.1%)</td>
</tr>
<tr>
<td>Tests passed</td>
<td>436 (12.6%)</td>
<td>235 (36.0%)</td>
</tr>
<tr>
<td>Discarded</td>
<td>155 (4.5%)</td>
<td>21 (3.2%)</td>
</tr>
<tr>
<td>Total</td>
<td>3466</td>
<td>652</td>
</tr>
</tbody>
</table>
NClassified = |Model| + |Error| + |Counter| 
Classified = NClassified / |Total| 

Classified: 82.9% 
Attempts: 60.7%
Program normalisations

Move programs from Tests passed to Model

We analysed the Tests passed category to find out which program normalisations would help

<table>
<thead>
<tr>
<th>Remove type signatures</th>
<th>More Prelude functions</th>
<th>Beta reduction for function</th>
<th>Eta conversion</th>
<th>Alpha renaming bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>37</td>
<td>39</td>
<td>75</td>
<td>48</td>
</tr>
</tbody>
</table>

...
## Results

<table>
<thead>
<tr>
<th>Category</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler error</td>
<td>1920 (55.4%)</td>
</tr>
<tr>
<td>Model</td>
<td>1095 (31.6%)</td>
</tr>
<tr>
<td>Counterexample</td>
<td>206 (5.9%)</td>
</tr>
<tr>
<td>Tests passed</td>
<td>87 (2.5%)</td>
</tr>
<tr>
<td>Discarded</td>
<td>158 (4.6%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>3466</strong></td>
</tr>
</tbody>
</table>
Conclusions

❖ IPTs appear in all kinds of forms, for all kinds of programming languages
❖ Few IPTs give feedback and hints on intermediate student products
❖ There are four main techniques for analyzing student programs, of which comparing seems most powerful
❖ Ask-Elle is an IPT for Haskell combining many techniques