Learning (through) Recursion:  
A Multidimensional Analysis of the Competences Achieved by CS1 Students

Claudio Mirolo

Dept. of Mathematics and Computer Science, University of Udine, Italy

ACM-SIGCSE Conference on Innovation and Technology in Computer Science Education, Bilkent, Ankara 2010
Talk subject

What this talk is about:

- Broad test addressing various aspects related to the learning of recursion
- Administered after about 1/3 of the introductory programming course (CS1)
- Independent of the exams
Outline

1. Introduction
2. Instrument
   - questionnaire – part I
   - questionnaire – part II
   - administration
3. Analysis and results
   - computation model
   - relations and structures
   - language and abstraction
4. Discussion
   - summary
   - conclusions
Outline

1. Introduction
2. Instrument
   - questionnaire – part I
   - questionnaire – part II
   - administration
3. Analysis and results
   - computation model
   - relations and structures
   - language and abstraction
4. Discussion
   - summary
   - conclusions
Keywords in the title:

- **Recursion**
- **Multidimensional**

CS1 / Introductory Programming: *functional-first*

Wider scope: learning *through* recursion

Recursion is not only important in itself, but also because it requires the students to think at a more abstract level
Recursion is not only important in itself, but also because it requires the students to think at a more abstract level.
Keywords in the title:

- Recursion

CS1 / Introductory Programming: *functional-first*

- Multidimensional

Wider scope: learning *through* recursion

Recursion is not only important in itself, but also because it requires the students to think at a more abstract level
Keywords in the title:

- Recursion

  CS1 / Introductory Programming: *functional-first*

- Multidimensional

  Wider scope: learning *through* recursion

Recursion is not only important in itself, but also because it requires the students to think at a more abstract level
Indeed, what do we want our students to achieve?

Often a major concern is the *computation model*, which may be helpful to understand:

- that a recursive procedure can *effectively* be computed
- when a recursive definition is *well-founded*

Interesting, in this respect, the *mental models* analysis of Götschi and colleagues (2003, 2006)
Indeed, what do we want our students to achieve?

Often a major concern is the *computation model*,

which may be helpful to understand:

- that a recursive procedure can *effectively* be computed
- when a recursive definition is *well-founded*

Interesting, in this respect, the *mental models* analysis of Götschi and colleagues (2003, 2006)
Indeed, what do we want our students to achieve?

Often a major concern is the computation model, which may be helpful to understand:

- that a recursive procedure can effectively be computed
- when a recursive definition is well-founded

Interesting, in this respect, the mental models analysis of Götschi and colleagues (2003, 2006)
However, thinking in terms of computation model doesn’t seem to be of much help to approach a problem recursively.

I was curious to investigate:

- students’ fluency with the mechanics of computation in the specific context of our undergraduate CS program
- how this ability correlates to other higher-level achievements

So I have tried to design an instrument to this aim.
However, thinking in terms of computation model doesn’t seem to be of much help to approach a problem recursively.

I was curious to investigate:

- students’ fluency with the mechanics of computation in the specific context of our undergraduate CS program
- how this ability correlates to other higher-level achievements

So I have tried to design an instrument to this aim.
However, thinking in terms of computation model doesn’t seem to be of much help to approach a problem recursively.

I was curious to investigate:

- students’ fluency with the mechanics of computation in the specific context of our undergraduate CS program
- how this ability correlates to other higher-level achievements

So I have tried to design an instrument to this aim.
Learning dimensions

Multidimensional range of programming competences:

i. Language *notation* and meaning

ii. *Notional machine* computation model

iii. Relations in/with the problem domain (basic problem solving skills)

iv. Data and program *structures* (basic organizational skills)

v. Abstraction and management of complexity (higher-level organizational skills)
Learning dimensions

Multidimensional range of programming competences:

i. Language *notation* and meaning

ii. *Notional machine* computation model

iii. Relations in/with the problem domain
    (basic problem solving skills)

iv. Data and program *structures*
    (basic organizational skills)

v. Abstraction and management of complexity
    (higher-level organizational skills)
Learning dimensions

Multidimensional range of programming competences:

i. Language *notation* and meaning

ii. *Notional machine* computation model

iii. Relations in/with the problem domain (basic problem solving skills)

iv. Data and program *structures* (basic organizational skills)

v. Abstraction and management of complexity (higher-level organizational skills)
Multidimensional range of programming competences:

i. Language *notation* and meaning

ii. *Notional machine* computation model

iii. Relations in/with the problem domain (basic problem solving skills)

iv. Data and program *structures* (basic organizational skills)

v. Abstraction and management of complexity (higher-level organizational skills)
Learning dimensions

Multidimensional range of programming competences:

1. Language *notation* and meaning
2. *Notional machine* computation model
3. Relations in/with the problem domain (basic problem solving skills)
4. Data and program *structures* (basic organizational skills)
5. Abstraction and management of complexity (higher-level organizational skills)
Learning dimensions

Multidimensional range of programming competences:

i. Language *notation* and meaning

ii. *Notional machine* computation model

iii. Relations in/with the problem domain (basic problem solving skills)

iv. Data and program *structures* (basic organizational skills)

v. Abstraction and management of complexity (higher-level organizational skills)
Not an easy task

Of course, the challenge is to conceive appropriate questions:

- to separate as accurately as possible the aspects pertaining to each learning dimension
- to limit the application of stereotypical solution patterns
- to attain a balance between “exam anxiety” and weakening of motivation

The present work is (just) an attempt in this direction
Not an easy task

Of course, the challenge is to conceive appropriate questions:

- to separate as accurately as possible the aspects pertaining to each learning dimension
- to limit the application of stereotypical solution patterns
- to attain a balance between “exam anxiety” and weakening of motivation

The present work is (just) an attempt in this direction
Of course, the challenge is to conceive appropriate questions:

- to separate as accurately as possible the aspects pertaining to each learning dimension
- to limit the application of stereotypical solution patterns
- to attain a balance between “exam anxiety” and weakening of motivation

The present work is (just) an attempt in this direction
To sum up, the main research questions are:

- How do the findings of Götschi et al. (2003), Sanders et al. (2006) generalize to a different context?
- To what extent is fluency with the computation model predictive of the achievement of other higher-level competences?
- Which learning dimensions are more problematic for the students?
Research questions

To sum up, the main research questions are:

- How do the findings of Götschi et al. (2003), Sanders et al. (2006) generalize to a different context?

- To what extent is fluency with the computation model predictive of the achievement of other higher-level competences?

- Which learning dimensions are more problematic for the students?
To sum up, the main research questions are:

- How do the findings of Götschi et al. (2003), Sanders et al. (2006) generalize to a different context?
- To what extent is fluency with the computation model predictive of the achievement of other higher-level competences?
- Which learning dimensions are more problematic for the students?
To sum up, the main research questions are:

- How do the findings of Götschi et al. (2003), Sanders et al. (2006) generalize to a different context?

- To what extent is fluency with the computation model predictive of the achievement of other higher-level competences?

- Which learning dimensions are more problematic for the students?
Questionnaire structure

Part I

- Dimension (ii)  computation model: Q1–Q2
- Dimension (iii)  problem domain: Q3–Q5
- Dimension (iv)  recursive structures: Q6–Q7

Part II

- Dimension (i)  language structures: Q8–Q9
- Dimension (v)  abstraction: Q10–Q11
Questionnaire structure

Part I

- Dimension (ii) *computation model*: Q1–Q2
- Dimension (iii) *problem domain*: Q3–Q5
- Dimension (iv) *recursive structures*: Q6–Q7

Part II

- Dimension (i) *language structures*: Q8–Q9
- Dimension (v) *abstraction*: Q10–Q11
Questionnaire structure

Part I

Dimension (ii)  \textit{computation model}:  Q1–Q2

Dimension (iii)  \textit{problem domain}:  Q3–Q5

Dimension (iv)  \textit{recursive structures}:  Q6–Q7

Part II

Dimension (i)  \textit{language structures}:  Q8–Q9

Dimension (v)  \textit{abstraction}:  Q10–Q11
Consider the following procedure definition:

```
(define power
  (lambda (b e)
    (cond ((= e 0) 1)
      ((even? e) (power (* b b) (/ e 2)))
      (else (* b (power b (- e 1)))))
  ))
```

Based on Scheme’s computation model, show the key evaluation steps for the expression `(power 2 6)`. In particular, report all the recursive procedure calls.

(See Götschi and colleagues, 2003)
Consider the following procedure definition:

```
(define power
  (lambda (b e)
    (cond ((= e 0) 1)
          ((even? e) (power (* b b) (/ e 2)))
          (else (* b (power b (- e 1)))))
  )
```

Based on Scheme’s computation model, show the key evaluation steps for the expression \( \text{power} \ 2 \ 6 \). In particular, report all the recursive procedure calls.

\( \text{(See Götschi and colleagues, 2003)} \)
Q2 – Trace of a computation: tree recursion

(define ldiff
  (lambda (u v)
    (cond
      ((= (string-length u) 0) (string-length v))
      ((= (string-length v) 0) (string-length u))
      ((char=? (string-ref u 0) (string-ref v 0))
       (ldiff (substring u 1) (substring v 1)))
      (else
       (+ 1 (min (ldiff (substring u 1) v)
                  (ldiff u (substring v 1))))
      )))

...key evaluation steps of (ldiff "tap" "tea") ... as above
Two positive integers are co-primes if and only if their \(gcd\) is 1 …

Someone has defined for you a boolean procedure \texttt{co-primes?}\(\) returning \texttt{true} if its arguments are co-primes, \texttt{false} otherwise.

Given two positive integers \(x\) and \(y\), neither of which is a divisor of the other, you are required to promptly determine if they are co-primes.

However, you are playing a game and you are not allowed to enter directly the expression \((\texttt{co-primes? } x \ y)\);

you can use \texttt{co-primes?}, provided you change the value of at least one of the arguments — and you cannot simply swap them.

Which Scheme expression would you enter to answer the query?
Two positive integers are co-primes if and only if their $gcd$ is 1.

Someone has defined for you a boolean procedure `procedure` returning `true` if its arguments are co-primes, `false` otherwise.

Given two positive integers $x$ and $y$, neither of which is a divisor of the other, you are required to promptly determine if they are co-primes.

However, you are playing a game and you are not allowed to enter directly the expression `(procedure args)`;

you can use `procedure`, provided you change the value of at least one of the arguments

Which Scheme expression would you enter to answer the query?
A palindrome is a word that reads the same forward or backwards …

Someone has defined for you a boolean procedure `palindrome?` returning `true` if its argument is a palindrome, `false` otherwise.

Given a string `s` of more than 2 letters …

**Similar game**
How many digits are needed to represent a natural number $N$ in a given base $B \geq 2$? . . .

Someone has already defined a procedure `number-of-digits` . . .

Given two integers $n$ and $b$ such that $n \geq b \geq 2$ . . .

Similar game
Q6 – Recursive structures: informal analysis

The figure represents a Sierpinski’s triangle $STr(s, k)$ where $s$ is the side length and $k$ is the recursion depth.

Look at the figure and describe the recursive structure in words.

In particular, answer the following questions:

- Which figure represents the base case of the recursion?
- Which operations would you apply in order to obtain $STr(s, k+1)$ from $STr(s, k)$? (to increase the recursion depth)
Q6 – Recursive structures: informal analysis

The figure represents a Sierpinski’s triangle \( STr(s, k) \) where \( s \) is the side length and \( k \) is the recursion depth . . .

Look at the figure and describe the recursive structure in words.

In particular, answer the following questions:

- Which figure represents the base case of the recursion?
- Which operations would you apply in order to obtain \( STr(s, k + 1) \) from \( STr(s, k) \)? (to increase the recursion depth)
A string of open and close parentheses is a *depth nesting* if all parentheses match correctly and if no open parenthesis follows a close one.

For example, "()", "((()))", "(((((())))))" are depth nestings, whereas "(()", "(()())", "(()))" are not.

Write a boolean procedure `depth-nesting?` in Scheme that, given a string of parentheses, verifies if it is a *depth nesting*. 
Combine the following fragments of Scheme code into a consistent procedure definition:

```
(define f _________)  (if (= x 1) 1 ________)

(if (> y x) 1 ____ ) (lambda (x) __________)

(let ( ____ ) _____ ) (y (+ (f (- x 1)) 2)) y
```

Each of the seven fragments must be used exactly once and the free spaces have to be replaced with structures built from other fragments.

The procedure should return a definite value for all positive integers.
Write in Scheme a recursive procedure which computes the function $h$ defined as follows for all pairs of nonnegative integers:

$$h(u, v) = \begin{cases} 
1 & \text{if } u = 0 \\
2 \cdot h(u - 1, v) & \text{if } u = v \\
h(v, u) & \text{if } u > v \\
h(u - 1, v) + h(v - 1, u) & \text{otherwise}
\end{cases}$$
Q10 – Procedural abstraction: functional relations

The procedure `add` adds two nonnegative integers represented by binary strings of 0s and 1s.

The program processes the strings starting from the less significant bits and passing forward the appropriate carry values.

For each procedure, choose a meaningful name and describe the relations between the returned value and the input arguments . . .

```
(define add ; returns: binary string
  (lambda (x y) ; x, y: binary strings
    (P 0 (Q x "") (Q y "") "")
  )
)```

C. Mirolo, University of Udine
Learning (through) Recursion...
Q10 – Procedural abstraction: functional relations

(define \textit{P} ; returns: binary string
 (\textit{lambda} (v x y z); x, y, z: binary strings
 (\textit{cond}
  ; v: integer 0/1
  ((and (string=? x "") (= v 0))
   (Q y z))
  ((and (string=? y "") (= v 0))
   (Q x z))
  (else
   (let ((u (+ v (\textit{F} x) (\textit{F} y))))
     \textit{P} (quotient u 2) (G x) (G y)
     (H (remainder u 2) z))
   ))
 )))

(define \textit{Q} ; returns: binary string
 (\textit{lambda} (x y); x, y: binary strings
 (if (string=? x "")
   y
   (Q (substring x 1)
    (string-append (substring x 0 1) y))
 )))

(define \textit{F} ; returns: integer 0/1
 (\textit{lambda} (x); x: binary string
 (\textit{cond}
  ((string=? x "")
   0)
  ((char=? (string-ref x 0) #\0)
   0)
  (else ; #\1
   1)
 )))

(define \textit{G} ; returns: binary string
 (\textit{lambda} (x); x: binary string
 (if (string=? x "")
   ""
   (substring x 1))
 )))

(define \textit{H} ; returns: binary string
 (\textit{lambda} (v x); x: binary string
 (if (= v 0) ; v: integer 0/1
   (string-append "0" x)
   (string-append "1" x)
 )))

C. Mirolo, University of Udine
Learning (through) Recursion...
The procedure qval takes as input a string that represents in binary notation a number with integral and fractional parts, separated by a point, and returns its numerical value.

For example, the evaluation of the expression (qval "10.011") returns 2.375...

At each recursive call, either the most significant bit of the integral part or the least significant bit of the fractional part is processed.

(define qval ; returns: rational number
  (lambda (b) ; b: string (of 0/1/.)
    (valrec b 0 0)
  ))
Q11 – Abstraction over patterns: Code organization

```
(define valrec ; returns: rational number
  (lambda (b n f) ; n >= 0 integer, 0 <= f < 1, b: string
    (cond ((= (string-length b) 0)
      n) ; only integral part
      ((char=? (string-ref b 0) '#\0)
        (valrec (substring b 1) (* 2 n) f))
      ((char=? (string-ref b 0) '#\1)
        (valrec (substring b 1) (+ (* 2 n) 1) f))
      (else ; #\.
        (cond ((= (string-length b) 1)
          (+ n f)) ; integral + fractional parts
          ((char=? '#\0
             (string-ref b (- (string-length b) 1)))
            (valrec
              (substring b 0 (- (string-length b) 1))
              n (/ f 2)))
          (else ; #\1
            (valrec
              (substring b 0 (- (string-length b) 1))
              n (+ 0.5 (/ f 2)))
          )))
  ))
```
Q11 – Abstraction over patterns: Code organization

Give the program a better structure for an easier understanding.

To this aim, apply the `let` construct and/or define suitable (sub)procedures to subdivide the complex operations into simpler ones and so improve the readability.

The full text of the questionnaire is available online at:

http://nid.dimi.uniud.it/download/rec_test_09.pdf
Q11 – Abstraction over patterns: Code organization

Give the program a better structure for an easier understanding.

To this aim, apply the `let` construct and/or define suitable (sub)procedures to subdivide the complex operations into simpler ones and so improve the readability.

The full text of the questionnaire is available online at:

http://nid.dimi.uniud.it/download/rec_test_09.pdf
Context

- CS1 students: 45 (’08); 50 (’09, part I); 38 (’09, part II)
- “Introductory programming”, first semester (after 2 months)
- Procedural abstraction in Scheme, Hailperin’s et al. (1999)
- Both computational and abstract program models
- Unusual questions w.r.t. exam questions
Context

- CS1 students: 45 ('08); 50 ('09, part I); 38 ('09, part II)
- “Introductory programming”, first semester (after 2 months)
- Procedural abstraction in Scheme, Hailperin’s et al. (1999)
- Both computational and abstract program models
- Unusual questions w.r.t. exam questions
Context

- CS1 students: 45 (’08); 50 (’09, part I); 38 (’09, part II)
- “Introductory programming”, first semester (after 2 months)
- Procedural abstraction in Scheme, Hailperin’s et al. (1999)
- Both computational and abstract program models
- Unusual questions w.r.t. exam questions
CS1 students: 45 (’08); 50 (’09, part I); 38 (’09, part II)

“Introductory programming”, first semester (after 2 months)

Procedural abstraction in Scheme, Hailperin’s et al. (1999)

Both computational and abstract program models

Unusual questions w.r.t. exam questions
Context

- CS1 students: 45 (’08); 50 (’09, part I); 38 (’09, part II)
- “Introductory programming”, first semester (after 2 months)
- Procedural abstraction in Scheme, Hailperin’s et al. (1999)
- Both computational and abstract program models
- Unusual questions w.r.t. exam questions
Two sessions

- **First session** (2 hours):
  Part I of the questionnaire (7 questions)

- **Second session** (2 hours, a week later):
  Part II of the questionnaire (4 questions)
  plus specific *individual* questions on Part I
  to clarify ambiguities, to correct minor mistakes, ...
Sessions

Two sessions

- **First session** (2 hours):
  Part I of the questionnaire (7 questions)

- **Second session** (2 hours, a week later):
  Part II of the questionnaire (4 questions)
  plus specific *individual* questions on Part I
  to clarify ambiguities, to correct minor mistakes, ...
Two sessions

- **First session** (2 hours):
  Part I of the questionnaire (7 questions)

- **Second session** (2 hours, a week later):
  Part II of the questionnaire (4 questions)

  plus specific *individual* questions on Part I to clarify ambiguities, to correct minor mistakes, ...
Two sessions

- **First session** (2 hours):
  Part I of the questionnaire (7 questions)

- **Second session** (2 hours, a week later):
  Part II of the questionnaire (4 questions)
  plus specific *individual* questions on Part I to clarify ambiguities, to correct minor mistakes, ...
Outline

1. Introduction
2. Instrument
   - questionnaire – part I
   - questionnaire – part II
   - administration
3. Analysis and results
   - computation model
   - relations and structures
   - language and abstraction
4. Discussion
   - summary
   - conclusions

C. Mirolo, University of Udine

Learning (through) Recursion...
Most answers can be classified into three categories:

- Accurate answers
- Superficial, fragmentary or stereotypical answers
- Answers affected by serious mistakes (or not provided)
Most answers can be classified into three categories:

- Accurate answers
- Superficial, fragmentary or stereotypical answers
- Answers affected by serious mistakes (or not provided)
Assessment

Most answers can be classified into three categories:

- Accurate answers
- Superficial, fragmentary or stereotypical answers
- Answers affected by serious mistakes (or not provided)
Most answers can be classified into three categories:

- Accurate answers
- Superficial, fragmentary or stereotypical answers
- Answers affected by serious mistakes (or not provided)
Q1–Q2: Traces of recursive computations

Analysis based on Götschi’s et al. (2003) methodology:

<table>
<thead>
<tr>
<th></th>
<th>sound model</th>
<th>fragile model</th>
<th>forward model</th>
<th>other (e.g.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>invoc. flow</td>
<td>accurate</td>
<td>accurate</td>
<td>accurate</td>
<td>unclear</td>
</tr>
<tr>
<td>base cases</td>
<td>switch</td>
<td>switch</td>
<td>stop</td>
<td>unclear</td>
</tr>
<tr>
<td>return flow</td>
<td>accurate</td>
<td>unclear</td>
<td>none</td>
<td>elusive</td>
</tr>
</tbody>
</table>
Q1–Q2: Traces of recursive computations

Analysis based on Götschi’s et al. (2003) methodology:

<table>
<thead>
<tr>
<th></th>
<th>sound model</th>
<th>fragile model</th>
<th>forward model</th>
<th>other (e.g.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>invoc. flow</strong></td>
<td>accurate</td>
<td>accurate</td>
<td>accurate</td>
<td>unclear</td>
</tr>
<tr>
<td><strong>base cases</strong></td>
<td>switch</td>
<td>switch</td>
<td>stop</td>
<td>unclear</td>
</tr>
<tr>
<td><strong>return flow</strong></td>
<td>accurate</td>
<td>unclear</td>
<td>none</td>
<td>elusive</td>
</tr>
</tbody>
</table>
Q1–Q2: Traces of recursive computations

Analysis based on Götschi’s et al. (2003) methodology:

<table>
<thead>
<tr>
<th></th>
<th>sound model</th>
<th>fragile model</th>
<th>forward model</th>
<th>other (e.g.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>invoc. flow</td>
<td>accurate</td>
<td>accurate</td>
<td>accurate</td>
<td>unclear</td>
</tr>
<tr>
<td>base cases</td>
<td>switch</td>
<td>switch</td>
<td>stop</td>
<td>unclear</td>
</tr>
<tr>
<td>return flow</td>
<td>accurate</td>
<td>unclear</td>
<td>none</td>
<td>elusive</td>
</tr>
</tbody>
</table>
Q1–Q2: Traces of recursive computations

Analysis based on Götschi’s et al. (2003) methodology:

<table>
<thead>
<tr>
<th></th>
<th>sound model</th>
<th>fragile model</th>
<th>forward model</th>
<th>other (e.g.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>invoc. flow</td>
<td>accurate</td>
<td>accurate</td>
<td>accurate</td>
<td>unclear</td>
</tr>
<tr>
<td>base cases</td>
<td>switch</td>
<td>switch</td>
<td>stop</td>
<td>unclear</td>
</tr>
<tr>
<td>return flow</td>
<td>accurate</td>
<td>unclear</td>
<td>none</td>
<td>elusive</td>
</tr>
</tbody>
</table>
Q1–Q2: Traces of recursive computations

Analysis based on Götschi’s et al. (2003) methodology:

<table>
<thead>
<tr>
<th></th>
<th>sound model</th>
<th>fragile model</th>
<th>forward model</th>
<th>other (e.g.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>invoc. flow</td>
<td>accurate</td>
<td>accurate</td>
<td>accurate</td>
<td>unclear</td>
</tr>
<tr>
<td>base cases</td>
<td>switch</td>
<td>switch</td>
<td>stop</td>
<td>unclear</td>
</tr>
<tr>
<td>return flow</td>
<td>accurate</td>
<td>unclear</td>
<td>none</td>
<td>elusive</td>
</tr>
</tbody>
</table>
Q1: “Sound model” trace

\[
\begin{align*}
&\text{(power } 2 \ 6) \\
&\text{(power } 4 \ 3) \\
&\text{(* } 4 \text{ (power } 4 \ 2)) \\
&\text{(power } 16 \ 1) \\
&\text{(*16 (power } 16 \ 0)) \\
&1
\end{align*}
\]
Q1: “Forward model” trace

\[
e \rightarrow \text{pan} \Rightarrow (\text{power } 4 \ 3)
\]

\[
\downarrow\ e \rightarrow \text{dispair}
\]

\[
(16 \ (\text{power } 4 \ 2))\ \text{dispair}
\]

\[
\downarrow\ e \rightarrow \text{pan}
\]

\[
(\text{power } 16 \ 1)
\]

\[
\downarrow\ e \rightarrow \text{dispair}
\]

\[
(16 \ (\text{power } 16 \ 0))
\]

\[
\downarrow\ e \rightarrow 0
\]

\[
1
\]
Q2: “Odd model” trace

\[
(\text{ldiff} \text{ "top" "tea"}) \\
(\text{ldiff} \text{ "op" "ea"}) \\
(\text{ldiff} \text{ "n" "a"}) \\
(\text{ldiff} \text{ "n" "a"}) \downarrow \\
0
\]
Q1–Q2: Mental models inferred from students’ traces

- **linear recursion (2009)**
  - sound model
  - fragile model
  - forward model
  - odd model
  - other
  - no answer

- **linear recursion (2008)**
  - sound model
  - fragile model
  - forward model
  - odd model
  - other
  - no answer

- **tree recursion (2009)**
  - sound model
  - fragile model
  - forward model
  - odd model
  - other
  - no answer

- **tree recursion (2008)**
  - sound model
  - fragile model
  - forward model
  - odd model
  - other
  - no answer

- **Sanders et al. (2005)**
  - sound model
  - fragile model
  - forward model
  - odd model
  - other
  - no answer
Q3–Q5: Relations in/with the problem domain

The relations between problem instances may be:

- Correct and also appropriate for a recursive definition
- Correct but not appropriate for a recursive definition
- Incorrect
Q3: Sample answers

Correct and appropriate for recursion:

\[
(\text{if } (> \times y) (\text{co-primes? } (- \times y) y) (\text{co-primes? } \times (-y \times)))
\]

Correct but not appropriate for recursion:

\[
(\text{co-primes? } (+ \times y) \times)
\]

Incorrect:

\[
(\text{co-primes? } (+ \times 1) y)
\]
Q6–Q7: Recursive structures

The approach may be:

- Structurally recursive (*top-down*)
- The result of adapting some “iteration” pattern (or *bottom-up*)
- Useless
Q6: **Top-down** vs. **flat characterization**

"...you should copy the given triangle three times, ..."
Q6: Top-down vs. flat characterization

"Decompose once more every small triangle of the drawing into 4 equilateral triangles."
Q3–Q7: Recursive relations and structures (’09)

Q1-2: sound, uneven, unsound

Q3-5: 3 sound, 2 uneven, 1 unsound

Q6: struct. consistent, useless

Q7: structurally recursive
Q8–Q9: Language constructs

The definition resulting from Q8’s assembly may be:

- A well founded recursive definition (*base case!*)
- Only syntactically correct (stereotypical construction)
- Syntactically incorrect

The feedback relative to Q9 is less interesting
The definition resulting from Q8’s assembly may be:

- A well founded recursive definition (*base case*)!
- Only syntactically correct (stereotypical construction)
- Syntactically incorrect

The feedback relative to Q9 is less interesting
Q8: Well founded recursive definition...

(define f
  (lambda (x) ; int x > 0
    (if (= x 1)
      1
      (let ( (y (+ (f (- x 1)) 2)) )
        (if (> y x) 1 y)
        )))
))
Q8: ...vs. ill founded recursive definition

```
(define f
  (lambda (x)
    (let ((y (+ (f (- x 1)) 2)))
      (if (= x 1)
        1
        (if (> y x) 1 y)
      )))
  )
```

C. Mirolo, University of Udine
Learning (through) Recursion...
Q10–Q11: Abstraction over patterns and processes

Analysis of more complex code:

- Abstraction over complex patterns (*what to*)
- Abstraction over simple patterns (*how to*)
- Difficulty to cope with complexity
Q10: Dichotomy "what to" vs. "how to"

"... Adds two strings of 0s and 1s, and appends the result to z; v is the carry, x and y are the strings to add, and z is the partial result..."
Q10: Dichotomy “what to” vs. “how to”

“If x is not empty, it adds to the first character of x the string y and proceeds recursively until x becomes empty…”

```
(define Q ; valore: stringa di 0/1
    (lambda (x y) ; x, y: stringhe di 0/1
        (if (string=? x "")
            y
            (Q (substring x 1)
                (string-append (substring x 0 1) y)))
    )
```

Name da preferire a Q: STRUNION

Function solved: if x is not empty, add y to the first character of x and proceed recursively until x becomes empty.

```
xs: x = abc y = def -> y = cdef
```
Q8–Q11: Language and abstraction ('09)

- **Q1-2**: Sound (even), Uneven, Unsound
- **Q8**: Well-founded, Ill-founded
- **Q9**: Strict adherence to syntax, Loose
- **Q10**: High, Moderate, Low
- **Q11**: High, Moderate

Abstraction level:

- 0
- 5
- 10
- 15
- 20
- 25
- 30
- 35

C. Mirolo, University of Udine

Learning (through) Recursion...
Outline

1 Introduction
2 Instrument
   - questionnaire – part I
   - questionnaire – part II
   - administration
3 Analysis and results
   - computation model
   - relations and structures
   - language and abstraction
4 Discussion
   - summary
   - conclusions
Summary of the results

- Most students develop viable mental models of recursive computations
- About 1/4 of the students experience serious difficulties to trace both the proposed computations
- Good accordance with the figures of Sanders et al. (2006)
- A consistent mental model of recursive computations is implied by the ability to use recursion in problem solving

C. Mirolo, University of Udine
Summary of the results

- Most students develop viable mental models of recursive computations
- About 1/4 of the students experience serious difficulties to trace both the proposed computations
- Good accordance with the figures of Sanders et al. (2006)
- A consistent mental model of recursive computations is implied by the ability to use recursion in problem solving
Summary of the results

- Most students develop viable mental models of recursive computations.
- About 1/4 of the students experience serious difficulties to trace both the proposed computations.
- Good accordance with the figures of Sanders et al. (2006).
- A consistent mental model of recursive computations is implied by the ability to use recursion in problem solving.
Summary of the results

- Most students develop viable mental models of recursive computations

- About 1/4 of the students experience serious difficulties to trace both the proposed computations

- Good accordance with the figures of Sanders et al. (2006)

- A consistent mental model of recursive computations is implied by the ability to use recursion in problem solving

...
Summary of the results

... However, it doesn’t seem to be *per se* sufficient for the achievement of higher-level skills.

- Fewer students find it natural to establish relations between problem instances or to think at a higher level of abstraction.

- Among the students who have traced at least one computation properly:
  - About 1/3 provide poor answers to Q3–Q5
  - About 1/3 provide poor answers to Q10–Q11
Summary of the results

• However, it doesn’t seem to be *per se* sufficient for the achievement of higher-level skills.

• Fewer students find it natural to establish relations between problem instances or to think at a higher level of abstraction.

• Among the students who have traced at least one computation properly:
  • About 1/3 provide poor answers to Q3–Q5
  • About 1/3 provide poor answers to Q10–Q11
Summary of the results

... However, it doesn’t seem to be *per se* sufficient for the achievement of higher-level skills.

Fewer students find it natural to establish relations between problem instances or to think at a higher level of abstraction.

Among the students who have traced at least one computation properly:
- About 1/3 provide poor answers to Q3–Q5
- About 1/3 provide poor answers to Q10–Q11
Summary of the results (2009)
Conclusions

After replication of the experiment, I am reasonably convinced of the consistency of the results.

Two main conclusions can be drawn from the study:

- Neither the language syntax nor the computation model are crucial for the learning process.
- We should spent more effort on the declarative aspects implied by programming.

“The key to comprehending any form of abstraction, including recursion, is to focus on the what and down play the how”

Sooriamurthi (2001)
After replication of the experiment, I am reasonably convinced of the consistency of the results.

Two main conclusions can be drawn from the study:

- Neither the language syntax nor the computation model are crucial for the learning process.
- We should spend more effort on the declarative aspects implied by programming.

“The key to comprehending any form of abstraction, including recursion, is to focus on the what and down play the how”

Sooriamurthi (2001)
Conclusions

After replication of the experiment, I am reasonably convinced of the consistency of the results

Two main conclusions can be drawn from the study:

- Neither the language syntax nor the computation model are crucial for the learning process
- We should spent more effort on the declarative aspects implied by programming

“The key to comprehending any form of abstraction, including recursion, is to focus on the what and down play the how”

Sooriamurthi (2001)
After replication of the experiment, I am reasonably convinced of the consistency of the results.

Two main conclusions can be drawn from the study:

- Neither the language syntax nor the computation model are crucial for the learning process.
- We should spent more effort on the declarative aspects implied by programming.

“The key to comprehending any form of abstraction, including recursion, is to focus on the what and down play the how”

Sooriamurthi (2001)
Perspectives

Possible directions of future work:

- Correlating fluency with *recursion* and with *iteration*
- Correlating the above results with students’ performance in the exams
- Keys to interpret the difficulties
- And, of course, improving the questionnaire

Anyway, trying to restate fine-grained learning goals in terms of verifiable micro-tasks is a good exercise . . .
Perspectives

Possible directions of future work:

- Correlating fluency with recursion and with iteration
- Correlating the above results with students’ performance in the exams
- Keys to interpret the difficulties
- And, of course, improving the questionnaire

Anyway, trying to restate fine-grained learning goals in terms of verifiable micro-tasks is a good exercise . . .
Perspectives

Possible directions of future work:

- Correlating fluency with *recursion* and with *iteration*
- Correlating the above results with students’ performance in the exams
- Keys to interpret the difficulties
- And, of course, improving the questionnaire

Anyway, trying to restate fine-grained learning goals in terms of verifiable micro-tasks is a good exercise . . .
Possible directions of future work:

- Correlating fluency with *recursion* and with *iteration*
- Correlating the above results with students’ performance in the exams
- Keys to interpret the difficulties
- And, of course, improving the questionnaire

Anyway, trying to restate fine-grained learning goals in terms of verifiable micro-tasks is a good exercise . . .
And finally . . .

Thanks for your patience
References


References


References


## Appendix 1: Students’ answers to items Q3–Q5

<table>
<thead>
<tr>
<th>Q3–Q5</th>
<th>logically correct</th>
<th>appropriate for recursion</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>2009</td>
<td>2009</td>
</tr>
<tr>
<td>3 answers</td>
<td>32%</td>
<td>16%</td>
</tr>
<tr>
<td>2 answers</td>
<td>20%</td>
<td>27%</td>
</tr>
<tr>
<td>1 answer</td>
<td>16%</td>
<td>33%</td>
</tr>
</tbody>
</table>

C. Mirolo, University of Udine

Learning (through) Recursion...
Appendix

Q3–Q5: Comparison 2008/2009

- Correct relations (2009)
- Correct relations (2008)
- Complexity reduced (2009)
- Complexity reduced (2008)

C. Mirolo, University of Udine
## Appendix 2: Students’ answers to Q6, Q7, Q10, Q11

<table>
<thead>
<tr>
<th>year</th>
<th>2009</th>
<th>2008</th>
<th>2009</th>
<th>2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>recursion:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td>Q6</td>
<td>Q7</td>
<td>Q7</td>
<td></td>
</tr>
<tr>
<td>consistent</td>
<td>42%</td>
<td>40%</td>
<td>50%</td>
<td>42%</td>
</tr>
<tr>
<td>structural</td>
<td>12%</td>
<td>11%</td>
<td>50%</td>
<td>33%</td>
</tr>
<tr>
<td>abstraction:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q10</td>
<td></td>
<td>Q11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>high-level</td>
<td>21%</td>
<td>24%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>moderate-level</td>
<td>39%</td>
<td>50%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C. Mirolo, University of Udine

Learning (through) Recursion...
Appendix

Q6–Q7: Comparison 2008/2009

- Q6 (2009)
  - Smart rec. structure: 10%
  - Otherwise correct: 30%

- Q6 (2008)
  - Smart rec. structure: 10%
  - Otherwise correct: 30%

- Q7 (2009)
  - Smart rec. structure: 50%

- Q7 (2008)
  - Smart rec. structure: 30%
  - Otherwise correct: 10%
### Appendix 3: Correlations (2009)

<table>
<thead>
<tr>
<th></th>
<th>Q3–Q5</th>
<th>Q6–Q7</th>
<th>Q8–Q9</th>
<th>Q10–Q11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1–Q2</td>
<td>0.62</td>
<td>0.71</td>
<td>0.70</td>
<td>0.67</td>
</tr>
<tr>
<td>Q3–Q5</td>
<td></td>
<td>0.66</td>
<td>0.44</td>
<td>0.51</td>
</tr>
<tr>
<td>Q6–Q7</td>
<td></td>
<td></td>
<td>0.59</td>
<td>0.58</td>
</tr>
<tr>
<td>Q8–Q9</td>
<td></td>
<td></td>
<td></td>
<td>0.56</td>
</tr>
</tbody>
</table>

C. Mirolo, University of Udine  
Learning (through) Recursion...
### Appendix 3: Correlations (2008)

<table>
<thead>
<tr>
<th></th>
<th>Q1–Q2</th>
<th>Q3–Q5</th>
<th>Q6–Q7</th>
<th>Q3–Q7</th>
<th>Q1–Q7</th>
</tr>
</thead>
<tbody>
<tr>
<td>exam</td>
<td>0.70</td>
<td>0.69</td>
<td>0.53</td>
<td>0.68</td>
<td>0.74</td>
</tr>
</tbody>
</table>
Q10 – Procedural abstraction: functional relations

The procedure `add` adds two nonnegative integers represented by binary strings of 0s and 1s.

The program processes the strings starting from the less significant bits and passing forward the appropriate carry values.

For each procedure, choose a meaningful name and describe the relations between the returned value and the input arguments...

```
(define add ; returns: binary string
  (lambda (x y) ; x, y: binary strings
    (P 0 (Q x "") (Q y "") ")")
)
```
Q10 – Procedural abstraction: functional relations

(define P ; returns: binary string
  (lambda (v x y z) ; x, y, z: binary strings
    (cond ; v: integer 0/1
      ((and (string=? x "") (= v 0))
        (Q y z))
      ((and (string=? y "") (= v 0))
        (Q x z))
      (else
        (let ((u (+ v (F x) (F y))))
          (P (quotient u 2) (G x) (G y)
              (H (remainder u 2) z)))))
  ))
Q10 – Procedural abstraction: functional relations

(define Q ; returns: binary string
  (lambda (x y) ; x, y: binary strings
    (if (string=? x "")
        y
        (Q (substring x 1)
            (string-append (substring x 0 1) y)))))

(define F ; returns: integer 0/1
  (lambda (x) ; x: binary string
    (cond ((string=? x "") 0)
          ((char=? (string-ref x 0) #\0) 0)
          (else 1)))

C. Mirolo, University of Udine
Learning (through) Recursion...
Q10 – Procedural abstraction: functional relations

(define G ; returns: binary string
  (lambda (x) ; x: binary string
    (if (string=? x "")
      ""
      (substring x 1))
  )))

(define H ; returns: binary string
  (lambda (v x) ; x: binary string
    (if (= v 0) ; v: integer 0/1
      (string-append "0" x)
      (string-append "1" x)
    )))

C. Mirolo, University of Udine
Learning (through) Recursion...
The procedure \texttt{qval} takes as input a string that represents in binary notation a number with integral and fractional parts, separated by a point, and returns its numerical value.

For example, the evaluation of the expression \((\texttt{qval } "10.011")\) returns \(2.375\ldots\)

At each recursive call, either the most significant bit of the integral part or the least significant bit of the fractional part is processed.

\begin{verbatim}
(define qval ; returns: rational number
  (lambda (b) ; b: string (of 0/1/.)
      (valrec b 0 0)
  ))
\end{verbatim}
(define valrec ; returns: rational number
  (lambda (b n f) ; n >= 0, 0 <= f < 1
    (cond
      ((= (string-length b) 0) n) ; integral part
      ((char=? (string-ref b 0) #\0)
        (valrec (substring b 1) (* 2 n) f))
      ((char=? (string-ref b 0) #\1)
        (valrec (substring b 1) (+ (* 2 n) 1) f))
      (else
        (cond
          . . .
          . . .
          ))))
... (else
  (cond
    ((= (string-length b) 1) (+ n f))
    ((char=? #\0 (string-ref b (- (string-length b) 1)))
      (valrec
        (substring b 0 (- (string-length b) 1))
        n (/ f 2)))
    (else
      (valrec
        (substring b 0 (- (string-length b) 1))
        n (+ 0.5 (/ f 2)))
    )) ...
Give the program a better structure for an easier understanding.

To this aim, apply the `let` construct and/or define suitable (sub)procedures to subdivide the complex operations into simpler ones and so improve the readability.