What do gambling, magic, and human evolution have in common?

Limsoon Wong

Fun With Invariants

Limsoon Wong
Plan

- What is an invariant?
  - Bet on color of the bean
  - 21 cards
- Origin of Polynesians
- Make a list sorted
- Design a good database
- Diagnose leukemia's
- Apples vs oranges
- Make exponentiation faster
- Draw a straight line

- What will we learn?
  - Problem solving by logical reasoning on invariants
  - Problem solving by rectifying violation of invariants
  - Guilt by association of invariants
  - Solution optimization by preserving invariants

What is an invariant?
Shall we bet on the color of the bean that is left behind?

• Suppose you have a bag of x red beans and y green beans

• Repeat the following:
  – Remove 2 beans
  – If both green, discard both
  – If both red, discard one, put back one
  – If one green and one red, discard red, put back green

• If one bean is left behind, can you predict its colour?

Bet on the last green bean

• Suppose you have a bag of x red beans and y green beans

• Repeat the following:
  – Remove 2 beans
  – If both green, discard both
  – If both red, discard one, put back one
  – If one green and one red, discard red, put back green

• If one bean is left behind, can you predict its colour?

• When the parity of green beans is odd, it remains odd...
  – Start with $y=2n+1$

  – $y=2n+1 \rightarrow y=2n-1$
  – $y=2n+1 \rightarrow y=2n+1$
  – $y=2n+1 \rightarrow y=2n+1$

• It must be green!
Bet on the last red bean

• Suppose you have a bag of x red beans and y green beans
• Repeat the following:
  – Remove 2 beans
  – If both green, discard both
  – If both red, discard one, put back one
  – If one green and one red, discard red, put back green
• If one bean is left behind, can you predict its colour?
  • When the parity of green beans is even, it remains even…
  • Start with y=2n

  – y=2n \rightarrow y=2n-2
  – y=2n \rightarrow y=2n
  – y=2n \rightarrow y=2n

• If one bean is left behind, can you predict its colour?
  • It must be red!

Bet on color of the last bean … and win!

• Suppose you have a bag of x red beans and y green beans
• Repeat the following:
  – Remove 2 beans
  – If both green, discard both
  – If both red, discard one, put back one
  – If one green and one red, discard red, put back green
• If one bean is left behind, can you predict its colour?
  • If you start with odd # (even #) of green beans, there will always be an odd # (even #) of green beans in the bag

\Rightarrow Parity of green beans is invariant

\Rightarrow Bean left behind is green iff you start with odd # of green beans
• What have we just seen?

• Problem solving by logical reasoning on invariants

Welcome to the Magical World...
The 21 Card Trick

1. Magician asks you to remember any one card from a deck of 21 cards as your card. Do not tell him what the card is.

2. He deals the 21 cards face down, from top to bottom and left to right, into 3 equal piles.

3. Next, he fans the piles to you and asks you to look for the pile of cards which contains your card and pass the pile back to him.

4. Again, he stacks up the 3 piles on top of each other and redistribute, from top to bottom and left to right, into 3 equal piles.

5. He repeats step (3) and (4) 2 more times.

6. Finally, he deals your card right out from the rest of the 21 cards!

How does he manage that?!

The Trick

- The pile containing the card is being placed in the middle of the other 2 piles.

- Imposing constraints on where the card can move to...
The Invariant Underlying the Trick

Assuming the chosen card is in the first pile.

<table>
<thead>
<tr>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
</tr>
</tbody>
</table>

After the first distribution, ...

After the second distribution, ...

After the third distribution, ...

This section of the ppt is courtesy of Toh Xiu Ping

• What have we just seen?

• Problem solving by logical reasoning on invariants
Where do Polynesians come from?

In the course of evolution…
What is the invariant?

• Mitochondrial DNA accumulates 1 mutation about every 10,000 years
• Human history is not so long relative to this

⇒ When a nucleotide in mitochondrial DNA is mutated it stays mutated through future generations

Do Polynesians come from Asia or America?
Origin of Polynesians

- Common mitochondrial control seq from Rarotonga have variants at positions 189, 217, 247, 261. Less common ones have 189, 217, 261.
- More 189, 217 closer to Taiwan. More 189, 217, 261 closer to Rarotonga.
- 247 not found in America ⇒ Polynesians came from Taiwan!
- Seq from Taiwan natives have variants 189, 217.
- Seq from regions in betw have variants 189, 217, 261.
- More 189, 217 closer to Taiwan. More 189, 217, 261 closer to Rarotonga.
- 247 not found in America ⇒ Polynesians came from Taiwan!
- Seq from Taiwan natives have variants 189, 217.
- Seq from regions in betw have variants 189, 217, 261.
- More 189, 217 closer to Taiwan. More 189, 217, 261 closer to Rarotonga.
- 247 not found in America ⇒ Polynesians came from Taiwan!

Are Europeans descended purely from Cro Magnons? Purely from Neanderthals? Or mixed?

Neanderthal

Cro Magnon
Neanderthal vs Cro Magnon

- Based on palaeontology, Neanderthal & Cro Magnon last shared an ancestor 250,000 yrs ago
- Mitochondrial DNA accumulates 1 mutation per 10,000 yrs

⇒ If Europeans have mixed ancestry, the mitochondrial DNA betw 2 Europeans should have ~25 diff w/ high probability

- The number of diff betw Welsh is ~3, & at most 8.
- When compared w/ other Europeans, 14 diff at most

⇒ Ancestor either 100% Neanderthal or 100% Cro Magnon

- Mitochondrial DNA from Neanderthal have 26 diff from Europeans

⇒ Ancestor must be 100% Cro Magnon

The “Invariant” Perspective

- The invariant:

  When a nucleotide in mitochondrial DNA is mutated it stays mutated through future generations

- The lesson learned:

  Figure out origins of Polynesians and Europeans by logical reasoning on invariant
How to get a list sorted?

What makes a list a sorted list?

• What is a sorted list?

  A list L is sorted iff \( L[i] \leq L[j] \) for all adjacent positions \( i < j \)

• So how do you make a list M become sorted?

  While \( M[i] > M[j] \) for some adjacent positions \( i < j \) {
    
    swap \( M[i], M[j] \)

  }

  What makes a list a sorted list?
• Invariant of sorted lists

A list L is sorted iff \( L[i] \leq L[j] \) for all adjacent positions \( i < j \)

• Making a list M become sorted:

```c
While M[i] > M[j] for some adjacent positions i < j {
    swap M[i], M[j]
}
```

• Find violation of the invariant

• Fix it

• When no more violation, the list must be sorted!

---

• What have we just seen?

• Problem solving by rectifying violation of invariants
What is a good database design?

Relational Data Model

• Data are represented as a two-dimensional table

• It is a logical representation, not a physical representation
  – Ordering of the rows is irrelevant
  – Ordering of the columns is irrelevant
  – How the rows and columns of a table are stored is irrelevant
  – …
Example

Contracts

<table>
<thead>
<tr>
<th>Contract No</th>
<th>Star</th>
<th>Studio</th>
<th>Title</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Carrie Fisher</td>
<td>Fox</td>
<td>Star Wars</td>
<td>$$ $$</td>
</tr>
<tr>
<td>2</td>
<td>Mark Hamill</td>
<td>Fox</td>
<td>Star Wars</td>
<td>$$ $$</td>
</tr>
<tr>
<td>3</td>
<td>Harrison Ford</td>
<td>Fox</td>
<td>Star Wars</td>
<td>$$ $$</td>
</tr>
</tbody>
</table>

Stars

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carrie Fisher</td>
<td>Hollywood</td>
</tr>
<tr>
<td>Mark Hamill</td>
<td>Brentwood</td>
</tr>
<tr>
<td>Harrison Ford</td>
<td>Beverly Hills</td>
</tr>
</tbody>
</table>

Movies

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>Film Type</th>
<th>Studio</th>
<th>Star</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mighty Ducks</td>
<td>1991</td>
<td>104</td>
<td>Color</td>
<td>Fox</td>
<td>Carrie Fisher</td>
</tr>
<tr>
<td>Wayne's World</td>
<td>1992</td>
<td>95</td>
<td>Color</td>
<td>Fox</td>
<td>Mark Hamill</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Harrison Ford</td>
</tr>
</tbody>
</table>

Wrong Movies

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>Film Type</th>
<th>Studio</th>
<th>Star</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Carrie Fisher</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Mark Hamill</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Mighty Ducks</td>
<td>1991</td>
<td>104</td>
<td>Color</td>
<td>Disney</td>
<td>Emilio Estevez</td>
</tr>
</tbody>
</table>

Design Issues

- How many possible alternate ways to represent movies using tables?
- Why this particular set of tables to represent movies?
- Indeed, why not use this alternative single table below to represent movies?
Anomalies

• What’s wrong with the “Wrong Movies” table?

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>Film Type</th>
<th>Studio</th>
<th>Star</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Carrie Fisher</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Mark Hamill</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Mighty Ducks</td>
<td>1991</td>
<td>104</td>
<td>Color</td>
<td>Disney</td>
<td>Emilio Estevez</td>
</tr>
</tbody>
</table>

• Redundancy: Unnecessary repetition of info
• Update anomalies: If Star Wars is 125 min, we might carelessly update row 1 but not rows 2 & 3
• Deletion anomalies: If Emilio Estevez is deleted from stars of Mighty Ducks, we lose all info on that movie

Functional Dependency

• Functional dependency \((A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m)\)
  – If two tuples of a table \(R\) agree on attributes \(A_1, \ldots, A_n\), then they must also agree on attributes \(B_1, \ldots, B_m\)
  \(\Rightarrow\) Values of \(B\)'s depend on values of \(A\)'s

• Example: Title, Year \(\rightarrow\) Length, Film Type, Studio

• \(FD (A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m)\) is trivial if a \(B_i\) is an \(A_j\)
Can you identify the FD’s here?

Wrong Movies

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>Film Type</th>
<th>Studio</th>
<th>Star</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Carrie Fisher</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Mark Hamill</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Mighty Ducks</td>
<td>1991</td>
<td>104</td>
<td>Color</td>
<td>Disney</td>
<td>Emilio Estevez</td>
</tr>
</tbody>
</table>

- **Some FD’s:**
  - Title, Year $\rightarrow$ Length
  - Title, Year $\rightarrow$ Film Type
  - Title, Year $\rightarrow$ Studio

---

Keys

- **Key**
  - A minimal set of attributes \(\{A_1, \ldots, A_n\}\) that functionally determine all other attributes of a table
  - A key is trivial if it comprises the entire set of attributes of a table

- **Superkey**
  - A set of attributes that contains a key
Can you identify the superkeys here?

Wrong Movies

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>Film Type</th>
<th>Studio</th>
<th>Star</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Carrie Fisher</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Mark Hamill</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Mighty Ducks</td>
<td>1991</td>
<td>104</td>
<td>Color</td>
<td>Disney</td>
<td>Emilio Estevez</td>
</tr>
</tbody>
</table>

• **Superkeys**:
  – Any set of attributes that contains {Title, Year, Star} as a subset

Boyce-Codd Normal Form

• A relation $R$ is in **Boyce-Codd Normal Form** iff whenever there is a nontrivial FD $(A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m)$ for $R$, it is the case that $\{A_1, \ldots, A_n\}$ is a superkey for $R$

• **Theorem A1** (Codd, 1972)
  A database design has no anomalies due to FD iff all its relations are in Boyce-Codd Normal Form
How is BCNF violated here?

- A nontrivial FD:
  - Title, Year $\rightarrow$ Length, Film Type, Studio
  - The LHS not superset of the key \{Title, Year, Star\}
  $\Rightarrow$ Violate BCNF!

- Anomalies are due to FD’s whose LHS is not superkey

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>Film Type</th>
<th>Studio</th>
<th>Star</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Carrie Fisher</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Mark Hamill</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Mighty Ducks</td>
<td>1991</td>
<td>104</td>
<td>Color</td>
<td>Disney</td>
<td>Emilio Estevez</td>
</tr>
</tbody>
</table>

Towards a Better Design

- Use an offending FD $(A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m)$ to decompose $R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_h)$ into 2 tables
  - $R_1(A_1, \ldots, A_n, B_1, \ldots, B_m)$
  - $R_2(A_1, \ldots, A_n, C_1, \ldots, C_h)$

Wrong Movies

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>Film Type</th>
<th>Studio</th>
<th>Star</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Carrie Fisher</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Mark Hamill</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1997</td>
<td>124</td>
<td>Color</td>
<td>Fox</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>Mighty Ducks</td>
<td>1991</td>
<td>104</td>
<td>Color</td>
<td>Disney</td>
<td>Emilio Estevez</td>
</tr>
</tbody>
</table>
The “Invariant” Perspective

- The invariants:
  
  BCNF is an invariant of a good database design

- The lesson learned:

  Deliver a better database design by fixing violated invariants

Diagnosing Leukemias
Does Mr. A have cancer?

- Let's rearrange the rows…

- Does Mr. A have cancer?
and the columns too…

genes

samples

Mr. A: ???

Invariant Profile of Leukemia Subtypes
• What have we just seen?

• Guilt by association of invariants
Dissimilarity as Invariant

- Differences with other fruits are identical
- Differences between members of 2 different groups are constant

<table>
<thead>
<tr>
<th>Orange₁</th>
<th>Banana₂</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple₁</td>
<td>Color = red vs orange&lt;br&gt;Sk = smooth vs rough&lt;br&gt;Size = small vs small&lt;br&gt;Shape = round vs round</td>
<td>Color = red vs yellow&lt;br&gt;Sk = smooth vs smooth&lt;br&gt;Size = small vs small&lt;br&gt;Shape = round vs oblong</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orange₂</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color = orange vs orange&lt;br&gt;Sk = rough vs rough&lt;br&gt;Size = small vs small&lt;br&gt;Shape = round vs round</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unknown₁</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color = red vs orange&lt;br&gt;Sk = smooth vs rough&lt;br&gt;Size = small vs small&lt;br&gt;Shape = round vs round</td>
<td>Color = red vs yellow&lt;br&gt;Sk = smooth vs smooth&lt;br&gt;Size = small vs small&lt;br&gt;Shape = round vs oblong</td>
</tr>
</tbody>
</table>

The unknown is an APPLE !!!

What have we just seen?
- Guilt by association of invariants &
- Even differences can be invariant!
How to take exponentiation faster?

Exponentiation

• What does this program do?
  
  \[ F(a, 0) = 1 \]
  
  \[ F(a, n+1) = a \times F(a, n) \]

• We see that
  
  \[ F(a, n) = a^n \]
Playing the invariant...

- What does this program do?
  
  \[ F(a, 0) = 1 \]
  
  \[ F(a, n+1) = a \times F(a, n) \]

- We see that
  
  \[ F(a, n) = a^n \]

- Then
  
  \[ F(a, 2^n) = a^{2^n} = a^n \times a^n = y \times y \text{ where } y = F(a, n) \]

- \[ F(a, 2^n+1) = a^{2^n+1} = a \times a^n \times a^n = a \times y \times y \text{ where } F(a, n) \]

- So we get ...

What's the difference?

- Original program:
  
  \[ F(a, 0) = 1 \]
  
  \[ F(a, n+1) = a \times F(a, n) \]

- New program:
  
  \[ F(a, 0) = 1 \]
  
  \[ F(a, 1) = a \]
  
  \[ F(a, n) = \text{if } n \text{ is odd then } a \times y \times y \text{ else } y \times y \text{ where } y = F(a, n \text{ div } 2) \]

- Cost of \( F(a, n) = n \)

- Cost of \( F(a, n) = \log_2 n \)

<table>
<thead>
<tr>
<th>n</th>
<th>( \log n )</th>
<th>call sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>3</td>
<td>4 2 1</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>4 2 1</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>5 2 1</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>5 2 1</td>
</tr>
</tbody>
</table>

Parity can be tested by checking least significant bit

Div2 can be implemented by bit shifting
• What have we just seen?

• Optimizing a solution by preserving invariant
Raster Line Drawing

- Basic Bresenham's algo
  \[ \epsilon \leftarrow 0, \ y \leftarrow y_1 \]
  For \( z \leftarrow z_1 \) to \( z_2 \) do
    Plot point at \((x, y)\).
    If \( \epsilon + m < 0.5 \)
      \[ \epsilon \leftarrow \epsilon + m \]
    Else
      \[ y \leftarrow y + 1, \ \epsilon \leftarrow \epsilon + m - 1 \]
    EndIf
  EndFor

- Want to plot \( f(X) = m \cdot X + c \), where \( 0 \leq m \leq 1 \)
- At point \( x + 1 \), should we plot at \( y \) or at \( y + 1 \)?
- Choose whichever is closer to \( f(x + 1) \)

- \( m \) is a floating pt #.
- floating pt ops are expensive

\[ \text{Let } m = \frac{\Delta y}{\Delta x} \text{ and } \epsilon' = \epsilon \cdot \Delta x, \ldots \]

- \( \text{Then} \)
  \[ \epsilon + m < 0.5 \rightarrow 2(\epsilon' + \Delta y) < \Delta x \]
  \[ \epsilon_{\text{new}} = \epsilon + m \rightarrow \]
  \[ \epsilon'_{\text{new}} = \epsilon_{\text{new}} \cdot \Delta x \rightarrow \]
  \[ \epsilon'_{\text{new}} = \epsilon' + \Delta y \]
  \[ \epsilon_{\text{new}} = \epsilon + m - 1 \rightarrow \]
  \[ \epsilon'_{\text{new}} = \epsilon_{\text{new}} \cdot \Delta x \rightarrow \]
  \[ \epsilon'_{\text{new}} = \epsilon' + \Delta y - \Delta x \]
A much faster algo

- Basic Bresenham's algo
  
  $\epsilon \leftarrow 0, \quad y \leftarrow y_1$
  
  For $x \leftarrow x_1$ to $x_2$ do
  
  Plot point at $(x, y)$.
  
  If ($\epsilon + m < 0.5$)
  
  $\epsilon \leftarrow \epsilon + m$
  
  Else
  
  $y \leftarrow y + 1, \quad \epsilon \leftarrow \epsilon + m - 1$
  
  EndIf
  
  EndFor

- Integer Bresenham's algo
  
  $\epsilon' \leftarrow 0, \quad y \leftarrow y_1$
  
  For $x \leftarrow x_1$ to $x_2$ do
  
  Plot point at $(x, y)$.
  
  If ($2(\epsilon' + \Delta y) < \Delta x$)
  
  $\epsilon' \leftarrow \epsilon' + \Delta y$
  
  Else
  
  $y \leftarrow y + 1, \quad \epsilon' \leftarrow \epsilon' + \Delta y - \Delta x$
  
  EndIf
  
  EndFor

This works because $A$ is preserved by $B$, as $\epsilon + m < 0.5$ iff $2(\epsilon' + \Delta y) < \Delta x$
• What have we just seen?

• Optimizing a solution by preserving invariant

What have we learned?
What have we learned?

- Invariant is a fundamental property of many problems
- Paradigms of problem solving
  - Problem solving by logical reasoning on invariants
  - Problem solving by rectifying violation of invariants
  - Guilt by association of invariants
  - Solution optimization by preserving invariants

I didn’t get to telling you yet, but …

- Every time you write a loop in a program, it involves an invariant
- Every time you do a recursive function call, it involves an invariant
- Every time you do an induction proof, it involves an invariant
- … Computing is about discovering, understanding, exploiting, and having fun with invariants!