Programming by Analogy

Thomas Hieber
Lehrstuhl für Angewandte Informatik - Kognitive Systeme

3. Februar 2009
Inhaltsverzeichnis

1. Introduction
2. Validating Debugging
3. Inference
4. Abstraction
5. Left to Do
Analogy is (1) similarity in which the same relations hold between different domains or systems; (2) inference that if two things agree in certain respects then they probably agree in others. [...] Analogy is [...] central in the study of LEARNING and discovery.
Literature

- Nachum Dershowitz - *Programming by Analogy*
- Dedre Gentner - *Structure-Mapping: A Theoretical Framework for Analogy*
- Eva Wiese - *Mapping and Inference in Analogical Problem Solving - As Much As Needed or As Much As Possible?*
Validation & Debugging
Proceeding

- Validation & Debugging
- Inference
Proceeding

- Validation & Debugging
- Inference
- Abstraction
Proceeding

- Validation & Debugging
- Inference
- Abstraction
- [Instantiation]
Introduction
Validating
Debugging
Inference
Abstraction
Left to Do

Proceeding

- Validation & Debugging
- Inference
- Abstraction
- [Instantiation]
- [Extension]
Real Division

Real Division Specification

D1: begin comment real-division specification
assert $0 \leq c < d, e > 0$
achieve $|c/d - q| < e$ varying q
end
Real Division

Real Division Program 1

T1: begin comment suggested real-division program
B1: assert \(0 \leq c < d, e > 0\)
purpose \(|c/d - q| < e\)
purpose \(q \leq c/d < q + s, s \leq e\)
\((q, s) := (0, 1)\)
loop L1: suggest \(q \leq c/d < q + s\)
until \(s \leq e\)
purpose \(q \leq c/d < q + s, 0 < s < s[L1]\)
if \(d \ast (q + s) \leq c\) then \(q := q + s\) fi
s := s/2
repeat
suggest \(q < c/d < q + s, s \leq e\)
E1: suggest \(|c/d - q| < e\)
end
Real Division

Real Division Loop

loop L1: suggest $q \leq c/d < q + s$
until $s \leq e$
purpose $q \leq c/d < q + s$, $0 < s < s[L1]$
if $d \times (q + s) \leq c$ then $q := q + s$ fi
$s := s/2$
repeat
suggest $q < c/d < q + s$, $s \leq e$

$s = 1 \lor 2s > e$
Real Division

Real Division Loop

loop L1: suggest $q \leq c/d < q + s$
until $s \leq e$
purpose $q \leq c/d < q + s, 0 < s < s[L1]$
if $d \times (q + s) \leq c$ then $q := q + s$ fi
$s := s/2$
repeat
suggest $q < c/d < q + s, s \leq e$

- $s = 1 \lor 2s > e$
- $d \times q \leq c$
Real Division

Real Division Loop

\[
\text{loop L1: suggest } q \leq \frac{c}{d} < q + s \\
\text{until } s \leq e \\
\text{purpose } q \leq \frac{c}{d} < q + s, 0 < s < s[L1] \\
\text{if } d \times (q + s) \leq c \text{ then } q := q + s \text{ fi} \\
\text{s} := s / 2 \\
\text{repeat} \\
\text{suggest } q < \frac{c}{d} < q + s, s \leq e
\]

- $s = 1 \lor 2s > e$
- $d \times q \leq c$
- $c < d \times (q + 2s)$
Real Division

Real Division Loop

```
loop L1: suggest q ≤ c/d < q + s
until s ≤ e
purpose q ≤ c/d < q + s, 0 < s < s[L1]
if d*(q+s) ≤ c then q := q + s fi
s := s/2
repeat
suggest q < c/d < q + s, s ≤ e
```

- s = 1 ∨ 2s > e
- d * q <= c
- c < d*(q + 2s)

L1: assert d * q ≤ c, c < d * (q + 2s), s = 1 ∨ 2s > e
Real Division

Real Division Annotated

T1: begin comment suggested real-division program
B1: assert $0 \leq c < d, e > 0$
purpose $|c/d - q| < e$
purpose $q \leq c/d < q + s, s \leq e$
$(q, s) := (0, 1)$
loop L1: assert $d * q \leq c, c < d * (q + 2s), s = 1 \lor 2s > e$
suggest $c/d < q + s$
until $s \leq e$
purpose $q \leq c/d < q + s, 0 < s < s[L1]$
if $d * (q + s) \leq c$ then $q := q + sfi$
$s := s/2$
repeat
assert $q < c/d < q + 2s, s \leq e$
suggest $c/d - q + s$
E1: assert $|c/d - q| < 2e$
suggest $|c/d - q| < e$
end
**Suggested Analogy**

\[
|c/d - q| < 2e \implies |c/d - q| < e
\]

**Transformation**

\[
e \mapsto e/2
\]
Real Division

Real Division Transformed

\[(q, s) := (0, 1)\]
loop L2: assert \( d \cdot q \leq c, c < d \cdot (q + 2s), s = 1 \lor 2s > e \)
until \( s \leq e/2 \)
purpose \( q \leq c/d < q + 2s, 0 < s < s[L2] \)
if \( d \cdot (q + s) \leq c \) then \( q := q + s \) fi
\( s := s/2 \)
repeat
assert \( q < c/d < q + 2s, 2s \leq e \)

- \( s \Rightarrow s/2 \)
- \( s =: 1. \)
Real Division

Real Division Transformed

\[(q, s) := (0, 1)\]

loop L2: assert \[d \times q \leq c, c < d \times (q + 2s), s = 1 \lor 2s > e\]
until \[s \leq e/2\]

purpose \(q \leq c/d < q + 2s, 0 < s < s[L2]\)

if \(d \times (q + s) \leq c\) then \(q := q + s\) fi

\(s := s/2\)

repeat

assert \(q < c/d < q + 2s, 2s \leq e\)

- \(s \Rightarrow s/2\)
- \(s =: 1.\)
- achieve \(s/2 = 1\) varying \(s\)
Real Division

Real Division Transformed

\[(q, s) := (0, 1)\]
\[\text{loop L2: assert } d \times q \leq c, c < d \times (q + 2s), s = 1 \lor 2s > e\]
\[\text{until } s \leq e/2\]
\[\text{purpose } q \leq c/d < q + 2s, 0 < s < s[L2]\]
\[\text{if } d \times (q + s) \leq c \text{ then } q := q + s \text{ fi}\]
\[s := s/2\]
\[\text{repeat}\]
\[\text{assert } q < c/d < q + 2s, 2s \leq e\]

- \( s \Rightarrow s/2 \)
- \( s =: 1. \)
- achieve \( s/2 = 1 \) varying \( s \)
- \( \Rightarrow \) achieve \( s = 2 \) varying \( s \)
Real Division

Real Division Transformed

\[
(q, s) := (0, 1)
\]

\[
\text{loop L2: assert } d \cdot q \leq c, \ c < d \cdot (q + 2s), s = 1 \lor 2s > e
\]

\[
\text{until } s \leq e/2
\]

\[
\text{purpose } q \leq c/d < q + 2s, 0 < s < s[L2]
\]

\[
\text{if } d \cdot (q + s) \leq c \text{ then } q := q + s \text{ fi}
\]

\[
s := s/2
\]

\[
\text{repeat}
\]

\[
\text{assert } q < c/d < q + 2s, 2s \leq e
\]

• \( s \Rightarrow s/2 \)

• \( s =: 1. \)

• achieve \( s/2 = 1 \) varying \( s \)

• \( \Rightarrow \) achieve \( s = 2 \) varying \( s \)

• \( \Rightarrow s =: 2 \)
if \( d \times (q + s/2) \leq c \) then \( q := q + s/2 \) fi

\( s := s/2 \)
Real Division

if \( d \times (q + s/2) \leq c \) then \( q := q + s/2 \) fi

\( s := s/2 \)

\( s := s/2 \)

if \( d \times (q + s) \leq c \) then \( q := q + s \) fi
Real Division

Corrected Real Division Program

D2: begin comment real-division program
B2: assert 0 ≤ c < d, e > 0
purpose |c/d − q| < e
purpose q ≤ c/d < q + s, s ≤ e
(q, s) := (0, 2)
loop L2: assert d * q ≤ c, c < d * (q + 2s), s = 2 ∨ 2s > e
until s = e
purpose q ≤ c/d < q + s, 0 < s < s[L1]
s := s/2
if d * (q + s) ≤ c then q := q + s fi
repeat
assert q < c/d < q + 2s, s ≤ e
E2: assert |c/d − q| < e
end
Cube Root Specification

C3: begin comment cuberoot specification
assert $a \geq 0, e > 0$
achieve $|a^{1/3} - r| < e$ varying $r$
end
Cube Root Specification

C3: begin comment cuberoot specification
assert \( a \geq 0, e > 0 \)
achieve \( |a^{1/3} - r| < e \) varying \( r \)
end

Real Division Output Invariant

assert \( |c/d - q| < e \)
Cube Root

Cube Root Specification

C3: begin comment cuberoot specification
assert \( a \geq 0, e > 0 \)
achieve \( |a^{1/3} - r| < e \) varying \( r \)
end

Real Division Output Invariant

assert \( |c/d - q| < e \)

Analogy

\( q \Rightarrow r \)
\( c/d \Rightarrow a^{1/3} \)
Cube Root

Cube Root Specification

C3: begin comment cuberoot specification
assert a \geq 0, e > 0
achieve \left| a^{1/3} - r \right| < e varying r
end

Real Division Output Invariant

assert \left| c/d - q \right| < e

Analogy

q \Rightarrow r
c/d \Rightarrow a^{1/3}

Transformations

q \Rightarrow r
u/v \Rightarrow u^{1/3}
c \Rightarrow a
Cube Root

Transformation & Validation

...
Cube Root

Transformation & Validation
...

Cube Root Program
C3: begin comment cube-root program
B3 assert \( a > 0, e > 0 \)
\((r, s) := (0, a + 1)\)
loop L3: assert \( r \leq a^{1/3} < r + s \)
until \( s \leq e \)
s := s/2
if \( (r + s)^3 \leq a \) then \( r := r + s \) fi
repeat
E1: assert \(|a^{1/3} - r| < e\)
end

Thomas Hieber  Programming by Analogy
Abstraction

What we know:
- real division (D2)
- cube root calculation (C3)

Analog

\[
\begin{align*}
q & \iff r \\
\frac{u}{v} & \iff u^{1/3} \\
c & \iff a \\
u \ast v & \iff v^3
\end{align*}
\]
Abstraction

Transformations

\[ q \rightarrow z \]
\[ u/v \rightarrow \gamma(u, v) \]
\[ c \rightarrow x \]
\[ u \ast v \rightarrow \delta(u, v) \]
Abstraction

Transformations

- \( q \Rightarrow z \)
- \( u/v \Rightarrow \gamma(u, v) \)
- \( c \Rightarrow x \)
- \( u \ast v \Rightarrow \delta(u, v) \)

Applied to Cube Root Specification

- \( \text{achieve } |\gamma(x, d) - z| < e \) varying \( z \)
Abstraction

Transformations

\[ q \Rightarrow z \]
\[ u/v \Rightarrow \gamma(u, v) \]
\[ c \Rightarrow x \]
\[ u \cdot v \Rightarrow \delta(u, v) \]

Applied to Cube Root Specification

\[ \text{achieve } |\gamma(x, d) - z| < e \text{ varying } z \]

Applied to Loop Invariant

\[ \delta(d, z) \leq x, x < \delta(d, z + s) \]
Abstraction

Transformations
\[ q \Rightarrow z \]
\[ u/v \Rightarrow \gamma(u, v) \]
\[ c \Rightarrow x \]
\[ u \ast v \Rightarrow \delta(u, v) \]

Applied to Cube Root Specification
achieve \(|\gamma(x, d) - z| < e\) \text{ varying } z

Applied to Loop Invariant
\[ \delta(d, z) \leq x, x < \delta(d, z + s) \]

Problematic Initialization
initialization: \((z, s) := (0, 2)\)
evaluation: \(\delta(d, 0) \leq x, x < \delta(d, 2)\)
New Subgoal

\[ \text{achieve } \delta(d, z) \leq x, x < \delta(d, z + s) \text{ varying } z, s \]
**New Subgoal**

\[
\text{achieve } \delta(d, z) \leq x, x < \delta(d, z + s) \text{ varying } z, s
\]

**Relation Between \( \delta \) and \( \gamma \)**

\[
\delta(w, u) \leq v \equiv u \leq \gamma(v, w)
\]
Abstraction

New Subgoal

achieve $\delta(d, z) \leq x, x < \delta(d, z + s)$ varying $z, s$

Relation Between $\delta$ and $\gamma$

$\delta(w, u) \leq v \equiv u \leq \gamma(v, w)$

Schema for Binary Search

S6: begin comment binary-search schema
B6: assert $e > 0, \delta(w, u) \leq v \equiv u \leq \gamma(v, w)$
achieve $\delta(d, z) \leq x, x < \delta(d, z + s)$ varying $z, s.$
loop L6: assert $\delta(d, z) \leq x, x < \delta(d, z + s)$
until $s \leq e$
$s := s/2$
if $\delta(d, z + s) \leq x$ then $z := z + s$ fi
repeat
E2: assert $|\gamma(x, d) - z| < e$
end
**Further Steps**

- **Instantiation** takes the derived schema of *binary-search* and tries to find an analogy to a new problem specification (here *integer square root*) and infer a working program which also uses *binary-search* as core technique.

- **Extension** is a necessary discipline as soon as a suggested program still doesn’t solve a problem after all possible transformations have been applied. This means, that now the algorithm of the program itself has to be modified, in Dershowitz’ example it is the insertion of a second loop.