Programming by Analogy

Thomas Hieber

Lehrstuhl für Angewandte Informatik - Kognitive Systeme

3. Februar 2009

Inhaltsverzeichnis

Analogy

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?*

Proceeding

- Validation & Debugging

Proceeding

- Validation & Debugging
- Inference

Proceeding

- Validation & Debugging
- Inference
- Abstraction

Proceeding

- Validation & Debugging
- Inference
- Abstraction
- [Instantiation]

Proceeding

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

Real Division

Real Division Specification

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

Real Division

Real Division Program 1

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

Real Division

Real Division Loop

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

- $s = 1 \vee 2s > e$

Real Division

Real Division Loop

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

- $s = 1 \lor 2s > e$
- $d * q <= c$

Real Division

Real Division Loop

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

- $s = 1 \vee 2s > e$
- $d * q <= c$
- $c < d * (q + 2s)$

Real Division

Real Division Loop

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

- $s = 1 \vee 2s > e$
- $d * q <= c$
- $c < d * (q + 2s)$

L1: assert $d * q \leqq c, c < d * (q + 2s), s = 1 \vee 2s > e$

Real Division

Real Division Annotated

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

Real Division

Suggested Analogy

$|c/d - q| < 2e => |c/d - q| < e$

Transformation

$e \Rightarrow e/2$

Real Division

Real Division Transformed

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

- $s \Rightarrow s/2$
- $s =: 1.$

Real Division

Real Division Transformed

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

- $s \Rightarrow s/2$
- $s =: 1.$
- achieve $s/2 = 1$ varying s

Real Division

Real Division Transformed

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

- $s \Rightarrow s/2$
- $s =: 1.$
- achieve $s/2 = 1$ varying s
- $\Rightarrow$ achieve $s = 2$ varying s

Real Division

Real Division Transformed

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

- $s \Rightarrow s/2$
- $s =: 1.$
- achieve $s/2 = 1$ varying s
- $\Rightarrow$ achieve $s = 2$ varying s
- $\Rightarrow s =: 2$

Real Division

$\texttt{if}\ d*(q+s/2) \leqq c\ \texttt{then}\ q := q+s/2\ \texttt{fi}$
$s := s/2$

Real Division

$\texttt{if}\ d*(q+s/2) \leqq c\ \texttt{then}\ q := q+s/2\ \texttt{fi}$
$s := s/2$

$s := s/2$
$\texttt{if}\ d*(q+s) \leqq c\ \texttt{then}\ q := q+s\ \texttt{fi}$

Real Division

Corrected Real Division Program

D2: begin comment real-division program
B2: assert $0 \leqq c < d, e > 0$
purpose $|c/d - q| < e$
purpose $q \leqq c/d < q + s, s \leqq e$
$(q, s) := (0, 2)$
loop L2: assert $d * q \leqq c, c < d * (q + 2s), s = 2 \vee 2s > e$
until s ¡= e
purpose $q \leqq c/d < q + s, 0 < s < s[L1]$
$s := s/2$
if $d * (q + s) \leqq c$ then $q := q + s$ fi
repeat
assert $q < c/d < q + 2s, s \leqq e$
E2: assert $|c/d - q| < e$
end

Cube Root

Cube Root Specification

```
C3: begin comment cuberoot specification
assert a ≧ 0, e > 0
achieve |a^(1/3) − r| < e varying r
end
```

Cube Root

Cube Root Specification

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

Real Division Output Invariant

assert $|c/d - q| < e$

Analogy

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

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 $r \leqq a^{1/3} < r + s$
until $s \leqq e$
$s := s/2$
if $(r + s)^3 \leqq \ a$ then $r := r + s$ fi
repeat
E1: assert $|a^{1/3} - r| < e$
end

Abstraction

What we know:

- real division (D2)
- cube root calculation (C3)

Anlogy

$q \Leftrightarrow r$
$u/v \Leftrightarrow u^{1/3}$
$c \Leftrightarrow a$
$u * v \Leftrightarrow v^3$

Abstraction

Transformations

$q \Rightarrow z$
$u/v \Rightarrow \gamma(u, v)$
$c \Rightarrow x$
$u * v \Rightarrow \delta(u, v)$

Abstraction

Transformations

$q \Rightarrow z$
$u/v \Rightarrow \gamma(u, v)$
$c \Rightarrow x$
$u * v \Rightarrow \delta(u, v)$

Applied to Cube Root Specification

`achieve` $|\gamma(x, d) - z| < e$ `varying z`

Abstraction

Transformations

$q \Rightarrow z$
$u/v \Rightarrow \gamma(u, v)$
$c \Rightarrow x$
$u * v \Rightarrow \delta(u, v)$

Applied to Cube Root Specification

achieve $|\gamma(x, d) - z| < e$ varying z

Applied to Loop Invariant

$\delta(d, z) \leqq x, x < \delta(d, z + s)$

Abstraction

Transformations

$q \Rightarrow z$
$u/v \Rightarrow \gamma(u, v)$
$c \Rightarrow x$
$u * v \Rightarrow \delta(u, v)$

Applied to Cube Root Specification

`achieve` $|\gamma(x, d) - z| < e$ `varying z`

Applied to Loop Invariant

$\delta(d, z) \leqq x, x < \delta(d, z + s)$

Problematic Initialization

`initialization:` $(z, s) := (0, 2)$
`evaluation:` $\delta(d, 0) \leqq x, x < \delta(d, 2)$

Abstraction

New Subgoal

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

Abstraction

New Subgoal

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

Relation Between δ and γ

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

Abstraction

New Subgoal

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

Relation Between δ and γ

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

Schema for Binary Search

S6: begin comment binary-search schema
B6: assert $e > 0, \delta(w,u) \leqq v \equiv u \leqq \gamma(v,w)$
achieve $\delta(d,z) \leqq x, x < \delta(d,z+s)$ varying z,s.
loop L6: assert $\delta(d,z) \leqq x, x < \delta(d,z+s)$
until s <= e
$s := s/2$
if $\delta(d,z+s) \leqq x$ 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.