

A relation algebraic semantics of reversible loop programs

2007.7.3

Dependable System Workshop at
Oonuma

Yoshiki Kinoshita

The inverse of a program

- Wouldn't it be nice to be able to run a program backwards, or, better yet, to derive from one program P a second program P^{-1} that computes the inverse of P ? (Gries)

$$(p', y) \text{ ---A---} \rightarrow (p, y') \quad (p, y') \text{ ---B---} \rightarrow (p', y)$$

- If A transforms p into a (standard) value which is its initial value in B , and if B transforms y into a (standard) value which is its initial value in A , then transformations A and B are inverse transformations on the pair (p, y) . (Dijkstra)

$$(p_0, y) \text{ ---A---} \rightarrow (p, y') \quad (p, y_0) \text{ ---B---} \rightarrow (p', y)$$

$$y \text{ ---} \rightarrow (p_0, y) \text{ ---A---} \rightarrow (p, y') \text{ ---} \rightarrow (p, y_0) \text{ ---B---} \rightarrow (p', y) \text{ ---} \rightarrow y$$

$$p \text{ ---} \rightarrow (p, y_0) \text{ ---B---} \rightarrow (p', y) \text{ ---} \rightarrow (p_0, y) \text{ ---A---} \rightarrow (p, y') \text{ ---} \rightarrow p$$

Reversible computation

- R. Landauer
 - “Invertibility and heat generation in the computing process”, IBM J. Res. Dev. vol.5 No. 3 (1961)
- Bennett
 - “Logical reversibility of computation”, IBM J. Res. Dev. vol. 17 pp. 525-532 (1973)
- Feynman
 - “Feynman lectures on computation”, Tony Hey and R. W. Allen (eds.), Westview, ISBN 0-7382-0296-7, 1996.

Program Inversion

- E.D. Dijkstra
 - EWD 671: Program inversion
<http://www.cs.utexas.edu/users/EWD/ewd06xx/EWD671.PDF>
- David Gries

Inverting Programs, Chapter 21 of "The Science of Programming", Springer-Verlag, ISBN 0-387-90641-X and 3-540-90641-X, 1981.

Reversible Programming

- Emulate a Turing machine by a reversible (3-tape) Turing machine (Bennett)
- Structured reversible programming (Glück-Yokoyama-Axelsen)
Programming techniques in reversible loop commands, reversible machine language.
Emulate a Turing machine by a reversible loop commands (sketch).

Reversible Programming

- [YAG] Reversible Machine Code and its Abstract Processor Architecture
 - In Diekert V., Volkov M., Voronkov A. (eds.), *Computer Science - Theory and Applications. Proceedings.* Springer-Verlag 2007.
- [YG] A Reversible Programming Language and its Invertible Self-Interpreter
 - In *Partial Evaluation and Program Manipulation. Proceedings*, pp. 144-153. ACM Press. 2007.
- [YG] Irreversible Functions and Unbounded Sized Data in a Reversible Programming Language
 - In The 9th JSSST Workshop on Programming and Programming Languages (PPL 2007), 14 pages, March 2007.
- [AGY] (Reversible Flowchart etc.)

General interests

- **From physics:** energy needed for computation?
Ideally, no energy needed for reversible computation! (Landauer, Bennett, Feynman)
- **From programming:** inverse problems is much easier, sometimes.
Dijkstra (EWD671): `` We are interested in these inverse transformations because in general program A is regarded as easier than B: we have solved problem B as soon as we have for A a reversible solution!''

Reversible loop commands

[Essentially due to Yokoyama and Glück]

Given a set A (of atomic commands) and E (of conditional expressions), the abstract syntax of reversible loop commands (RLC) C over A and E is defined as follows:

$$C ::= A \mid A^{-1} \mid C ; C \mid \text{if } E \text{ then } C \square C \text{ neht } E \text{ fi} \mid \text{loop } E \text{ do } C \square C \text{ od pool}$$

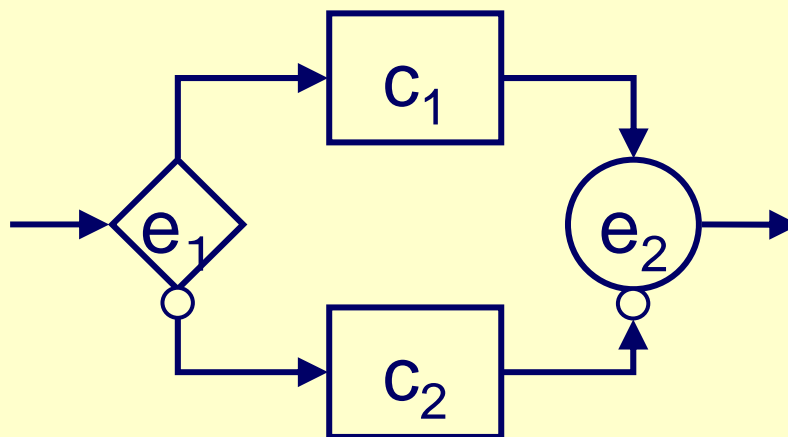
Intuitive meaning of if

if e_1 then c_1 \square c_2 neht e_2 fi

If e_1 holds, execute c_1 ; if c_1 terminates successfully, assert e_2 , i.e., if e_2 does not hold, abort.

If e_1 does not hold, execute c_2 ; if c_2 terminates successfully, assert $\sim e_2$.

Flow diagram à la
Yokoyama-Alexen-
Glück

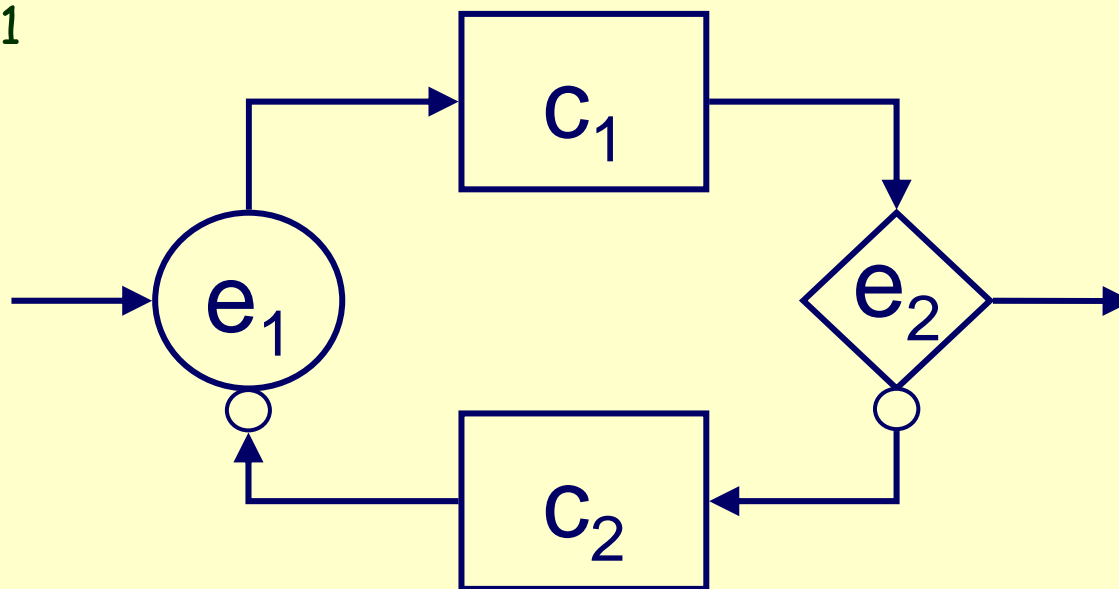


Intuitive meaning of loop

loop e_1 do $c_1 \square c_2$ od e_2 pool

If e_1 holds, execute c_1 ; then, while e_2 does not hold, repeat "execute c_2 ; assert $\sim e_1$; execute c_1 "

Flow diagram à la
Yokoyama-Alexen-
Glück



A rel. alg. semantics of RLC

The interpretation of RLC
with the set A of atomics and
the set E of conditional expressions
in a relation algebra $(B, 1, \sim, \cdot, ;, id, \circ)$
is a map $I: A+E \rightarrow B$, where each element
of $I(A)$ is injective and univalent, and
each element of $I(E)$ is less than id.

NB. for $e \in E$, $I(e)^{-1} = I(e)$

A rel. alg. semantics of RLC

Define $[-]$, an extension of I to the whole RLC, as follows.

- $[\text{skip}] = \text{id}$
- $[\text{abort}] = 0$ ($:= \sim 1$)
- If $a \in A$, $[a] = I(a)$, and if $e \in E$, $[e] = I(e)$.
- For $a \in A$, $[a^{-1}] = [a]^{\circ}$
- For $c_1, c_2: \text{RLC}$, $[c_1 ; c_2] = [c_1] ; [c_2]$

A rel. alg. semantics of RLC

- For c_1, c_2 : RLC, e_1, e_2 : cond., [if e_1 then c_1 \square c_2 neht e_2 fi] = $[e_1];[c_1];[e_2] + \sim[e_1];[c_2];\sim[e_2]$
- For c_1, c_2 : RLC, e_1, e_2 : cond., [loop e_1 do c_1 \square c_2 od e_2 pool] = $[e_1];[c_1]; (\sim[e_2];[c_2];\sim[e_1];[c_1])^*;[e_2]$

Inverse

For extend $(-)^{-1}$ to arbitrary command c ,
by induction on the construction of c .

$$(c_1 ; c_2)^{-1} = c_2^{-1} ; c_1^{-1}$$

$$\begin{aligned} &(\text{if } e_1 \text{ then } c_1 \square c_2 \text{ neht } e_2 \text{ fi})^{-1} \\ &= \text{if } e_2 \text{ then } c_1^{-1} \square c_2^{-1} \text{ neht } e_1 \text{ fi} \end{aligned}$$

$$\begin{aligned} &(\text{loop } e_1 \text{ do } c_1 \square c_2 \text{ od } e_2 \text{ pool})^{-1} \\ &= \text{loop } e_2 \text{ do } c_1^{-1} \square c_2^{-1} \text{ od } e_1 \text{ pool} \end{aligned}$$

Proposition. $[c^{-1}] = [c]^{\circ}$

∴ the loop case:

$$\begin{aligned}
 & [\text{loop } e_1 \text{ do } c_1 \square c_2 \text{ od } e_2 \text{ pool}]^{\circ} \\
 &= ([e_1]; [c_1]; (\sim[e_2]; [c_2]; \sim[e_1]; [c_1])^*; [e_2])^{\circ} \\
 &= [e_2]^{\circ}; ([c_1]^{\circ}; \sim[e_1]^{\circ}; [c_2]^{\circ}; \sim[e_2]^{\circ})^*; [c_1]^{\circ}; [e_1]^{\circ} \\
 &\quad \because (x^*)^{\circ} = (x^{\circ})^*, (x;y)^{\circ} = y^{\circ}; x^{\circ} \\
 &= [e_2]; ([c_1]; \sim[e_1]; [c_2]; \sim[e_2])^*; [c_1]; [e_1] \\
 &\quad \because [e]^{\circ} = [e], \text{ for } e \in E \\
 &= [e_2]; [c_1]; (\sim[e_1]; [c_2]; \sim[e_2]; [c_1])^*; [e_1] \\
 &\quad \because (x;y)^*; x = x;(y;x)^* \\
 &= [\text{loop } e_2 \text{ do } c_1 \square c_2 \text{ od } e_1 \text{ pool}] \\
 &= [(\text{loop } e_1 \text{ do } c_1 \square c_2 \text{ od } e_2 \text{ pool})^{-1}]
 \end{aligned}$$

My further interests & imaginations

- Problem: Emulate (usual) while commands by reversible loop commands? (Cf. Bennett)
- Problem: (Essentially) algebraic structure for reversible loop commands?
- Imagination: Uninstalling for free
- Imagination: Debugger w/o breakpoints

FIN