Introduction and Foundations of Program Inversion

Robert Glück

University of Copenhagen

Today's Plan

- Inversion: a basic operation on programs
- Short review: inversion in mathematics
  - Relations, functions
  - Applications in science & technology
- The programming world
  - Inverse interpreter
  - Program inverter
  - and their existence

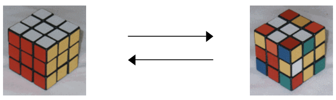
Review: Inverse Computation

- Standard Computation:

  Calculation of the output of a program for a given input ('forward execution')

- Inverse Computation:

  Calculation of the possible input of a program for a given output ('backward execution')



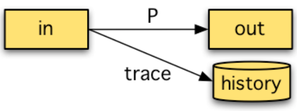
Manifestations: Underlying Principles and Relation?

Software
- **Undo Operations:** Word, Emacs, ...
- **Tracing:** debugging, simulation, ...
- **Translation:** decompilation, unparsing, ...
- **AI:** problem solving, logic reasoning, ...

Hardware
- **Reversible hardware:** reduce heat dissipation, quantum computer, low-power CMOS, ...
- **Reversible languages:** for reversible hardware, ...

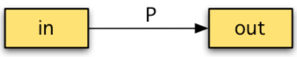
Two Types of Inversion

- Tracing:



Bennett:73, ... Carothers at al.99
apps: Word, Excel, ...
- usually *ad hoc* inversion
- partial trace of non-invertibles
- size of trace ≈ length of comp.
- first forward, then backward
- instrumented forward prog.
- synchronized with inverse prog.

- Stand-alone:



Dijkstra:78, Gries:81, manual;
MuBird:02, conv-of-fct-theorem;
automatic: Korf&Eppstein:85
automatic: *one of our goals!*

A Common Operation on Programs: Inversion

- Fundamental concept in mathematics
- Received little attention in computer science
- Inverse programs frequently used. A familiar example:

- State of the art: inverse programs are written manually
- Motivation: "Get two programs from one program."

Programs as Data Objects (review)

1. Three Basic Operations

- Specialization of programs (A)
- Composition of programs (B)
- Inversion of programs (C)

2. Layers of these Operations

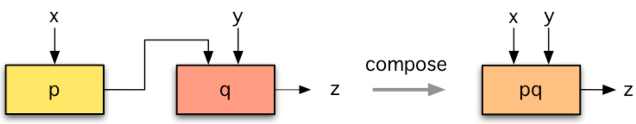
- Self-application (*e.g.* Futamura projections)
- Program generators (*e.g.* Ershov generating extensions)
- Metasystem transition schemes (by Turchin)

Operation A: Program Specializer

- faster program
- often shorter program
- most advanced operation

Example: partial evaluation [Jones et al. '93]

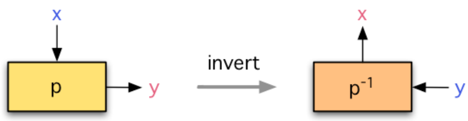
Operation B: Program Composer



- faster program
- rm interface operations
- rm redundant computations
- reduce memory

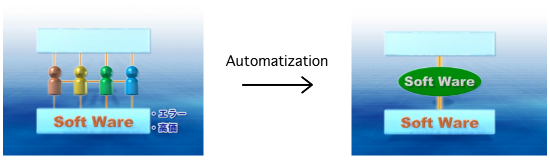
Example technique: deforestation [Wadler'90]

Operation C: Program Inverter



- "two for the price of one"
- example: encode / decode text

Approach: "Programs that produce Programs"



Build <u>programs</u> that treat <u>programs</u> as data objects:

- Automatically manipulate programs by programs
- Analyse & transform programs by programs

▲ Programs are semantically the **most complex form** of data objects in the computer.

Motivation: Industrial Principle

Despite tremendous progress in <u>hardware</u>,
the production of <u>software</u> is

- manual
- error prone
- costly

Recently, exploding demand for software led to

- dramatic lack of skilled programmers
- low software quality

⇨ **Business as usual is not an option in the long run.**
⇨ **Methods for automatized SW production needed.**
<u>Important goal for Computer Science research.</u>

Inverse Problems in Science and Engineering (1)

- Example: Given a math. model of how electromagnetic waves are reflected by an object, solve the inverse problem:

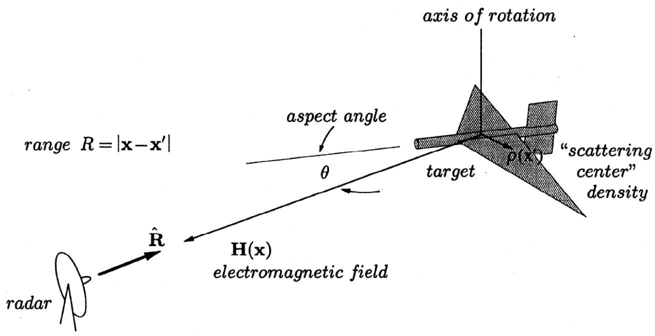
  What are the object attributes (e.g. shape, position), given the intensity of the electromagnetic waves scattered by it?

- Inverse problems appear frequently – for instance in

  medical imaging, computer tomography,
  radar recognition, seismic exploration, etc.

- A branch of mathematics deals with finding mathematical solutions to inverse problems in science and engineering. Continuous and discrete solutions, accurracy, etc.

Inverse Problems in Science and Engineering (2)



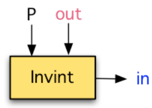
[Borden:97]

Programming World

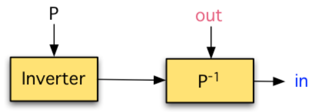
1. A program is the implementation of a function in a programming language: many variants!
2. In general, programs may not terminate, they are only partially defined.
3. Even if an efficient inverse program p^{-1} is known to exist, it may be hard or impossible to derive it automatically.
4. A trivial inverse program p^{-1} of p can always be generated.
5. Program properties: consumption of time, space, energy...

Two Approaches to Inversion of Program P in ⇒ out

- Inverse interpreter [McCarthy'56 ...]:



- Program inverter [Dijkstra'78, Gries'81...]:



- Related by Futamura projections [AbramovGlück'98]:
  The generating extension of an inverse interpreter is a pgm inverter.

Research Results

1. Algorithm for Inverse Interpreter

   Example: Universal Resolving Algorithm (URA), McCarthy's Generate-and-Test algorithm.

2. Algorithms for Program Inversion

   Example: Program inverter for functional language.

3. Transform Inverse Interpreter into Program Inverter

   Example: Self-applicable partial evaluator turns Inverse interpreter into program inverter.

State-of-the-Art

Inverse Computation:

- McCarthy '56
- logic programming '74
- ...
- Abramov, Glück '00

Program inversion:

- Dijkstra '78
- Gries '81
- ...
- Mu, Bird '02

Given the *fundamental importance* of program inversion, surprisingly few works have been devoted to this topic.

Status: little known about manual & automatic pgm inversion. state-of-the-art is rather modest (after 1/2 century CS).

Inverse Interpreter

- **Definition:** An L-program invint is an inverse interpreter for injective N-programs iff ∀injective p, ∀x,y:

  $[\text{invint}]_L(p, y) = x \quad \Leftrightarrow \quad [p]_N\ x = y$

- **Example**: let encode be a program to encode text:

  $[\text{encode}]_N$ text = code

  Instead of writing a decoder to reproduce the text:

  $[\text{decode}]_N$ code = text

  Use an inverse interpreter to obtain the same result:

  $[\text{invint}]_L$(encode, code) = text

Existence of Inverse Interpreter: Generate & Test

```
input: program p, output out
n ← const; k ← const; unmark all possible inputs;
loop
   B ← the first n possible inputs that are unmarked
   for all b ∈ B do
      compute ⟦p⟧ b (with maximally k*n steps)
      x ← result of computation
      if x is a value then
         mark b
         if x = out then print b endif
      endif
   endfor
   increment n
endloop
```


Assessement: Inverse Interpreter w/Generate & Test

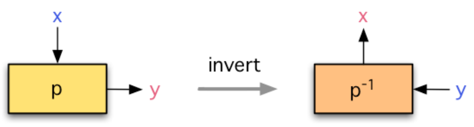
The algorithm is
- correct: finds all possible inputs for a given output.
- *very slow*: tries all possible inputs, many times, even if they can never lead to the desired result, combinatorial explosion!

BUT: sufficient to show the existence of inverse programs and program inverters in a Turing-complete programming language.

Trivial Inverse Program

Task: Given a program p and an inverse interpreter invint, define an inverse program p^{-1} in one line.
(Hint: There exists a trivial definition of the inverse program.)

Program Inverter



$[p]\ x = y \quad \Leftrightarrow \quad [p^{-1}]\ y = x \qquad$ *where* $p^{-1} = [\text{inverter}]\ p$
p *injective*

Trivial Program Inverter

Task: Given an inverse interpreter, define a program inverter.
(Hint: There exists a trivial definition of the program inverter.)

Trivial Inverse Interpreter

Task: Given a program inverter, define an inverse interpreter.
(Hint: There exists a trivial definition of the inverse interpreter.)

Summary: Existence of Inverse Programs and Tools

1. Existence of inverse program:
   For every program p written in a universal programming language L, there exists an inverse program p^{-1} written in L.

2. Existence of the inversion tools:
   For every universal programming language L, there exists an inverse interpreter and a program inverter for L written in L.

   For every program p written in L, one can perform inverse computation of p and invert program p.

   Remark: universal = Turing-complete.

Inverse Interpreter: Universal Resolving Algorithm (URA)

- Algorithm for inverse computation.
- First-order functional programming language.
- Computes universal solution for non-injective programs.
- Implemented in Scheme and Haskell.

Abramov, Glück: The universal resolving algorithm and its correctness: inverse computation in a functional language.
In: *Science of Computer Programming*, 43(2-3): 193-229, 2002.

Example: Inverse Computation of 'Inorder Traversal'

Source Program:

$$[\![inorder]\!]_{TSG}\ t_i = [7,6,5,4,3,2,1]$$

Input-Output:

$$cls_{io} = \langle ([Xe_1], [7,6,5,4,3,2,1]), \emptyset \rangle$$

Inverse Computation:

$$\begin{aligned}[\![ura]\!]_{SCM}\ [inorder, cls_{io}] = [\ & \\ & ([Xe_1 \mapsto ((1{:}2{:}3){:}4{:}5){:}6{:}7],\ \emptyset), \\ & ([Xe_1 \mapsto (1{:}2{:}3{:}4{:}5){:}6{:}7],\ \emptyset), \\ & ([Xe_1 \mapsto (1{:}2{:}3){:}4{:}5{:}6{:}7],\ \emptyset), \\ & ([Xe_1 \mapsto 1{:}2{:}(3{:}4{:}5){:}6{:}7],\ \emptyset), \\ & ([Xe_1 \mapsto 1{:}2{:}3{:}4{:}5{:}6{:}7],\ \emptyset),\ \ldots\end{aligned}$$

Logic Programming

Source Program:

```
append([], Ys, Ys).
append([X|Xs], Ys, [X|Zs]) :- append(Xs, Ys, Zs)
```

Standard Computation:
append([1,2,3], [4,5], Out)

Solution:
Out=[1,2,3,4,5]

Inverse Computation:
append(In1, In2, [1,2,3,4,5])

Solution:
In1=[], In2=[1,2,3,4,5]
In1=[1], In2=[2,3,4,5]
...
In1=[1,2,3,4,5], In2=[]

Technical Remark: Format for Program Inversion (1)

1. Single Output, Single Input

$$f(x) = y \quad \Leftrightarrow \quad x = f^{-1}(y)$$

2. Multiple Outputs, Multiple Inputs

$$\begin{matrix} f1(x1,\ldots,xm) = y1 & & x1 = f1^{-1}(y1,\ldots,yn) \\ \ldots & \Leftrightarrow & \ldots \\ fn(x1,\ldots,xm) = yn & & xm = fm^{-1}(y1,\ldots,yn) \end{matrix}$$

two injective systems of functions
where $fi{:}A \to B$, $fj^{-1}{:}B \to A$

Technical Remark: Format for Program Inversion (2)

a. Tupled

f([x1,...,xm]) = [y1,...,yn] ⇔ [x1,...,xm] = f^{-1}([y1,...,yn])

b. Untupled

f1(x1,...,xm) = y1 ⇔ x1 = $f1^{-1}$(y1,...,yn)
...
fn(x1,...,xm) = yn ⇔ xm = fm^{-1}(y1,...,yn)

injective system of functions where fi:A→B, fj^{-1}:B→A

Example for Program Inversion

- Function System

```
(take 3 '(A B C D E)) = (A B C)
(drop 3 '(A B C D E)) = (D E)

          (A B C D E) = (append '(A B C) '(D E))
                    3 = (length '(A B C) '(D E))
```

- Definition in Lisp-like source language

```
((take drop) (n xs)                       ; def.header
   (if (zerop n) nil                      ; take
       (cons (car xs)                     ; take
             (take (sub1 n) (cdr xs))))   ; take
   (if (zerop n) xs                       ; drop
       (drop (sub1 n) (cdr xs))))         ; drop
```


More Primitive Operators

Numbers:

(add x1 x2) = y1 ⇔ x1 = (sub y1 y2)
x2 = y2 ⇔ x2 = y2

(sub x1 x2) = y1 ⇔ x1 = (add y1 y2)
x2 = y2 ⇔ x2 = y2

(mul x1 x2) = y1 ⇔ x1 = (div y1 y2)
x2 = y2 ⇔ x2 = y2

(div x1 x2) = y1 ⇔ x1 = (mul y1 y2)
x2 = y2 ⇔ x2 = y2

Invertible in Theory and Practice?

- Limits of numerical representations

(div x1 x2) = y1 ⇔ x1 = (mul y1 y2)
x2 = y2 ⇔ x2 = y2

Example:

(div 1 3) = 0.333 ⇔ 0.999 = (mul 0.333 3)
3 = 3 ⇔ 3 = 3

1 ≠ 0.999