The Structure of a Program Inverter

Robert Glück[1]

and Masahiko Kawabe[2]

Dagstuhl IFIP WG 2.11

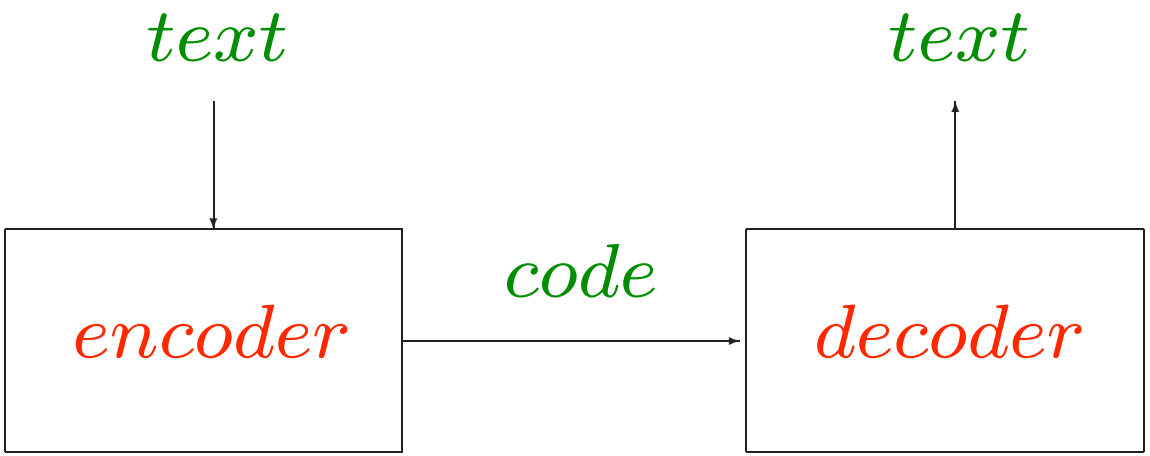
January 27, 2006

[1] University of Copenhagen, Denmark

[2] Waseda University, Tokyo, Japan

Inverse Programs

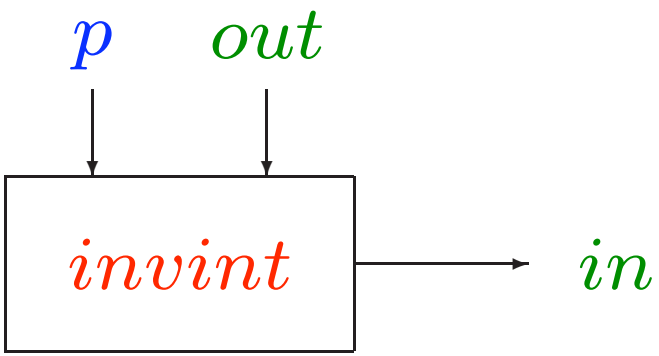
- Perhaps the most common example of two programs that are inverse to each other:



- Examples: `zip/unzip`, `uuencode/uudecode`,...

Inverse Interpreter

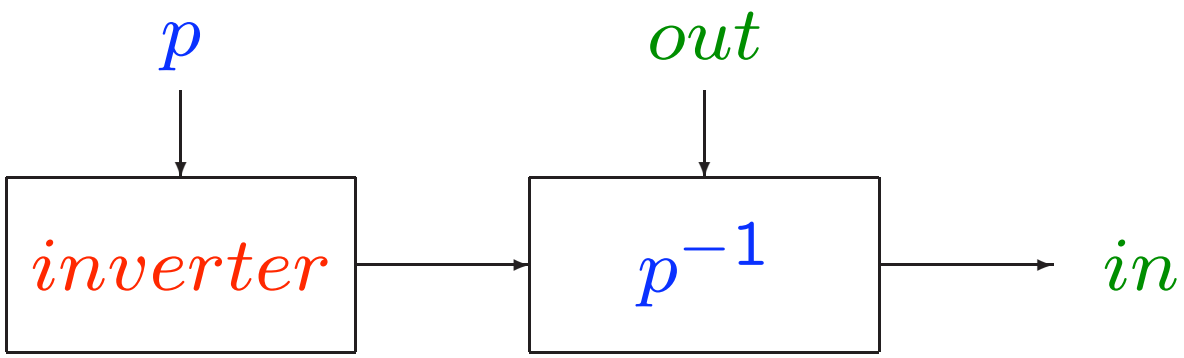
- Inverse Interpreter $invint$:
  - Let $p(in) = out$. Given program p and output out, compute the possible input in:



- A logic programming system (*e.g.*, Prolog).
- For functional languages, the Universal Resolving Algorithm [AbramovGlück02]

Our Approach: Program Inverter

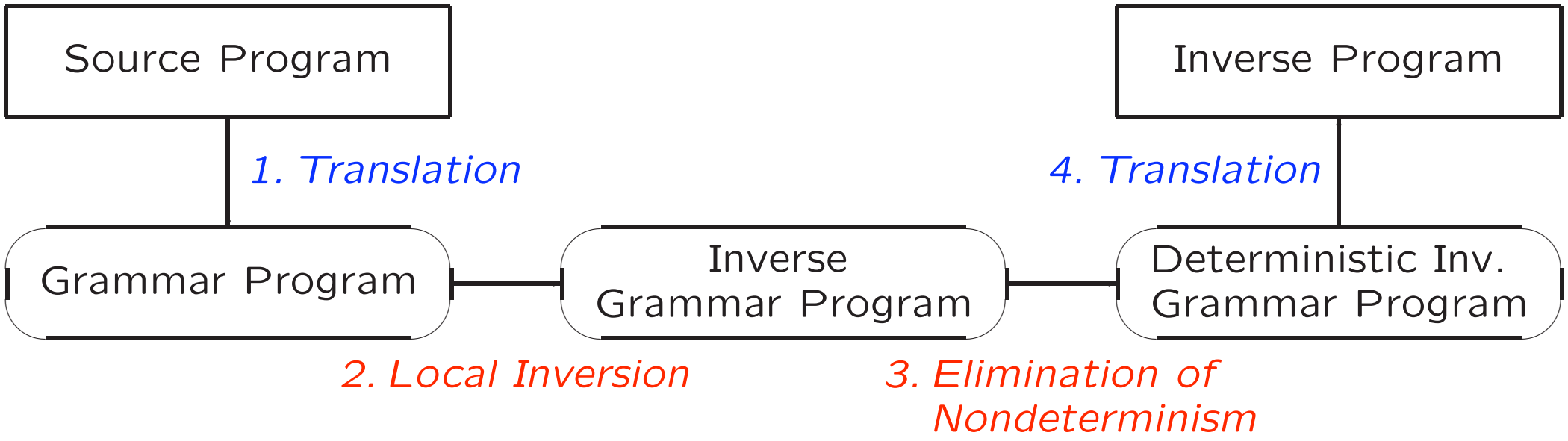
- Program Inverter $inverter$:
  - Given program p, generate an inverse program p^{-1}.
  - Inverse program p^{-1} computes an input in from a given output out:



- Challenge: automatic generation of deterministic and efficient inverse programs from injective programs.

Structure of Our Program Inverter

- Steps for program inversion
  1. Translation to grammar language.
  2. Local inversion.
  3. Elimination of nondeterminism.
  4. Translation to source language.



Example: Increment a Binary Number

- Source Program:

$$
\begin{array}{l}
inc(x) \triangleq \ \mathbf{case}\ x\ \mathbf{of} \\
\qquad\qquad 1 \longrightarrow 0{:}1 \\
\qquad\qquad x_1{:}xs \longrightarrow \mathbf{case}\ x_1\ \mathbf{of}\ \ 0 \longrightarrow 1{:}xs \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad 1 \longrightarrow 0{:}inc(xs)
\end{array}
$$

- Binary numbers are represented by improper lists of digits in reverse order: $110 \Rightarrow (0\ 1\ .\ 1)$

- Goal: generate a decrement program dec.

Well-formed for Inversion

- We require: Every defined variable is used.

  Non-example:

  $$fst(xs) \triangleq \mathbf{case}\ xs\ \mathbf{of}\ y{:}ys \longrightarrow y$$

  Programs that never discard values
  are called well-formed for inversion.

Translation

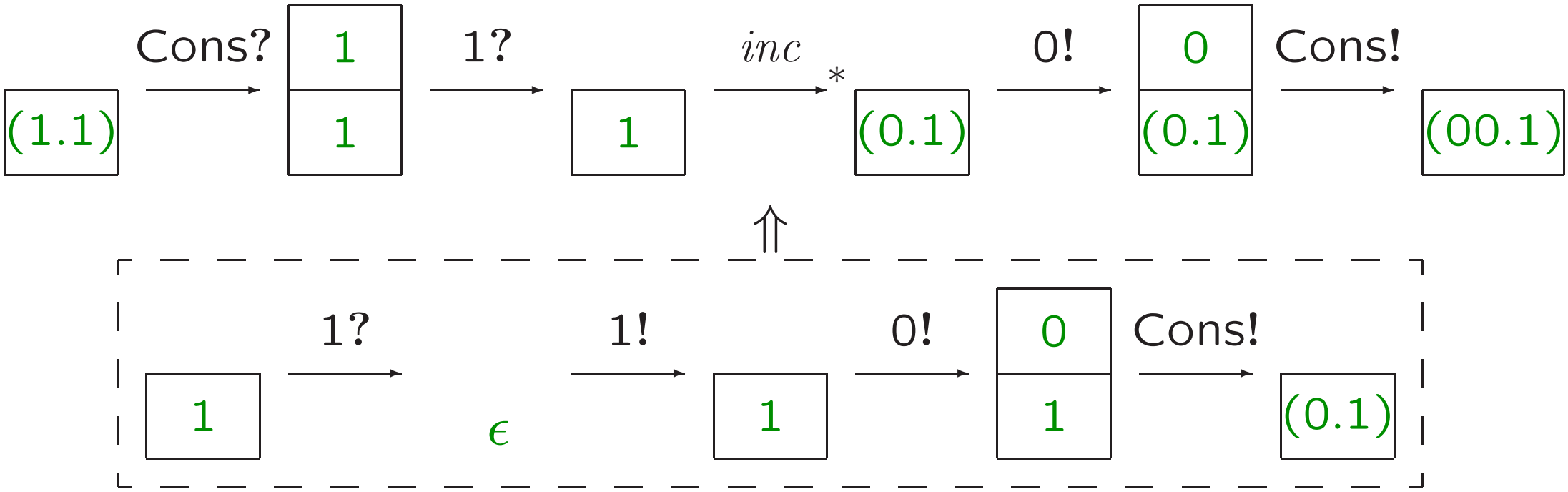
- Translation to Grammar Language:

  $inc \longrightarrow$ 1? 1! 0! Cons!

  $inc \longrightarrow$ Cons? 0? 1! Cons!

  $inc \longrightarrow$ Cons? 1? inc 0! Cons!

- Stack-based Evaluation:



Semantics of the Grammar Language

Transition rules (Ops $\times$ Stack $\longrightarrow$ Ops $\times$ Stack):

$$\langle a{:}ts, vs\rangle \;\Rightarrow\; \langle ts, \mathcal{A}[\![\, a \,]\!]\, vs\rangle$$

$$\langle f{:}ts, vs\rangle \;\Rightarrow\; \langle ts' +\!\!+ ts, vs\rangle \quad \text{if } f \longrightarrow ts' \in p$$

Nondeterministic choice!

Atomic operations (Op $\longrightarrow$ Stack $\longrightarrow$ Stack):

$$\mathcal{A}[\![\, c? \,]\!]\, c(v_1, \ldots, v_n){:}vs = v_1{:}\ldots{:}v_n{:}vs$$

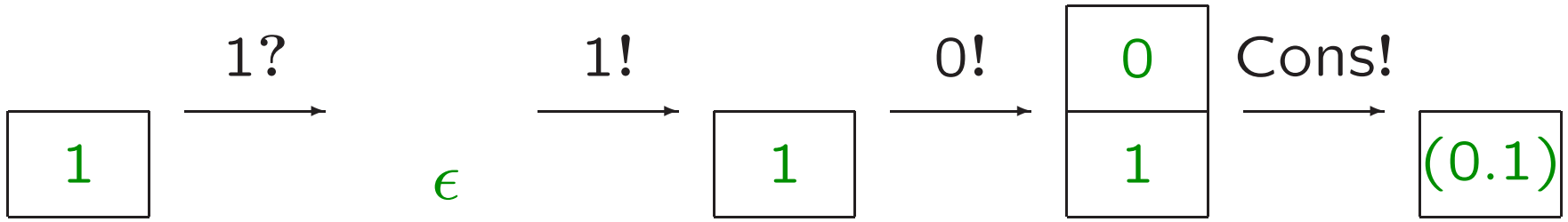
$$\mathcal{A}[\![\, c! \,]\!]\, v_1{:}\ldots{:}v_n{:}vs = c(v_1, \ldots, v_n){:}vs \quad \text{if } n = |c|$$

$$\mathcal{A}[\![\, \lfloor\_\rfloor \,]\!]\, v{:}vs = \lfloor v \rfloor{:}vs$$

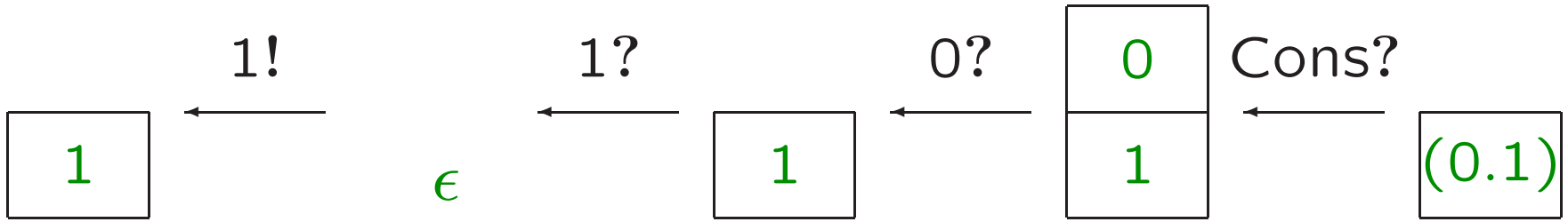
$$\mathcal{A}[\![\, \pi \,]\!]\, vs = \pi \circ vs$$

Comparison: Evaluation of *inc* and *dec*

- Source program:



- Inverse program:



Local Inversion

- Operation: Inverted individually.

$$
\begin{aligned}
\mathit{inv}[\![\ c!\ ]\!] &= c? \\
\mathit{inv}[\![\ c?\ ]\!] &= c! \\
\mathit{inv}[\![\ f\ ]\!] &= f^{-1} \\
\mathit{inv}[\![\ \lfloor \_ \rfloor\ ]\!] &= \lfloor \_ \rfloor \\
\mathit{inv}[\![\ (i_1, \ldots, i_n)\ ]\!] &= (i_1, \ldots, i_n)^{-1}
\end{aligned}
$$

- Sequences: Backwards reading.

$$\mathit{Inv}[\![\ f \rightarrow t_1 \ldots t_n\ ]\!] = f^{-1} \rightarrow \mathit{inv}[\![\ t_n\ ]\!] \ldots \mathit{inv}[\![\ t_1\ ]\!]$$

- Program: Global inversion by local inversion.

$$\mathit{INV}[\![\ pgm\ ]\!] = \{\ \mathit{Inv}[\![\ def\ ]\!] \mid def \in pgm\ \}$$

Local Inversion of *inc*

- Source program:

  $inc \longrightarrow$ 1? 1! 0! Cons!

  $inc \longrightarrow$ Cons? 0? 1! Cons!

  $inc \longrightarrow$ Cons? 1? inc 0! Cons!

- Inverse program:

  $dec \longrightarrow$ Cons? 0? 1? 1!

  $dec \longrightarrow$ Cons? 1? 0! Cons!

  $dec \longrightarrow$ Cons? 0? dec 1! Cons!

Correctness of Local Inversion

- Inverse programs produced by local inversion of injective programs:

$$[\![p]\!]\, vs_{\mathsf{in}} = vs_{\mathsf{out}} \iff [\![p^{-1}]\!]\, vs_{\mathsf{out}} = vs_{\mathsf{in}}$$

- Generally:
  There is a computation of p with vs_{in} that terminates and yields output vs_{out} iff there is a computation of p^{-1} with vs_{out} that terminates and yields output vs_{in}.

Transformation by Left-Factoring (1/2)

- Nondeterministic program:

$dec \rightarrow$ Cons? 1? 0! Cons!

$dec \rightarrow$ Cons? 0? 1? 1!

$dec \rightarrow$ Cons? 0? dec 1! Cons!

- 1. Left-factoring (introduce new f_1):

$dec \rightarrow$ Cons? f_1

$f_1 \rightarrow$ 1? 0! Cons!

$f_1 \rightarrow$ 0? 1? 1!

$f_1 \rightarrow$ 0? dec 1! Cons!

Transformation by Left-Factoring (2/2)

- 2. Left-factoring (introduce new f_2):

$dec \to \text{Cons?}\ f_1$ $\quad f_1 \to 1?\ 0!\ \text{Cons!}$

$f_1 \to \boxed{0?}\ f_2$ $\quad f_2 \to 1?\ 1!$

$f_2 \to dec\ 1!\ \text{Cons!}$

- 3. Unfold function call dec:

$dec \to \text{Cons?}\ f_1$ $\quad f_1 \to 1?\ 0!\ \text{Cons!}$

$f_1 \to 0?\ f_2$ $\quad f_2 \to 1?\ 1!$

$f_2 \to \text{Cons?}\ f_1\ 1!\ \text{Cons!}$

Inverse Program *dec*

- Translation to the source language:

$$dec(x_0) \triangleq \textbf{case}\ x_0\ \textbf{of}\ \ x_1{:}x_2 \rightarrow f_1(x_1, x_2)$$

$$f_1(x_0, x_1) \triangleq\ \textbf{case}\ x_0\ \textbf{of}$$

$$0 \rightarrow \textbf{case}\ x_1\ \textbf{of}\ \ 1 \rightarrow 1$$

$$x_2{:}x_3 \rightarrow 1{:}f_1(x_2, x_3)$$

$$1 \rightarrow 0{:}x_1$$

- dec is a deterministic inverse program of inc.
  - *e.g.*, $inc(11) = 100$ and $dec(100) = 11$.

Example: *reverse* (tail-recursive)

- In grammar language:

  $reverse \longrightarrow$ Nil! swap rev

  $rev \longrightarrow$ Nil?

  $rev \longrightarrow$ Cons? swapd Cons! swap rev

- Local inversion:

  ${reverse}^{-1} \longrightarrow {rev}^{-1}$ swap Nil?

  ${rev}^{-1} \longrightarrow$ Nil!

  ${rev}^{-1} \longrightarrow {rev}^{-1}$ swap Cons? swapd Cons!

Analogy with Context-Free Grammars

- View programs as context-free grammar:
  - Constructors $c!$ and pattern matchings $c?$ correspond to terminal symbols.
  - Functions correspond to nonterminal symbols.

- Program:

  $dec \longrightarrow$ Cons? 0? 1? 1!

  $dec \longrightarrow$ Cons? 1? 0! Cons!

  $dec \longrightarrow$ Cons? 0? dec 1! Cons!

  Grammar:

  $D \longrightarrow a\,c\,e\,f$

  $D \longrightarrow a\,e\,d\,b$

  $D \longrightarrow a\,c\,D\,f\,b$

- LR(0) parsing can be applied to the grammar to obtain a deterministic parser.

Good Advice for Automatic Inversion

- Language for Inversion
  - Each operation t has one operation t^{-1} as its inverse.
  - Each operation t^{-1} is part of the language.
  - Multiple inputs give multiple outputs (co-arity).
  - Value domain: postcond. become tests in program.
- Structure of Inverter
  - local inversion + eliminate nondet.
- As in PE, some programs invert well, others don't.

Thanks for Generous Support

- Japan Science and Technology Agency (JST)
- Waseda University, Tokyo (Y. Futamura)

Bibliography

- S.M. Abramov and R. Glück. Principles of Inverse Computation and the Universal Resolving Algorithm. In The Essence of Computation: Complexity, Analysis, Transformation (T.Mogensen, D.Schmidt, I.H.Sudborough, eds.), LNCS 2566: 269–295. Springer-Verlag. 2002.

- R. Glück and M. Kawabe. A Program Inverter for a Functional Language with Equality and Constructors. In Programming Languages and Systems. Proceedings (A.Ohori, ed.), LNCS 2895: 246–264. Springer-Verlag. 2003.

- R. Glück and M. Kawabe. Derivation of deterministic inverse programs based on LR parsing. In Functional and Logic Programming. Proceedings (Y.Kameyama, P.J.Stuckey, eds.). LNCS 2998: 291–306. Springer-Verlag. 2004.

- M. Kawabe and R. Glück . The Program Inverter LRinv and its Structure. In Practical Aspects of Declarative Languages. Proceedings (M.Hermenegildo, D.Cabeza, eds.). LNCS 3350: 219–234. Springer-Verlag. 2005.