PAT - Assignment 4

Mikkel Willén
bmq419

May 29, 2024

Task 1

The paper shows examples of full-inversion, partial-inversion and semi-inversion. Below is four examples of inversions.

Multiplication to division:
This is a partial inversion with index sets: $I = \{2\}$ and $O = \{1\}$. The multiplication function $\texttt{mul}(x, y)$ with inputs x and y and output z is partially inverted to the division function $\texttt{div}(z, y)$.
Rules:
$\texttt{mul}(0, y) \rightarrow 0$
$\texttt{mul}(s(x), y) \rightarrow \texttt{add}(y, \texttt{mul}(x, y))$
Inverted rules:
$\texttt{div}(0, y) \rightarrow 0$
$\texttt{div}(z, y) \rightarrow s(x) \leftarrow \texttt{sub}(z, y) \wedge \texttt{div}(w, y)$

Symmetric encrypter to decrypter This is a partial inversion with index sets: $I = \{2\}$ and $O = \{1\}$. The symmetric encrypter function is partially inverted to produce a decrypter function.
Rules:
$\texttt{encrypt}(\texttt{nil}, \texttt{key}) \rightarrow \texttt{nil}$
$\texttt{encrypt}(x : xs, \texttt{key}) \rightarrow z : zs \leftarrow \texttt{mod4}(\texttt{key}) \wedge \texttt{add}(x, y) \wedge \texttt{encrypt}(xs, \texttt{key})$
Inverted rules:
$\texttt{encrypt}^{\{2\}\{1\}}(\texttt{key}, \texttt{nil}) \rightarrow \texttt{nil}$
$\texttt{encrypt}^{\{2\}\{1\}}(\texttt{key}, z : zs) \rightarrow x : xs \leftarrow \texttt{mod4}^{\{1\}\emptyset}(\texttt{key}) \wedge \texttt{add}^{\{2\}\{1\}}(y, z) \wedge \texttt{encrypt}^{\{2\}\{1\}}(\texttt{key}, zs)$

Discrete simulation of free fall:
This is a full inversion with index sets: $I = \{0\}$ and $O = \{0\}$. The function for simulating free fall is fully inverted to determine initial conditions from final states.
Rules:
$\texttt{fall}_0(v, h, 0) \rightarrow \langle v, h \rangle$
$\texttt{fall}_0(v_0, h_0, s(t)) \rightarrow \langle v, h \rangle \leftarrow \texttt{add}(v_0, s_5(0)) \wedge \texttt{height}(h_0, v_n) \wedge \texttt{fall}_0(v_n, h_n, t)$
Inverted rules:
$\texttt{fall}_1(v, h) \rightarrow \langle v_0, h_0, s(t) \rangle \leftarrow \texttt{fall}_1(v_n, h_n) \wedge \texttt{add}^{\{2\}\{1\}}(s_{10}(0), v_n) \wedge \texttt{height}^{\{2\}\{1\}}(v_n, h_n)$

Janus language inverter:
This is a full inversion with index sets: $I = \{\}$ and $O = \{1\}$. The Janus language inverter is fully inverted to produce the same procedure.
Rules:
$\texttt{inv(proc}(u, v)) \rightarrow \langle \texttt{proc(name, progr)} \rangle \leftarrow \texttt{invName}(u) \wedge \texttt{inv}(v)$
$\texttt{inv}(+= x, y) \rightarrow \langle -= x, y \rangle$

Inverted rules:
$\texttt{inv}(\texttt{proc}(name, progr)) \rightarrow \langle \texttt{proc}(u, v) \rangle \leftarrow \texttt{invName}(name) \wedge \texttt{inv(progr)}$
$\texttt{inv}(+ = x, y) \rightarrow \langle - = x, y \rangle$

Task 2

Here is the CCS definition for unary subtraction:
$\texttt{sub}(z, 0) \rightarrow z$
$\texttt{sub}(s(z), s(x)) \rightarrow \texttt{sub}(z, x)$
Subtracting zero from any unary number results inthe number itself, hence the first case. Subtracting the successor of a number from the successor of another number reduces the problem to subtracting the numbers themselves, hence the second case.

To check that the system rewrites $\texttt{sub}(s(s(0)), s(0))$ into $s(0)$, we will trace the rewrite steps.
First we have

$$\texttt{sub}(s(s(0)), s(0))$$

Then we apply the recursive rule to get

$$\texttt{sub}(s(s(0)), s(0)) \rightarrow \texttt{sub}(s(0), 0)$$

Then we apply the base case to get

$$\texttt{sub}(s(0), 0) \rightarrow s(0)$$

and thus we have that $\texttt{sub}(s(s(0)), s(0))$ correctly rewrites to $s(0)$.

Task 3

We have chosen to perform semi-inversion on fall with $I = \{1\}$ and $O = \{2\}$. Below is the output of the program.

Inverting rules

```
Inverting rule "fall" with I = {1} and O = {2}

-------------------------------
--   TRIVIAL RULE INVERTER   --
-------------------------------
(CONDITIONTYPE ORIENTED)
(VAR v h t v0 h0 vn hn ht x y z)
(SIG
  (fall 3 2) (add 2 1) (height 2 1) (sub 2 1))
(RULES
  fall{1}{2}(v, h) -> tp(h, 0, v)
  fall{1}{2}(v0, h) -> tp(h0, s(t), v)
            | add{1,2}{}(v0, s(s(s(s(s(s(s(s(s(s(0))))))))))) == tp(vn)
            , height{1,2}{}(h0, vn) == tp(hn)
            , fall{1,2,3}{}(vn, hn, t) == tp(v, h)
```

```
  add{1,2}{}(0, y) -> tp(y)
  add{1,2}{}(s(x), y) -> tp(s(z)) | add{1,2}{}(x, y) == tp(z)
  height{1,2}{}(h0, vn) -> tp(hn) | add{1,2}{}(h0, s(s(s(s(s(0))))))) == tp(ht)
            , sub{1,2}{}(vn, ht) == tp(hn)
  fall{1,2,3}{}(v, h, 0) -> tp(v, h)
  fall{1,2,3}{}(v0, h0, s(t)) -> tp(v, h)
            | add{1,2}{}(v0, s(s(s(s(s(s(s(s(s(s(0)))))))))) == tp(vn)
            , height{1,2}{}(h0, vn) == tp(hn)
            , fall{1,2,3}{}(vn, hn, t) == tp(v, h)
  sub{1,2}{}(0, y) -> tp(y)
  sub{1,2}{}(s(x), s(z)) -> tp(y) | sub{1,2}{}(x, z) == tp(y)
)
(COMMENT )
------------------------------
--  PURE FULL RULE INVERTER  --
------------------------------
(CONDITIONTYPE ORIENTED)
(VAR v h t v0 h0 vn hn ht x y z)
(SIG
  (fall 3 2) (height 2 1) (add 2 1) (sub 2 1))
(RULES
  fall{1}{2}(v, h) -> tp(h, 0, v)
  fall{1}{2}(v0, h) -> tp(h0, s(t), v) | fall{}{1,2}(v, h) == tp(vn, hn, t)
            , height{}{1}(hn) == tp(h0, vn)
            , add{}{1}(vn) == tp(v0, s(s(s(s(s(s(s(s(s(0))))))))))
  fall{}{1,2}(v, h) -> tp(v, h, 0)
  fall{}{1,2}(v, h) -> tp(v0, h0, s(t)) | fall{}{1,2}(v, h) == tp(vn, hn, t)
            , height{}{1}(hn) == tp(h0, vn)
            , add{}{1}(vn) == tp(v0, s(s(s(s(s(s(s(s(s(0))))))))))
  height{}{1}(hn) -> tp(h0, vn) | sub{}{1}(hn) == tp(vn, ht)
            , add{}{1}(ht) == tp(h0, s(s(s(s(0)))))
  add{}{1}(y) -> tp(0, y)
  add{}{1}(s(z)) -> tp(s(x), y) | add{}{1}(z) == tp(x, y)
  sub{}{1}(y) -> tp(0, y)
  sub{}{1}(y) -> tp(s(x), s(z)) | sub{}{1}(y) == tp(x, z)
)
(COMMENT )
------------------------------
--   PARTIAL RULE INVERTER   --
------------------------------
(CONDITIONTYPE ORIENTED)
(VAR v h t v0 h0 vn hn ht x y z)
(SIG
  (fall 3 2) (height 2 1) (add 2 1) (sub 2 1))
(RULES
  fall{1}{2}(v, h) -> tp(h, 0, v)
  fall{1}{2}(v0, h) -> tp(h0, s(t), v) | fall{}{1,2}(v, h) == tp(vn, hn, t)
            , height{2}{1}(vn, hn) == tp(h0)
```

```
            , add{1,2}{1}(v0, s(s(s(s(s(s(s(s(s(0)))))))))), vn) == tp( )
  fall{}{1,2}(v, h) -> tp(v, h, 0)
  fall{}{1,2}(v, h) -> tp(v0, h0, s(t)) | fall{}{1,2}(v, h) == tp(vn, hn, t)
            , height{2}{1}(vn, hn) == tp(h0)
            , add{2}{1}(s(s(s(s(s(s(s(s(s(0)))))))))), vn) == tp(v0)
  height{2}{1}(vn, hn) -> tp(h0) | sub{1}{1}(vn, hn) == tp(ht)
            , add{2}{1}(s(s(s(s(s(0))))), ht) == tp(h0)
  add{1,2}{1}(0, y, y) -> tp( )
  add{1,2}{1}(s(x), y, s(z)) -> tp( ) | add{1,2}{1}(x, y, z) == tp( )
  add{2}{1}(y, y) -> tp(0)
  add{2}{1}(y, s(z)) -> tp(s(x)) | add{2}{1}(y, z) == tp(x)
  sub{1}{1}(0, y) -> tp(y)
  sub{1}{1}(s(x), y) -> tp(s(z)) | sub{1}{1}(x, y) == tp(z)
)
(COMMENT )

------------------------------
--     SEMI RULE INVERTER    --
------------------------------
(CONDITIONTYPE ORIENTED)
(VAR v h t v0 h0 vn hn ht x y z)
(SIG
  (fall 3 2) (add 2 1) (height 2 1) (sub 2 1))
(RULES
  fall{1}{2}(v, h) -> tp(h, 0, v)
  fall{1}{2}(v0, h) -> tp(h0, s(t), v) |
            add{1,2}{}(v0, s(s(s(s(s(s(s(s(s(0)))))))))) == tp(vn)
            , fall{1}{2}(vn, h) == tp(hn, t, v)
            , height{2}{1}(vn, hn) == tp(h0)
  add{1,2}{}(0, y) -> tp(y)
  add{1,2}{}(s(x), y) -> tp(s(z)) | add{1,2}{}(x, y) == tp(z)
  height{2}{1}(vn, hn) -> tp(h0) | sub{1}{1}(vn, hn) == tp(ht)
            , add{2}{1}(s(s(s(s(s(0))))), ht) == tp(h0)
  sub{1}{1}(0, y) -> tp(y)
  sub{1}{1}(s(x), y) -> tp(s(z)) | sub{1}{1}(x, y) == tp(z)
  add{2}{1}(y, y) -> tp(0)
  add{2}{1}(y, s(z)) -> tp(s(x)) | add{2}{1}(y, z) == tp(x)
)
(COMMENT )
```

Comparisons

```
-------------------------------------
-       TRIVIAL RULE INVERTER       -
-------------------------------------
- Comparing fall and its inversion
-------------------------------------
                                      ORIG    TRIV
```

```
Number of functions                  4       5
Number of rules                      7       9

Functional                           [x]     [ ]
- Orthogonal                         [x]     [ ]
  . Non-overlapping                  [x]     [ ]
  . Left-linear                      [x]     [x]
- EV-Free                            [x]     [x]
- Left-to-right determ.              [x]     [ ]

Reversible                           [ ]     [ ]
- Functional                         [x]     [ ]
- Output-orthogonal                  [ ]     [ ]
  . Non-output-overlapping           [ ]     [ ]
  . Right-linear                     [x]     [x]
- Strictly non-erasing               [x]     [ ]
  . Non-erasing                      [x]     [x]
  . Weakly non-erasing               [x]     [ ]

-------------------------------------
-      PURE FULL RULE INVERTER     -
-------------------------------------
- Comparing fall and its inversion
-------------------------------------
                                     ORIG    FULL
Number of functions                  4       5
Number of rules                      7       9

Functional                           [x]     [ ]
- Orthogonal                         [x]     [ ]
  . Non-overlapping                  [x]     [ ]
  . Left-linear                      [x]     [x]
- EV-Free                            [x]     [x]
- Left-to-right determ.              [x]     [ ]

Reversible                           [ ]     [ ]
- Functional                         [x]     [ ]
- Output-orthogonal                  [ ]     [x]
  . Non-output-overlapping           [ ]     [x]
  . Right-linear                     [x]     [x]
- Strictly non-erasing               [x]     [ ]
  . Non-erasing                      [x]     [x]
  . Weakly non-erasing               [x]     [ ]

-------------------------------------
-      PARTIAL RULE INVERTER       -
-------------------------------------
- Comparing fall and its inversion
```

```
-------------------------------------
                                      ORIG    PART

-------------------------------------
-         SEMI RULE INVERTER        -
-------------------------------------
- Comparing fall and its inversion
-------------------------------------
                                      ORIG    SEMI
Number of functions                   4       5
Number of rules                       7       9

Functional                            [x]     [ ]
- Orthogonal                          [x]     [ ]
  . Non-overlapping                   [x]     [ ]
  . Left-linear                       [x]     [ ]
- EV-Free                             [x]     [x]
- Left-to-right determ.               [x]     [x]

Reversible                            [ ]     [ ]
- Functional                          [x]     [ ]
- Output-orthogonal                   [ ]     [ ]
  . Non-output-overlapping            [ ]     [ ]
  . Right-linear                      [x]     [x]
- Strictly non-erasing                [x]     [ ]
  . Non-erasing                       [x]     [ ]
  . Weakly non-erasing                [x]     [x]
```

The trivial rule inversion function results in overlapping rules and loss of orthogonality, which undermines the functional and reversible properties of the system. The pure full rule inversion function introduces overlapping rules and does not preserve strict non-erasing properties, resulting in a non-functional and non-orthogonal system. Partial rule inversion does not fully address all aspects of the rules, resulting in potential gaps in functionality and orthogonality. The semi rule inverter maintains left-to-right determinism but fails to preserve orthogonality and strict non-erasing properties, leading to a non-functional and non-reversible system.

Task 4

List reversal semi inversion:
The original program reverses the elements of a list. The semi-inversion of the program, reconstructs the orignial list given the head of the orignial list and the reversed list.

Original program:
$\texttt{reverse}([], [])$
$\texttt{reverse}([H|T], R) \leftarrow \texttt{reverse}(T, \texttt{rev}_t) \wedge \texttt{append}(\texttt{rev}_t, [H], R)$

Semi-Inversion:
$\texttt{reverse\_si}(H, \texttt{rev}_t, r) \rightarrow L$
$\texttt{reverse\_si}(H, [], [H]$

$\texttt{reverse\_si}(H, [H|\texttt{rev}_t], R) \rightarrow [H|T] \leftarrow \texttt{reverse}(T, \texttt{rev}_t)$

The semi-inversion `reverse_si` reconstructs the original list L given its head H, the reversed tail $\texttt{rev}_t$ and the fully reversed list R. This can be useful in a scenario where only partial knowledge of the input and output is available.