Towards Reverse Execution of Constraint Handling Rules

Amira Zaki*, Thom Fruehwirth* and Slim Abdennadher**

August 27, 2013 | Ulm University*, German University in Cairo**
Rubik’s Cube

Inverse Execution

- **Inverse/Reverse/Backwards**: non-deterministic process of discovering inputs to a program starting from an output
- It is a form of abduction, which attempts to infer the initial query from an observed result
- **Applications**: encryption/decryption, compression/decompression, inverting arithmetic functions, insert/delete operations on data structures, roll-back transactions
Reversing Color Mixing

Forward Constraint Handling Rules

- green, red $\leftrightarrow$ brown.
- orange, blue $\leftrightarrow$ brown.
Reversing Color Mixing

Forward Constraint Handling Rules

green, red $\leftrightarrow$ brown.

orange, blue $\leftrightarrow$ brown.
Reversing Color Mixing

Forward Constraint Handling Rules

green, red <=> brown.
orange, blue <=> brown.
Reversing Color Mixing

Forward Constraint Handling Rules

green, red $\leftrightarrow$ brown.

orange, blue $\leftrightarrow$ brown.
# Constraint Handling Rules

## Simplification rule

\[ r @ H_r \iff G | B_b, B_c \]

## Propagation rule

\[ r @ H_k \Rightarrow G | B_b, B_c \]
## Constraint Handling Rules

**Simplification rule**

\[ r \ @ \ H_r \Leftrightarrow G \mid B_b, \ B_c \]

**Propagation rule**

\[ r \ @ \ H_k \Rightarrow G \mid B_b, \ B_c \]

**Generalized (combined) simpagation rule**

\[ r \ @ \ H_k \ \backslash \ \ H_r \Leftrightarrow G \mid B_b, \ B_c \]
Simple Inversion

Generalized simpagation rule

\[ r @ H_k \setminus H_r \Leftrightarrow G \mid B_b, B_c \]
Simple Inversion

Generalized simpagation rule

\[ r \circ \left( H_k \setminus H_r \right) \Leftrightarrow G \mid B_b, B_c \]

Simple inversion means adding removed constraints and removing added constraints:

Inverse simpagation rule

\[ \text{inv-}r \circ \left( H_k \setminus B_c \right) \Leftrightarrow B_b, G \mid H_r \]
Simple Inversion - Example I

List reverse - forward

- **rule1** @ reverse(X) <-> reverse(X, []).  
- **rule2** @ reverse([X|Xs], Ys) <-> reverse(Xs, [X|Ys]).  
- **rule3** @ reverse([], X) <-> out(X).

reverse([1,2,3]) forward out([3,2,1])
Simple Inversion - Example I

List reverse - forward

rule1  @ reverse(X)  <=>  reverse(X,[]).
rule2  @ reverse([X|Xs],Ys)  <=>  reverse(Xs,[X|Ys]).
rule3  @ reverse([],X)  <=>  out(X).

List reverse - inverse

inv-rule1  @ reverse(X,[])  <=>  reverse(X).
inv-rule2  @ reverse(Xs,[X|Ys])  <=>  reverse([X|Xs],Ys).
inv-rule3  @ out(X)  <=>  reverse([],X).
Simple Inversion - Example I

List reverse - forward

rule1 @ reverse(X) <=> reverse(X, []).  
rule2 @ reverse([X|Xs], Ys) <=> reverse(Xs, [X|Ys]).  
rule3 @ reverse([], X) <=> out(X).

List reverse - inverse

inv-rule1 @ reverse(X, []) <=> reverse(X).  
inv-rule2 @ reverse(Xs, [X|Ys]) <=> reverse([X|Xs], Ys).  
inv-rule3 @ out(X) <=> reverse([], X).
Simple Inversion - Example II

Exchange sort - forward

\[ \text{esort} @ a(I, V), a(J, W) \iff I > J, V < W \mid a(I, W), a(J, V). \]
Simple Inversion - Example II

Exchange sort - forward

esort @ a(I,V), a(J,W) \iff I>J, V<W \mid a(I,W), a(J,V).

Exchange sort - inverse

inv-esort @ a(I,W), a(J,V) \iff I>J, V<W \mid a(I,V), a(J,W).

Problem: input uncovered by the inverse program is a possible solution but not the exact one used in the forward run
Inversion of Constraint Handling Rules

- Compute from a final state to one or all predecessor states
- Process is non-deterministic in two different dimensions:
  1. Different initial states could be uncovered from a final state, e.g. inverse of a list-append operation
  2. Since any state can be an initial CHR state, then any intermediate state of a computation is also a possible initial state
Inversion of Constraint Handling Rules

- Compute from a final state to one or all predecessor states
- Process is non-deterministic in two different dimensions:
  1. Different initial states could be uncovered from a final state, e.g. inverse of a list-append operation
  2. Since any state can be an initial CHR state, then any intermediate state of a computation is also a possible initial state

Proposed Exhaustive Execution Approach
Achieve incremental and exhaustive backwards rule application, while exploring different program paths; using CHR with disjunction whilst adopting a modified conflict resolution strategy
Exhaustive Inversion - the transformation

For every generalized simpagation rule, add rule pair:

\[
\begin{align*}
\text{delay-ignore-r} & @ H_k, B_c \Rightarrow B_b, G \mid \text{ruleset([rule}(H_k, B_c))]; \text{true} \\
\text{apply-r} & @ H_k \setminus B_c, \text{apply(rule}(H_k, B_c)) \Leftrightarrow B_b, G \mid H_r
\end{align*}
\]
Exhaustive Inversion - the transformation

For every generalized simpagation rule, add rule pair:

\[
\text{delay-ignore-r @ } H_k, B_c \Rightarrow B_b, G \mid \text{ruleset([rule}(H_k,B_c)]); \text{ true}
\]
\[
\text{apply-r @ } H_k \setminus B_c, \text{ apply(rule}(H_k,B_c)) \Leftrightarrow B_b, G \mid H_r
\]

For every inverse program, add rules:

\[
\text{collect @ ruleset}(L1), \text{ ruleset}(L2)
\]
\[
\Rightarrow \text{append}(L1,L2,L3), \text{ ruleset}(L3).
\]
\[
\text{halt-fire @ ruleset}(L) \setminus \text{fire} \Leftrightarrow L\=[] \mid \text{true; fireruleset.}
\]
\[
\text{halt-choose @ fireruleset, ruleset}(L)
\]
\[
\Rightarrow \text{choose}(L,R,L1), \text{ ruleset}(L1), \text{ apply}(R),
\]
\[
(\text{true; fireruleset}).
\]
Exhaustive Inversion - the transformation

For every generalized simpagation rule, add rule pair:

\[
\begin{align*}
\text{delay-ignore-r @ } & H_k, B_c \Rightarrow B_b, G \mid \text{ruleset([rule}(H_k, B_c))] \}; \text{true} \\
\text{apply-r @ } & H_k \setminus B_c, \text{apply(rule}(H_k, B_c)) \Leftrightarrow B_b, G \mid H_r
\end{align*}
\]

For every inverse program, add rules:

\[
\begin{align*}
\text{collect @ ruleset}(L_1), \text{ruleset}(L_2) \\
\mathrel{\Leftrightarrow} \text{append}(L_1, L_2, L_3), \text{ruleset}(L_3). \\
\text{halt-fire @ ruleset}(L) \setminus \text{fire} \Leftrightarrow L = \square \mid \text{true}; \text{fireruleset.} \\
\text{halt-choose @ fireruleset, ruleset}(L) \\
\mathrel{\Leftrightarrow} \text{choose}(L, R, L_1), \text{ruleset}(L_1), \text{apply}(R), \text{true}; \text{fireruleset).} \\
\text{bfs @ strategy}(S) \setminus \text{choose}(L, R, L_1) \\
\mathrel{\Leftrightarrow} S = \text{bfs} \mid L = [R | L_1]. \\
\text{rand @ strategy}(S) \setminus \text{choose}(L, R, L_1) \\
\mathrel{\Leftrightarrow} S = \text{rand} \mid \text{random_select}(R, L, L_1).
\end{align*}
\]
Exhaustive Inversion - Example I

Exchange Sort Revisited

delay-ignore-esort @ a(I,W), a(J,V) ==> I>J, V<W

| ruleset([[rule([], [a(I,W), a(J,V)])]]; true.

apply-esort @ a(I,W), a(J,V), apply(rule([], [a(I,W), a(J,V)]))

<= I>J, V<W | a(I,V), a(J,W).
Exhaustive Inversion - Example I

Exchange Sort Revisited

delay-ignore-esort @ a(I,W), a(J,V) ==> I>J, V<W
| ruleset ([rule([], [a(I,W), a(J,V)])]; true.
apply-esort @ a(I,W), a(J,V), apply(rule([], [a(I,W), a(J,V)]))
<= I>J, V<W | a(I,V), a(J,W).

![Exchange Sort Diagram](image-url)
Exhaustive Inversion - Example II

**GCD - forward**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base @ gcd(0) &lt;=&gt; true.</td>
<td></td>
</tr>
<tr>
<td>gcd @ gcd(N) \ gcd(M) &lt;=&gt; 0&lt;N, N=&lt;M</td>
<td>L eq M-N, gcd(L).</td>
</tr>
</tbody>
</table>

Sample forward run:

?- gcd(12),gcd(8),gcd(4).

gcd(4).
Exhaustive Inversion - Example II

**GCD - forward**

- base @ gcd(0) <=> true.
- gcd @ gcd(N) \ gcd(M) <=> 0<N, N=<M | L eq M-N, gcd(L).

**GCD - inverse**

- delay-ignore-base @ top ==> ruleset([rule([], [top])]); true.
- apply-base @ top, apply(rule([], [top])) <=> gcd(0), top.
- delay-ignore-gcd @ gcd(N), gcd(L)
  ==> L eq M-N, 0<N, N=<M
  | ruleset([rule([gcd(N)], [gcd(L)])); true.
- apply-gcd @ gcd(N) \ gcd(L), apply(rule([gcd(N)], [gcd(L)]))
  <=> L eq M-N, 0<N, N=<M | gcd(M).
Exhaustive Inversion - Example II with breadth first search

?- strategy(bfs),gcd(4),top,fire.
Exhaustive Inversion - Example II with random search

?- strategy(rand),gcd(4),top,fire.
Closing Remarks

- **Online CHR inversion tool:**
  
  http://chr.informatik.uni-ulm.de/inverter/

- **Summary:**
  
  - **Simple inversion** executes in depth-first; produces correct yet incomplete result; effective for one-to-one functions and one-ruler programs
  
  - **Exhaustive inversion** ensures that all possible inversion paths are explored incrementally; uncovers all possible inputs; extendable for other execution strategies

- **Future work:**
  
  - Test with large scale inversion applications
  
  - Optimize exhaustive execution transformation
  
  - Explore other execution strategies