Reversible Computations in Logic Programming

Germán Vidal

VRAIN – Universitat Politècnica de València

12th Conference on Reversible Computation
9-10, July, 2020
Oslo, Norway

Logic programming: invertibility vs reversibility

Example (addition on natural numbers $\mathtt{0/s(\_)}$)

$\mathtt{add(0,Y,Y).}$	*(fact)*
$\mathtt{add(s(X),Y,s(Z))} \leftarrow \mathtt{add(X,Y,Z).}$	*(rule)*

Invertibility

$$\mathtt{add(s(s(0)),s(0),A)} \rightarrow^*_\theta \Box \ \text{ with } \theta = \{\mathtt{A/s(s(s(0)))}\}$$

but also

$$\mathtt{add(B,s(0),s(s(s(0))))} \rightarrow^*_\sigma \Box \ \text{ with } \sigma = \{\mathtt{B/s(s(0))}\}$$

Reversibility

$$\begin{tikzcd}
\mathtt{add(s(s(0)),s(0),A)} \arrow[r] & \mathtt{add(s(0),s(0),A')} \arrow[r] \arrow[l, bend right] & \mathtt{add(0,s(0),A'')} \arrow[r] \arrow[l, bend right] & \Box \arrow[l, bend right]
\end{tikzcd}$$

Logic programming: invertibility vs reversibility

Example (addition on natural numbers $\mathtt{0}/\mathtt{s(\_)}$)

$$\begin{array}{ll} \mathtt{add(0,Y,Y).} & \textit{(fact)} \\ \mathtt{add(s(X),Y,s(Z))} \leftarrow \mathtt{add(X,Y,Z).} & \textit{(rule)} \end{array}$$

Invertibility

$$\mathtt{add(s(s(0)),s(0),A)} \to_{\theta}^{*} \Box \quad \text{with } \theta = \{\mathtt{A}/\mathtt{s(s(s(0)))}\}$$

but also

$$\mathtt{add(B,s(0),s(s(s(0))))} \to_{\sigma}^{*} \Box \quad \text{with } \sigma = \{\mathtt{B}/\mathtt{s(s(0))}\}$$

Reversibility

$$\begin{tikzcd} \mathtt{add(s(s(0)),s(0),A)} \arrow[r] & \mathtt{add(s(0),s(0),A')} \arrow[r] \arrow[l, bend right] & \mathtt{add(0,s(0),A'')} \arrow[r] \arrow[l, bend right] & \Box \arrow[l, bend right] \end{tikzcd}$$

Goal of this paper

Reversibilization

- Define an appropriate Landauer embedding so that logic programming derivations become reversible

Main applications: program understanding & debugging

Goal of this paper

Reversibilization

- Define an appropriate Landauer embedding so that logic programming derivations become reversible

Main applications: program understanding & debugging

Logic programming: a very brief introduction

A program is given by a set of clauses:

- Facts, e.g., $\mathtt{add(0, Y, Y)}$.
- Rules, e.g., $\mathtt{add(s(X), Y, s(Z))} \leftarrow \mathtt{add(X, Y, Z)}$.

The operational semantics is called SLD-resolution:

$$\overbrace{\underline{A_1}, \ldots, A_n}^{goal} \to_\sigma (B_1, \ldots, B_m, A_2, \ldots, A_n)\sigma$$

if
- there is a program clause $H \leftarrow B_1, \ldots, B_m$
- σ is the mgu of A_1 and H

  (substitution σ is a unifier of A and B if $A\sigma = B\sigma$)

A derivation is successful if it ends with the empty goal ($\Box$)
The computed answer is then the composition of the mgu's

Logic programming: a very brief introduction

A program is given by a set of clauses:

- Facts, e.g., $\mathtt{add(0, Y, Y)}$.
- Rules, e.g., $\mathtt{add(s(X), Y, s(Z))} \leftarrow \mathtt{add(X, Y, Z)}$.

The operational semantics is called SLD-resolution:

$$\overbrace{\underline{A_1}, \ldots, A_n}^{goal} \rightarrow_\sigma (B_1, \ldots, B_m, A_2, \ldots, A_n)\sigma$$

if
- there is a program clause $H \leftarrow B_1, \ldots, B_m$
- σ is the mgu of A_1 and H

  (substitution σ is a unifier of A and B if $A\sigma = B\sigma$)

A derivation is successful if it ends with the empty goal ($\Box$)
The computed answer is then the composition of the mgu's

Logic programming: a very brief introduction

A program is given by a set of clauses:

- Facts, e.g., $\mathtt{add(0, Y, Y).}$
- Rules, e.g., $\mathtt{add(s(X), Y, s(Z))} \leftarrow \mathtt{add(X, Y, Z).}$

The operational semantics is called SLD-resolution:

$$\overbrace{\underline{A_1}, \ldots, A_n}^{goal} \rightarrow_\sigma (B_1, \ldots, B_m, A_2, \ldots, A_n)\sigma$$

if
- there is a program clause $H \leftarrow B_1, \ldots, B_m$
- σ is the mgu of A_1 and H

  (substitution σ is a unifier of A and B if $A\sigma = B\sigma$)

A derivation is successful if it ends with the empty goal ($\square$)
The computed answer is then the composition of the mgu's

Example: ancestor relation

$$\begin{array}{ll}
(1) & \texttt{parent(john,peter).} \\
(2) & \texttt{parent(peter,emma).} \\
(3) & \texttt{ancestor(A,B)} \leftarrow \texttt{parent(A,B).} \\
(4) & \texttt{ancestor(A,B)} \leftarrow \texttt{parent(A,C),ancestor(C,B).}
\end{array}$$

E.g.,

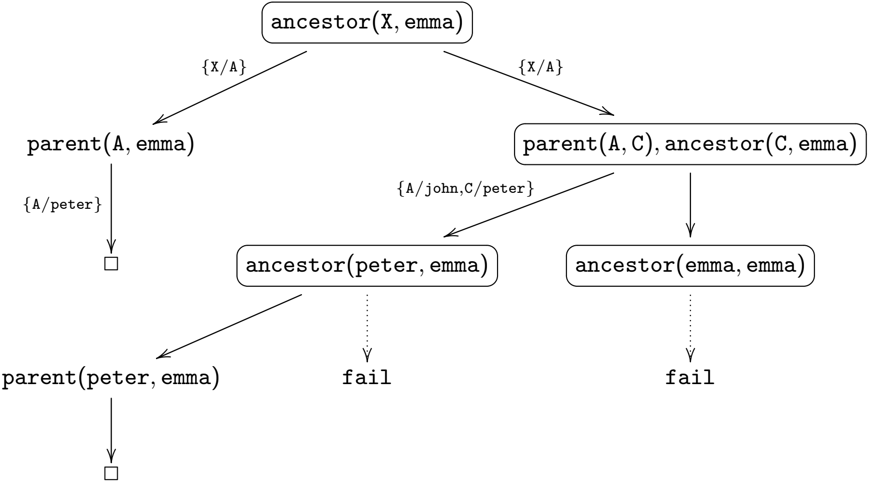
$$\begin{array}{lll}
\texttt{ancestor(X,emma)} & \rightarrow_{\{\texttt{X/A,B/emma}\}} & \texttt{parent(A,emma)} \\
 & \rightarrow_{\{\texttt{A/peter}\}} & \Box
\end{array}$$

with computed answer: $\{\texttt{X/peter}\}$, and

$$\begin{array}{lll}
\texttt{ancestor(X,emma)} & \rightarrow_{\{\texttt{X/A}\}} & \texttt{parent(A,C),ancestor(C,emma)} \\
 & \rightarrow_{\{\texttt{A/john,C/peter}\}} & \texttt{ancestor(peter,emma)} \\
 & \rightarrow & \texttt{parent(peter,emma)} \\
 & \rightarrow & \Box
\end{array}$$

with computed answer: $\{\texttt{X/john}\}$

SLD tree



ancestor(X, emma)

{X/A}

{X/A}

parent(A, emma)

parent(A, C), ancestor(C, emma)

{A/peter}

{A/john,C/peter}

□

ancestor(peter, emma)

ancestor(emma, emma)

parent(peter, emma)

fail

fail

□

Motivation

$$\begin{array}{ll} (1) & \mathtt{p(b,b,Y)} \leftarrow \mathtt{q(Y),r(Y,Y).} \\ (2) & \mathtt{q(b).} \\ (3) & \mathtt{r(b,b).} \end{array}$$

Given the goal $\mathtt{p(X,b,b),r(b,X)}$, we have:

$$\underline{\mathtt{p(X,b,b)}},\mathtt{r(b,X)} \rightarrow_\theta \underline{\mathtt{q(b)}},\mathtt{r(b,b)},\mathtt{r(b,b)} \rightarrow \underline{\mathtt{r(b,b)}},\mathtt{r(b,b)} \rightarrow \ldots$$

with $\theta = \{\mathtt{X/b},\mathtt{Y/b}\}$

In order to undo, e.g., the first step, we face several problems:

Motivation

$$\begin{array}{ll} (1) & \mathtt{p(b,b,Y)} \leftarrow \mathtt{q(Y)}, \mathtt{r(Y,Y)}. \\ (2) & \mathtt{q(b)}. \\ (3) & \mathtt{r(b,b)}. \end{array}$$

Given the goal $\mathtt{p(X,b,b)}, \mathtt{r(b,X)}$, we have:

$$\underline{\mathtt{p(X,b,b)}}, \mathtt{r(b,X)} \rightarrow_{\theta} \underline{\mathtt{q(b)}}, \mathtt{r(b,b)}, \mathtt{r(b,b)} \rightarrow \underline{\mathtt{r(b,b)}}, \mathtt{r(b,b)} \rightarrow \ldots$$

with $\theta = \{\mathtt{X/b}, \mathtt{Y/b}\}$

In order to undo, e.g., the first step, we face several problems:

- First, one needs to know the **applied rule**

  $\underline{\mathtt{q(b)}}, \mathtt{q(b)}, \mathtt{r(b,b)}, \mathtt{r(b,b)} \rightarrow \mathtt{q(b)}, \mathtt{r(b,b)}, \mathtt{r(b,b)}$

Motivation

$$\begin{array}{ll}(1) & \mathtt{p(b,b,Y)} \leftarrow \mathtt{q(Y),r(Y,Y).}\\ (2) & \mathtt{q(b).}\\ (3) & \mathtt{r(b,b).}\end{array}$$

Given the goal $\mathtt{p(X,b,b),r(b,X)}$, we have:

$$\underline{\mathtt{p(X,b,b)}},\mathtt{r(b,X)} \to_\theta \underline{\mathtt{q(b)}},\mathtt{r(b,b)},\mathtt{r(b,b)} \to \underline{\mathtt{r(b,b)}},\mathtt{r(b,b)} \to \ldots$$

with $\theta = \{\mathtt{X/b},\mathtt{Y/b}\}$

In order to undo, e.g., the first step, we face several problems:

- Second, we need to **unapply** the computed substitution (θ) E.g., given the last atom $\mathtt{r(b,b)}$ in the second query, we can undo the application of θ and get $\mathtt{r(b,X)}$ but also $\mathtt{r(X,b)}$ or $\mathtt{r(X,X)}$

Motivation

(1) $\mathtt{p(b,b,Y)} \leftarrow \mathtt{q(Y),r(Y,Y).}$
(2) $\mathtt{q(b).}$
(3) $\mathtt{r(b,b).}$

Given the goal $\mathtt{p(X,b,b),r(b,X)}$, we have:

$$\underline{\mathtt{p(X,b,b)}},\mathtt{r(b,X)} \rightarrow_\theta \underline{\mathtt{q(b)}},\mathtt{r(b,b)},\mathtt{r(b,b)} \rightarrow \underline{\mathtt{r(b,b)}},\mathtt{r(b,b)} \rightarrow \ldots$$

with $\theta = \{\mathtt{X/b},\mathtt{Y/b}\}$

In order to undo, e.g., the first step, we face several problems:

- Finally, we have no deterministic way to obtain the **selected call**

First try: a trivial Landauer embedding

Add a history that stores all goals in a derivation:

$$\begin{array}{l}\langle p(X,b,b), r(b,X); [\,]\rangle \leadsto_{\theta} \langle q(b), r(b,b), r(b,b); [p(X,b,b), r(b,X)]\rangle \\ \quad\leadsto \langle r(b,b), r(b,b); [q(b), r(b,b), r(b,b); p(X,b,b), r(b,X)]\rangle \\ \quad\leadsto \ldots\end{array}$$

Problem: The overhead would be very high. . .

First try: a trivial Landauer embedding

Add a history that stores all goals in a derivation:

$$\begin{array}{l}\langle p(X,b,b), r(b,X); [\,]\rangle \rightsquigarrow_\theta \langle q(b), r(b,b), r(b,b); [p(X,b,b), r(b,X)]\rangle \\ \quad \rightsquigarrow \langle r(b,b), r(b,b); [q(b), r(b,b), r(b,b); p(X,b,b), r(b,X)]\rangle \\ \quad \rightsquigarrow \ \ldots\end{array}$$

Problem: The overhead would be very high. . .

Second try

One of the main problems is undoing the application of a mgu

⇒ consider some non-standard goals where computed substitutions (mgu's) are not applied to the atoms of the goal but stored in a list

For instance, one could redefine SLD resolution as follows:

$$\begin{array}{l}\langle A_1, \ldots, A_k; [\theta_1, \ldots, \theta_n] \rangle \\ \rightarrow_{\theta_{n+1}} \langle B_1, \ldots, B_m, A_2, \ldots, A_k; [\theta_1, \ldots, \theta_n, \theta_{n+1}] \rangle\end{array}$$

if $H \leftarrow B_1, \ldots, B_m \ll P$ and $\mathrm{mgu}(A_1\theta_1 \ldots \theta_n, H) = \theta_{n+1}$

(an initial query A would now have the form $\langle A; [\,] \rangle$)

Second try

One of the main problems is undoing the application of a mgu

⇒ consider some non-standard goals where computed substitutions (mgu's) are not applied to the atoms of the goal but stored in a list

For instance, one could redefine SLD resolution as follows:

$$\begin{array}{l} \langle A_1, \ldots, A_k; [\theta_1, \ldots, \theta_n] \rangle \\ \rightarrow_{\theta_{n+1}} \langle B_1, \ldots, B_m, A_2, \ldots, A_k; [\theta_1, \ldots, \theta_n, \theta_{n+1}] \rangle \end{array}$$

if $H \leftarrow B_1, \ldots, B_m \ll P$ and $\mathrm{mgu}(A_1\theta_1 \ldots \theta_n, H) = \theta_{n+1}$

(an initial query A would now have the form $\langle A; [\,] \rangle$)

Second try

One of the main problems is undoing the application of a mgu

$\Rightarrow$ consider some non-standard goals where computed substitutions (mgu's) are not applied to the atoms of the goal but stored in a list

For instance, one could redefine SLD resolution as follows:

$$\begin{array}{l} \langle A_1, \ldots, A_k; [\theta_1, \ldots, \theta_n] \rangle \\ \rightarrow_{\theta_{n+1}} \langle B_1, \ldots, B_m, A_2, \ldots, A_k; [\theta_1, \ldots, \theta_n, \theta_{n+1}] \rangle \end{array}$$

if $H \leftarrow B_1, \ldots, B_m \ll P$ and $\mathrm{mgu}(A_1\theta_1 \ldots \theta_n, H) = \theta_{n+1}$

(an initial query A would now have the form $\langle A; [\,] \rangle$)

Second try (cont.)

Not enough... even if we store the (label of the) applied rule

$$\langle A_1, A_2, \ldots, A_k; [\theta_1, \ldots, \theta_n] \rangle$$
$$\rightarrow_{\theta_{n+1}} \langle B_1, \ldots, B_m, A_2, \ldots, A_k; [\theta_1, \ldots, \theta_n, \theta_{n+1}] \rangle$$

if $H \leftarrow B_1, \ldots, B_m \ll P$ and $\mathrm{mgu}(A_1\theta_1 \ldots \theta_n, H) = \theta_{n+1}$

Let $\theta_{n+1} = \mathrm{mgu}(A_1, H)$. Given θ_{n+1} and H, we don't know how to obtain A_1

(ok in functional programming because of the use of pattern matching)

Second try (cont.)

Not enough… even if we store the (label of the) applied rule

$$\begin{array}{l} \langle A_1, A_2, \ldots, A_k; [\theta_1, \ldots, \theta_n]\rangle \\ \rightarrow_{\theta_{n+1}} \langle B_1, \ldots, B_m, A_2, \ldots, A_k; [\theta_1, \ldots, \theta_n, \theta_{n+1}]\rangle \\ \text{if } H \leftarrow B_1, \ldots, B_m \ll P \text{ and } \mathrm{mgu}(A_1\theta_1 \ldots \theta_n, H) = \theta_{n+1} \end{array}$$

Let $\theta_{n+1} = \mathrm{mgu}(A_1, H)$. Given θ_{n+1} and H, we don't know how to obtain A_1

(ok in functional programming because of the use of pattern matching)

Third try: our solution

Basically, we need a history with

- selected call
- the head of the clause
- number of atoms introduced
  (i.e., the number of elements in the body of the applied rule)

Reversible (forward) SLD-resolution

(success)

$$\frac{\mathsf{subst}(\mathcal{H}) = \sigma}{\langle \Box; \mathcal{H}\rangle \rightharpoonup \langle \textsc{success}(\sigma); \mathcal{H}\rangle}$$

(failure)

$$\frac{\mathsf{subst}(\mathcal{H}) = \sigma \wedge \not\exists H \leftarrow B_1, \ldots, B_m \ll P \text{ such that } \mathsf{mgu}(A\sigma, H) \neq \mathsf{fail}}{\langle A, \mathcal{B}; \mathcal{H}\rangle \rightharpoonup \langle \textsc{fail}; \mathsf{fail}(A, \mathcal{B}) : \mathcal{H}\rangle}$$

(unfold)

$$\frac{\mathsf{subst}(\mathcal{H}) = \sigma \wedge \exists H \leftarrow B_1, \ldots, B_m \ll P \text{ such that } \mathsf{mgu}(A\sigma, H) \neq \mathsf{fail}}{\langle A, \mathcal{B}; \mathcal{H}\rangle \rightharpoonup \langle B_1, \ldots, B_m, \mathcal{B}; \mathsf{unf}(A, H, m) : \mathcal{H}\rangle}$$

Reversible (backward) SLD-resolution

$$(\overline{\mathsf{success}})\quad \langle \textsc{success}(\sigma); \mathcal{H}\rangle \leftharpoondown \langle \square; \mathcal{H}\rangle$$

$$(\overline{\mathsf{failure}})\quad \langle \textsc{fail}; \mathsf{fail}(A,\mathcal{B}):\mathcal{H}\rangle \leftharpoondown \langle A,\mathcal{B}; \mathcal{H}\rangle$$

$$(\overline{\mathsf{unfold}})\quad \langle B_1,\ldots,B_m,\mathcal{B}; \mathsf{unf}(A,H,m):\mathcal{H}\rangle \leftharpoondown \langle A,\mathcal{B}; \mathcal{H}\rangle$$

DEMO

https://github.com/mistupv/Prolog-reversible-debugger

Future work

- Formally define the debugger and its properties
- Improve debugger performance
- Possible extensions of the debugger:
  - break points
  - deterministic semantics (to undo backtracking steps)
  - . . .

Future work

- Formally define the debugger and its properties
- Improve debugger performance
- Possible extensions of the debugger:
  - break points
  - deterministic semantics (to undo backtracking steps)
  - ...

Future work

- Formally define the debugger and its properties
- Improve debugger performance
- Possible extensions of the debugger:
    - break points
    - deterministic semantics (to undo backtracking steps)
    - . . .

Thanks for your attention !