On Rewrite Programs: Semantics and Relationship with Prolog*

(joint work with Jieh Hsiang at SUNY Stony Brook)

Maria Paola Bonacina
INRIA - LORRAINE, FRANCE

* work done at SUNY STONY BROOK

Rewrite Programs

- Rewrite programs as functional programs [Hoffman - O'Donnell 1982]

$$t \longrightarrow \boxed{E} \longrightarrow t\downarrow_E$$

Data are first order terms.

- Rewrite programs as logical programs [Dershowitz - Josephson 1984]

$$\text{query} \longrightarrow \boxed{E} \longrightarrow \text{answer substitution}$$

Data are first order atoms:
relational language.

Interpreter : Linear Completion.

Outline

- Rewrite programs
- Examples: different behaviour of rewrite programs and Prolog programs.
- Operational semantics
- Denotational semantics
- Comparison with Prolog.
- Comparison with subsumption-based loop-checking mechanisms in Prolog.

append([], L, L).

append([X|L1], Y, [X|L2]) :- append(L1, Y, L2).

?- append(X, [b|Y], [a, b, c|Z]).

?- append(L1, [b|Y], [b, c|Z]).

σ_1: X ← a
Y ← [c|Z]

?- append(L2, [b|Y], [c|Z]).

?- append(L3, [b|Y], Z).

σ_2: X ← [a, b, c]
Z ← [b|Y]

?- append(L4, [b|Y], L2).

σ_3: X ← [a, b, c, X']
Z ← [X', b|Y]

?- append(L5, [b|Y], L3).

σ_4: X ← [a, b, c, X', X'']
Z ← [X', X'', b|Y]

?- append(L6, [b|Y], L4)

. . . . .

infinite loop

$append([\,], L, L) \to true$

$append([X|L_1], Y, [X|L_2]) \to append(L_1, Y, L_2)$

$append(X, [b|Y], [a,b,c|Z]) \to \underline{answer}(X, Y, Z)$

$append(L_1, [b|Y], [b,c|Z]) \to answer([a|L_1], Y, Z)$

$\boxed{ans([a], [c|Z], Z) \to true \quad (\sigma_1)}$

$append(L_2, [b|Y], [c|Z]) \to answer([a,b|L_2], Y, Z)$

(G1) $append(L_3, [b|Y], Z) \to answer([a,b,c|L_3], Y, Z)$

$\boxed{ans([a,b,c], Y, [b|Y]) \to true \quad (\sigma_2)}$

(G2) $append(L_4, [b|Y], L_2) \to answer([a,b,c,X'|L_4], Y, [X'|L_2])$

$\Big\downarrow$ G1

$answer([a,b,c|L_4], Y, L_2) \longleftrightarrow answer([a,b,c,X'|L_4], Y, [X'|L_2])$

<u>2 answers</u> ; <u>no</u> loop !

Prolog:

$\text{append}([X|L_1], Y, [X|L_2])$ if $\text{append}(L_1, Y, L_2)$

gives $\sigma_1, \sigma_2, \ldots\ldots \sigma_i \ldots\ldots$

Rewrite programs:

$\text{append}([X|L_1], Y, [X|L_2])$ iff $\text{append}(L_1, Y, L_2)$

gives σ_1 and σ_2 only.

$$Q = \text{append}(X, [b|Y], [a,b,c|Z])$$

$$\begin{tikzcd}
Q\sigma_1 \arrow[d, "*"] \\
\text{append}([\,], [b,c|Z], [b,c|Z]) \arrow[d] \\
\text{true}
\end{tikzcd}$$

$$\begin{tikzcd}
Q\sigma_2 \arrow[drr, "*"] & , & Q\sigma_3 \arrow[d, "*"] & \ldots\ldots & Q\sigma_n \arrow[dll, "*"] & \ldots\ldots \\
& & \text{append}([\,], [b|Y], [b|Y]) \arrow[d] & & & \\
& & \text{true} & & &
\end{tikzcd}$$

Prolog:

```
append([ ], L, L).
append([X|L1], Y, [X|L2]) :- append(L1, Y, L2).

P(X, Y, Z) :-
append(X, [b|Y], [a,b,c|Z]), non-member(a, X).

P(X, Y, Z) :- ...

?- P(X, Y, Z)
```

loops on the first clause

Rewrite programs:

fails on the first clause for P and applies the others.

$$\Downarrow$$

Loop avoidance capability.

Suppose we define :

```
P(X, Y, Z) :- append(X, [b|Y], [a,b,c|Z]),
              size(X) > 3.
```

Do we get any answers ?!!

$append(X, [b|Y], [a,b,c|Z]), size(X) > 3 \rightarrow ans(X, Y, Z)$

$size([a]) > 3 \rightarrow ans([a], [c|Z], Z)$

(G1) $append(L_3, [b|Y], Z), size([a,b,c]) > 3 \rightarrow ans([a,b,c|L_3], Y, Z)$

$size([a,b,c]) > 3 \rightarrow ans([a,b,c], Y, [b|Y])$

$append(L_4, [b|Y], L_2), size([a,b,c,X'|L_4]) > 3 \rightarrow ans([a,b,c,X'|L_4], Y, [X'|L_2])$

$size([a,b,c,X'|L_4]) > 3 \xrightarrow{*} true$

$\downarrow *$

(G2) $append(L_4, [b|Y], L_2) \rightarrow ans([a,b,c,X'|L_4], Y, [X'|L_2])$

$ans([a,b,c,X'], Y, [X', b|Y]) \rightarrow true$

$append(L_5, [b|Y], L_3) \rightarrow ans([a,b,c,X',X''|L_5], Y, [X',X''|L_3])$

$\downarrow G2$

$ans([a,b,c,X'|L_5], Y, [X'|L_3]) \leftrightarrow ans(\ldots)$

halt

```
ancestor(X,Y) :- parent(X,Y).

ancestor(X,Y) :- parent(Z,Y),
                 ancestor(X,Z).
```

are intended to be implications,

not logical equivalences.

How do we express them in a rewrite program?

Since $\underline{A \supset B}$ is equivalent to

$\underline{A \wedge B \equiv A}$,

we have

$$ancestor(X,Y),\ parent(X,Y) \rightarrow parent(X,Y)$$

$$ancestor(X,Y),\ parent(Z,Y),\ ancestor(X,Z)$$
$$\longrightarrow\ parent(Z,Y),\ ancestor(X,Z).$$

Syntax of rewrite programs

- Fact rules : $A \to true$.

- Iff-rules : $A \to B_1 \ldots B_n$

  meaning A iff $B_1 \ldots B_n$.

- If-rules : $A\, B_1 \ldots B_n \to B_1 \ldots B_n$

  meaning A if $B_1 \ldots B_n$.

The user may choose whether to define a predicate by iff-rules or by if-rules.

Automatic translation of Prolog programs into rewrite programs

$A \text{:-} B_1 \ldots B_n \approx$

- $A \rightarrow B_1 \ldots B_n$

  if A is mutually exclusively defined (no two heads of Prolog rules for A unify) and $A > B_1 \ldots B_n$ ($>$ well-founded),

- $A\, B_1 \ldots B_n \rightarrow B_1 \ldots B_n$

  otherwise.

$? \text{-}\ B_1 \ldots B_n \quad \approx \quad B_1 \ldots B_n \rightarrow answer(\bar{X})$

Linear Completion

State of the computation:

$$(\underbrace{E}_{\text{program rules}};\ \underbrace{\bar{L} \to \bar{R}}_{\text{current goal}};\ \underbrace{S}_{\text{ancestor goals}})$$

Inference rules:

Answer $$\frac{(E;\ answer(\bar{x})\sigma \to true;\ S)}{(E \cup \{answer(\bar{x})\sigma \to true\};\ -;\ S)}$$

Delete $$\frac{(E;\ L_1 \ldots L_n \leftrightarrow L_1 \ldots L_n;\ S)}{(E;\ -;\ S)}$$

Orient $$\frac{(E;\ L_1 \ldots L_n \leftrightarrow R_1 \ldots R_m;\ S)}{(E;\ L_1 \ldots L_n \to R_1 \ldots R_m;\ S)}$$

if $L_1 \ldots L_n > R_1 \ldots R_m$

Simplification

$$\frac{(E;\ L_1 \dots L_n \to R_1 \dots R_m;\ S)}{(E;\ L_1' \dots L_n' \to R_1' \dots R_m';\ S)}$$

The current goal may be simplified by:

- program rules (in E)
- ancestors (in S)
- previously generated answers (in E)
- $x \cdot x \to x$
  $x \cdot true \to x$
  $true \cdot x \to x$

Overlap

$$\frac{(E\,;\ L_1 \ldots L_n \rightarrow R_1 \ldots R_m\,;\ S)}{(E\,;\ L'_1 \ldots L'_p \leftrightarrow R'_1 \ldots R'_q\,;\ S \cup \{L_1 \ldots L_n \rightarrow R_1 \ldots R_m\})}$$

with an *if-rule*:

$$\frac{A\bar{B} \rightarrow \bar{B} \qquad A'\bar{L} \rightarrow \bar{R}}{(\bar{B}\bar{L})\sigma \rightarrow (\bar{B}\bar{R})\sigma} \quad A\sigma = A'\sigma$$

with an *iff-rule*:

$$\frac{A \rightarrow \bar{B} \qquad A'\bar{L} \rightarrow \bar{R}}{(\bar{B}\bar{L})\sigma \rightarrow \bar{R}\sigma} \quad A\sigma = A'\sigma$$

with a *fact rule*:

$$\frac{A \rightarrow \mathrm{true} \qquad A'\bar{L} \rightarrow \bar{R}}{\bar{L}\sigma \rightarrow \bar{R}\sigma} \quad A\sigma = A'\sigma$$

Soundness of Linear Completion

$E \vdash_{LC} Q_1 \ldots Q_m \sigma$ implies

$E^* \models Q_1 \ldots Q_m \sigma = true.$

where

$E \vdash_{LC} Q_1 \ldots Q_m \sigma$ stands for

$$E \cup \{ Q_1 \ldots Q_m \to answer(\bar{x}) \} \vdash_{LC} answer(\bar{x})\sigma \to true$$

and

$$E^* = E \cup \{ x \cdot x \to x , \ x \cdot true \to x, \ true \cdot x \to x \}$$

Denotational semantics

E: rewrite program

$\mathcal{B}$: Herbrand base

$$\mathbb{B} = \{ I' \mid I' = I \cup \{true\},\ I \subseteq \mathcal{B} \}$$

Lattice: $< \mathbb{B}, \subseteq, \cap, \cup, \{true\}, \mathcal{B} \cup \{true\} >$

$$\boxed{T_E : \mathbb{B} \longrightarrow \mathbb{B}}$$

$\forall I \in \mathbb{B} \quad \forall P \in \mathcal{B} \qquad P \in T_E(I)$ iff

$\exists \quad A_1 \ldots A_n \longleftrightarrow B_1 \ldots B_m \in E$

$\exists \sigma \ \exists i \qquad P = A_i \sigma$

$A_1\sigma \ldots A_{i-1}\sigma, A_{i+1}\sigma \ldots A_n\sigma, B_1\sigma \ldots B_m\sigma \in I.$

Denotational semantics

T_E is continuous.

Thus: $\boxed{lfp(T_E) = T_E \uparrow \omega}$

where

$T \uparrow \emptyset = \{true\}$

$$T \uparrow n = \begin{cases} T(T \uparrow (n-1)) & \text{if } n \text{ is a successor ordinal,} \\ \bigcup \{T \uparrow k \mid k < n\} & \text{if } n \text{ is a limit ordinal.} \end{cases}$$

Equivalence of operational, model-theoretic, denotational semantics

E: rewrite program

$\mathcal{B}$: Herbrand base

Operational semantics:

$\{ G \mid G \in \mathcal{B},\ E \vdash_{LC} G \}$ "success set"

Model theoretic semantics:

$\{ G \mid G \in \mathcal{B},\ E^* \models G = true \}$

Denotational semantics: $lfp(T_E)$

$\forall\, G \in \mathcal{B}$

$$\boxed{E \vdash_{LC} G \text{ iff } E^* \models G = true \text{ iff } G \in lfp(T_E)}$$

Comparison with Prolog

P: Prolog program

E: rewrite program

If $E \equiv P$, then:

- $\boxed{lfp(T_P) = lfp(T_E)}$,

- if $E^* \models G\vartheta = true$, $\exists \sigma$ $E \vdash_{Lc} G\sigma$
  such that $G\vartheta \xleftrightarrow[E^*]{*} G\sigma\rho$ for some ρ,

- if $P \vdash G\vartheta$, $\exists \sigma$ $E \vdash_{Lc} G\sigma$
  such that $G\vartheta \xleftrightarrow[E^*]{*} G\sigma\rho$ for some ρ.

Comparison with subsumption-based loop checking mechanisms for Prolog

[Bol, Apt, Klop 1990]

p(a).

p(Y) :- p(Z).

?-p(X)

$\boxed{X \leftarrow a}$?-p(Z) SIG, SVG, EIG, EVG

$\boxed{X \leftarrow Y}$ . . . .

loop

$p(a) \rightarrow true$

$p(Y)\ p(Z) \rightarrow p(Z)$

$p(X) \rightarrow answer(X)$ (G∅)

answer(a) → true p(Z), answer(X) → p(Z)

$\downarrow$ G∅

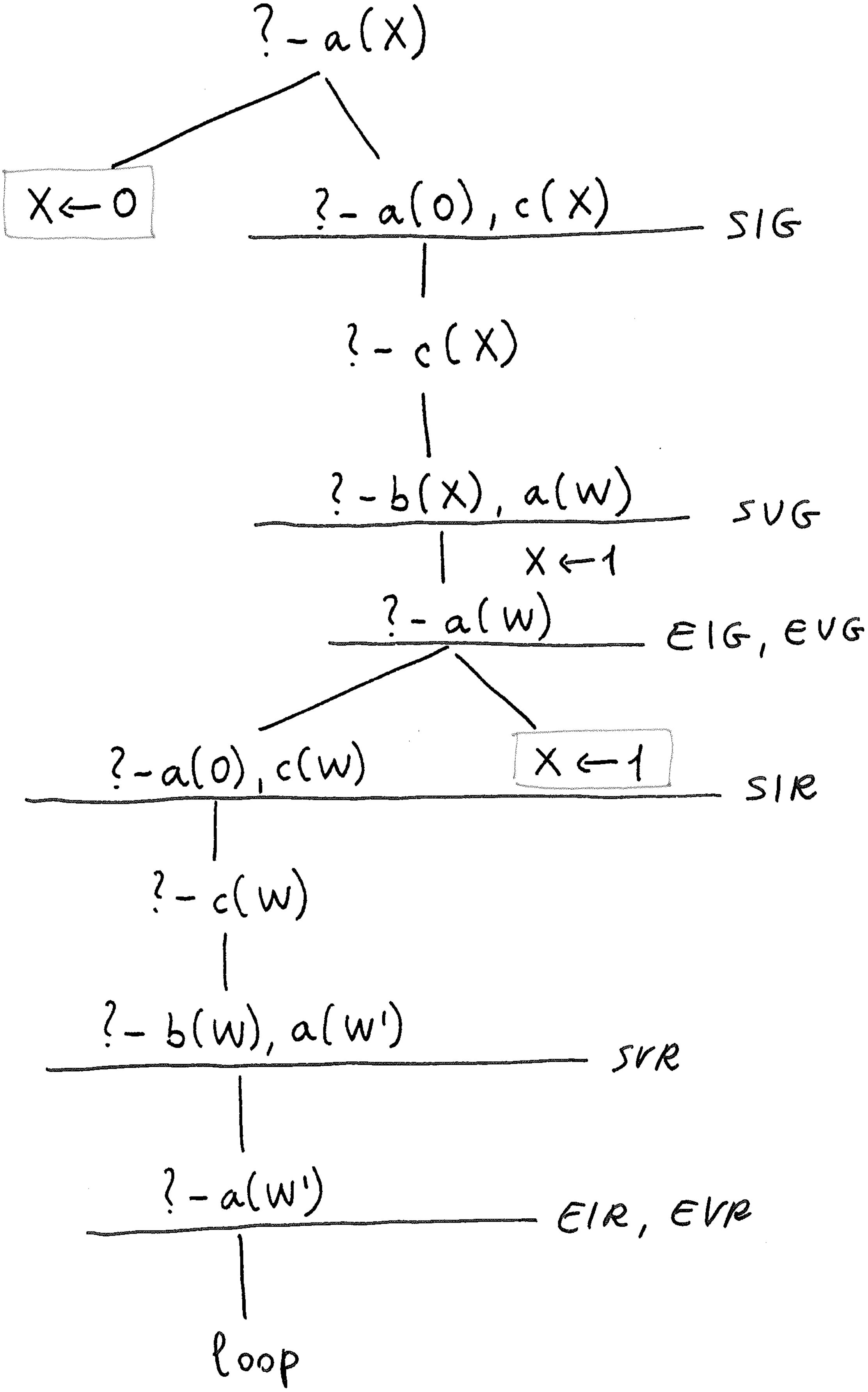
ans(Z), ans(X) → ans(Z)

$\downarrow$ $Z \leftarrow a$

ans(X) → true

halt

```
a(0).
a(Y) :- a(0), c(Y).
b(1).
c(Z) :- b(Z), a(W).
```



$a(0) \to true$ (F1)

$a(Y), a(0), c(Y) \to a(0), c(Y)$

$b(1) \to true$

$c(Z) \to b(Z), a(W)$

$a(X) \to answer(X)$

$a(X) \to answer(X)$ $(G\emptyset)$

$\underline{answer(0) \to true}$

$a(0), c(X), ans(X) \to a(0), c(X)$

$\big\downarrow$ F1

$c(X), answer(X) \to c(X)$

$\big\downarrow$

$b(X), a(W), ans(X) \to b(X), a(W)$

$\big\downarrow$ $G\emptyset$

$b(X), ans(W), ans(X) \to b(X), ans(W)$

$|$ $X \leftarrow 1$

$ans(W), ans(1) \to ans(W)$

$|$ $W \leftarrow 0$

$\underline{answer(1) \to true}$

halt

Discussion

- Rewrite programs are not the same as Prolog.
- Use of logical equivalences.
- Mutually exclusively defined predicates.
- Simplification.
- It prunes equivalent answers: same fix point, fewer answers.
- It prevents loops.
- Identification of the generated answers (refinement of $\overset{*}{\underset{E}{\longleftrightarrow}}$).
- Treatment of negation.