Lollipops taste of Vanilla too

Post-ICLP'94 Workshop on

Proof-Theoretical Extensions of Logic Programming

Iliano Cervesato

Dipartimento di Informatica
Università di Torino
Corso Svizzera, 185
10149 Torino - Italy

iliano@di.unito.it

S. Margherita Ligure, Italy
June 18th, 1994

Overview

- Meta-programming in logic programming
  - Meta-programs
  - The current trend
  - Granularity of a meta-program
  - The Vanilla meta-interpreter for Prolog
- The linear logic programming language Lolli
  - Linear logic
  - Uniform proofs
  - The $\top$, &, $\multimap$, $\Rightarrow$, $\forall$ fragment
  - Lolli
  - Operational semantics
- A Vanilla meta-interpreter for Lolli
  - Syntactic restrictions
  - The representation function
  - A vanilla meta-interpreter for the core of Lolli
  - Soundness of the meta-interpreter
  - Extensions and examples of use
- Conclusions and future work

Meta-programs

... are *programs that treat other programs as data*.

Advantages:

- powerful extensions of the base language are easily encoded
- programming tools can be developed as needed
- is a valid support to rapid prototyping

Drawbacks:

- low efficiency (in part recovered through partial evaluation)

Applications:

- software development (integrated programming environments)
- software analysis
- artificial intelligence (knowledge based systems)
- theorem proving

The current trend

There have been many proposal to improve the meta-programming capabilities of Prolog:

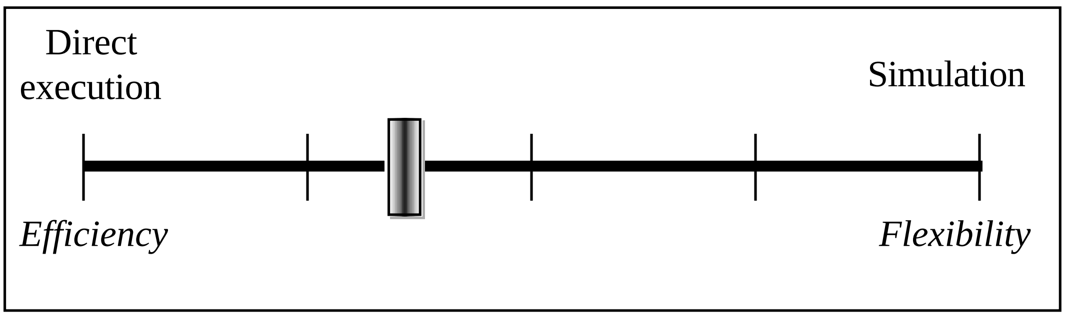
- 1982: Bowen & Kowalski's paper
- 1985: *MetaProlog* (Bowen et al.)
- 1989: *Reflective Prolog* (Costantini, Lanzarone)
- 1990: *Gödel* (Hill, Lloyd)
- 1991: *'Log* (Cervesato, Rossi)

No or little work has been done to investigate the meta-programming attitude of the *new generation logic programming languages* (λ-Prolog, Lolli, Forum, Elf, ...).

The underlying theory makes new techniques available to explore the properties of the meta-programs.

The granularity of a meta-program

... is the ratio between what is simulated and what is passed over to the underlying interpreter



Examples:

- ```
  solve(G) :- G.
  ```

Vanilla

- ```
  solve(true).
  solve((A,B)) :-
      solve(A), solve(B).
  solve(A) :-
      clause(A,B), solve(B).
  ```

[Bowen-Kowalski]

- ```
  solve(P,Gs) :- empty(G).
  solve(P,Gs) :-
      select(Gs,G,Gs'), member(C,P),
      rename(C,G,C'),   parts(C',H,B),
      unify(H,G,S),     append(G',B,G"),
      apply(S,G",G'"),  solve(P,G'").
  ```
  ```

The Vanilla meta-interpreter for Prolog

```
solve(true).

solve((A,B)) :-
    solve(A),
    solve(B).

solve(A) :-
    clause(A,B),
    solve(B).
```

- *Vanilla* is a medium granularity meta-interpreter
- it achieves the functionalities of the Horn core of Prolog

Extra-logical functionalities can be added:

```
solve((A;B)) :-
    solve(A);
    solve(B).
```

Disjunction

```
solve(not A) :-
    not solve(A).
```

Negation as failure

```
solve(bagof(X,G,Xs) :-
    bagof(X,solve(G),Xs).
```

Grouping

```
solve(A) :-
    functor(A,F,N),
    system(F,N),
    A.
```

System calls

Vanilla can be enhanced to deal correctly with control directives such as *cut* (`!`).

Linear logic

... refines traditional logic by constraining the number of times an assumption is used in a proof.

Weakening and *contraction* are ruled out.

This calls for a finer set of connectives:

Connectives								Context
Traditional	T	F	$\neg$	$\wedge$	$\vee$		$\rightarrow$	Unbound
Multiplicative	**1**	$\perp$	$^{\perp}$	$\otimes$	$\wp$		$\multimap$	Split
Additive	T	**0**		&	$\oplus$			Copy
Exponential			!			?		Unbound

Example of sequent rules:

$$\frac{\Delta_1 \xrightarrow{ll} B \quad \Delta_2 \xrightarrow{ll} C}{\Delta_1, \Delta_2 \xrightarrow{ll} B \otimes C} \otimes L \qquad \frac{\Delta \xrightarrow{ll} B \quad \Delta \xrightarrow{ll} C}{\Delta \xrightarrow{ll} B \& C} \& L$$

Controlled forms of *weakening* and *contraction* are made availabe through the use of ! and ?

$$\frac{\Delta \xrightarrow{ll} E}{\Delta, !B \xrightarrow{ll} E} !W \qquad \frac{\Delta, !B, !B \xrightarrow{ll} E}{\Delta, !B \xrightarrow{ll} E} !C \qquad \frac{\Delta, B \xrightarrow{ll} E}{\Delta, !B \xrightarrow{ll} E} !D$$

Uniform proofs

The logical connectives in a *logic programming language* should be used by the interpreter as search directives for finding proofs of goals

Solving a goal G in a program Δ = building a bottom up deduction for the sequent $\Delta \longrightarrow G$

Such a proof must be:

- cut-free
- goal directed

A cut-free proof Ξ of the sequent $\Delta \longrightarrow G$ is *uniform* iff every left rules is applied to sequents with an atomic rhs only.

- Horn intuitionistic logic YES
- Full intuitionistic logic NO!
- Full linear logic NO!

We can identify maximal fragments of logic having the uniform proof property

- hereditary Harrop formulas (the *language freely generated* from T, $\wedge$, $\rightarrow$ and $\forall$)

The $\top$, &, $-\!\circ$, $\Rightarrow$, $\forall$ fragment

Intuitionistic implication: $A \Rightarrow B = {!A} -\!\circ B$

Then, the language freely generated from $\top$, &, $-\!\circ$, $\Rightarrow$, $\forall$ *HAS* the uniform proof property

The uniform proof property is preserved if we allow positive occurrences of **1**, $\oplus$, $\otimes$, ! and $\exists$:

$$R ::= \top \mid A \mid R_1 \,\&\, R_2 \mid G -\!\circ R \mid G \Rightarrow R \mid \forall x.R$$

$$\mathrm{G} ::= \top \mid A \mid G_1 \,\&\, G_2 \mid R -\!\circ G \mid R \Rightarrow G \mid \forall x.G \mid \mathbf{1} \mid G_1 \otimes G_2 \mid G_1 \oplus G_2 \mid {!G} \mid \exists x.G$$

Specialized sequents:

$$\Gamma;\Delta \xrightarrow{ll^-} G$$

Unbound context:
$\Gamma = {!A_1}, \ldots, {!A_n}$
set of R-Formulas

Bound context
multiset of R-Formulas

Goal formula
G-Formula

Sequent calculus rules

$$\frac{}{\Gamma; A \longrightarrow A}\,id \qquad \frac{\Gamma, B; \Delta, B \longrightarrow C}{\Gamma, B; \Delta \longrightarrow C}\,absorb$$

$$\frac{}{\Gamma; \Delta \longrightarrow \top}\,\top R \qquad \frac{}{\Gamma; \emptyset \longrightarrow \mathbf{1}}\,\mathbf{1}R$$

$$\frac{\Gamma; \Delta \longrightarrow B \quad \Gamma; \Delta \longrightarrow C}{\Gamma; \Delta \longrightarrow B \,\&\, C}\,\&R \qquad \frac{\Gamma; \Delta_1 \longrightarrow B \quad \Gamma; \Delta_2 \longrightarrow C}{\Gamma; \Delta_1, \Delta_2 \longrightarrow B \otimes C}\,\otimes R$$

$$\frac{\Gamma; \Delta, B \longrightarrow C}{\Gamma; \Delta \longrightarrow B \multimap C}\,\multimap R \qquad \frac{\Gamma; \emptyset \longrightarrow B}{\Gamma; \emptyset \longrightarrow !B}\,!R$$

$$\frac{\Gamma, B; \Delta \longrightarrow C}{\Gamma; \Delta \longrightarrow B \Rightarrow C}\,\Rightarrow R \qquad \frac{\Gamma; \Delta \longrightarrow B_i}{\Gamma; \Delta \longrightarrow B_1 \oplus B_2}\,\oplus R_i$$

$$\frac{\Gamma; \Delta \longrightarrow B[c/x]}{\Gamma; \Delta \longrightarrow \forall x. B}\,\forall R \qquad \frac{\Gamma; \Delta \longrightarrow B[t/x]}{\Gamma; \Delta \longrightarrow \exists x. B}\,\exists R$$

where t is a term in $\exists R$, and c is not present in the lower sequent of $\forall R$

$$\frac{\Gamma; \emptyset \longrightarrow B_1 \quad \ldots \quad \Gamma; \emptyset \longrightarrow B_n \quad \Gamma; \Delta_1 \longrightarrow C_1 \quad \ldots \quad \Gamma; \Delta_m \longrightarrow C_m}{\Gamma; \Delta_1, \ldots, \Delta_m, B \longrightarrow A}\,BC$$

provided that $n, m \geq 0$, A is atomic and there exist a substitution σ

such that $B^\sigma \equiv D_1 \,\&\, \ldots \,\&\, D_p$

with $D_1 = A \multimap\, !B_1 \otimes \ldots \otimes !B_n \otimes C_1 \otimes \ldots \otimes C_m$

Lolli

Concrete syntax

$\top$	–>>	`erase`
1	–>>	`true`
$!A$	–>>	`{`A`}`
$A \mathbin{\&} B$	–>>	A `&` B
$A \otimes B$	–>>	A `,` B
$A \oplus B$	–>>	A `;` B
$A \multimap B$	–>>	A `-o` B *and* B `:-` A
$A \Rightarrow B$	–>>	A `=>` B *and* B `<=` A
$\forall x.A$	–>>	`forall` x `\` A
$\exists x.A$	–>>	`exists` x `\` A

Example: *toggling a switch* <u>s</u>

```
toggle s G :-
    on s,
    (off s -o G).
toggle s G :-
    off s,
    (on s -o G).
LINEAR off s.
```

Without resource consumption, turning <u>s</u> <u>on</u> would keep it also <u>off</u>

Operational semantics

- $D; \emptyset \vdash_{\text{LOLLI}}$ `true`
- $D; R \vdash_{\text{LOLLI}}$ `erase`
- $D; R \vdash_{\text{LOLLI}} \{G\}$ iff $D; \emptyset \vdash_{\text{LOLLI}} G$
- $D; R_1 + R_2 \vdash_{\text{LOLLI}} G_1 \texttt{,} G_2$ iff $D; R_1 \vdash_{\text{LOLLI}} G_1$ and $D; R_2 \vdash_{\text{LOLLI}} G_2$
- $D; R \vdash_{\text{LOLLI}} G_1 \,\&\, G_2$ iff $D; R \vdash_{\text{LOLLI}} G_1$ and $D; R \vdash_{\text{LOLLI}} G_2$
- $D; R \vdash_{\text{LOLLI}} G_1 \texttt{;} G_2$ iff $D; R \vdash_{\text{LOLLI}} \mathrm{G}_1$ or $D; R \vdash_{\text{LOLLI}} G_2$
- $D; R \vdash_{\text{LOLLI}} r \texttt{-o} G$ iff $D; R + r \vdash_{\text{LOLLI}} G$
- $D; R \vdash_{\text{LOLLI}} d \Rightarrow G$ iff $D \cup d; R \vdash_{\text{LOLLI}} G$
- $D; R \vdash_{\text{LOLLI}} \texttt{forall}\, x \setminus G$ iff $D; R \vdash_{\text{LOLLI}} [c/x]G$ where c is a new constant
- $D; R \vdash_{\text{LOLLI}} \texttt{exists}\, x \setminus G$ iff $D; R \vdash_{\text{LOLLI}} [t/x]G$ for some term t

- $D \cup h \texttt{:-} b; R \vdash_{\text{LOLLI}} A$ if $\sigma = \text{match}(A, h)$ and $D \cup h \texttt{:-} b; R \vdash_{\text{LOLLI}} b^\sigma$
- $D ; R + h \texttt{:-} b \vdash_{\text{LOLLI}} A$ if $\sigma = \text{match}(A, h)$ and $D; R \vdash_{\text{LOLLI}} b^\sigma$
- $D ; R + c_1\&...\&c_{\text{n}} \vdash_{\text{LOLLI}} A$ if $D; R + c_i \vdash_{\text{LOLLI}} A$ for some $i=1...n$

where $\sigma = \text{match}(A, A_1\&...\&A_n)$ iff $A^\sigma = A_i^\sigma$ for some $i=1...n$

Syntactic restrictions

The same Lolli formula can be written in several logically equivalent forms

Ex: `(a&b):-c` $\equiv$ `(a:-c)&(b:-c)`

`(a:-b):-c` $\equiv$ `a:-b,c`

This is annoying for program formulas

Normal form theorem for R-formulas

Every R-formula R is logically equivalent to an R-formula R' of the form:

$\forall \vec{y}_1 \ldots \vec{y}_n (C_1 \& \ldots \& C_n)$ for $n \geq 0$ [T if $n = 0$]

where each C_i has the form $G_i \multimap A_i$
and the $\vec{y}_i$ are disjoint and the only variable occurring in C_i

Proof

by induction on the *degree* of an R-formula.

The representation function

The new grammar:

$$
\begin{array}{lcl}
G & ::= & \top \mid A \mid G_1 \mathbin{\&} G_2 \mid R \multimap G \mid R \Rightarrow G \mid \forall x.G \mid \\
 & & \mathbf{1} \mid G_1 \otimes G_2 \mid G_1 \oplus G_2 \mid\, !G \mid \exists x.G \\
Q & ::= & G
\end{array}
$$

$$
\begin{array}{lcl}
C & ::= & A \mid A \mathrel{\circ\!-} G \mid A \Leftarrow G \\
W & ::= & \top \mid C \mid W_1 \mathbin{\&} W_2 \\
R & ::= & W \mid \forall x.R \\
P & ::= & \varepsilon \mid P\ R
\end{array}
$$

$$\Gamma;\Delta \longrightarrow G$$

P P Q

The representation function τ:

G	$\xrightarrow{\tau}$	G
Q	$\xrightarrow{\tau}$	`solve` Q
C	$\xrightarrow{\tau}$	C
W	$\xrightarrow{\tau}$	W
R	$\xrightarrow{\tau}$	`clause` $W \mid \forall x.R^\tau$
P	$\xrightarrow{\tau}$	P^τ

A Vanilla meta-interpreter for the core of Lolli

```
MODULE meta.

LOCAL   makeClause solveAtomic.
```

```
solve true :- true.

solve (G1 , G2) :-
      solve G1,
      solve G2.

solve (G1 ; G2) :-
      solve G1;
      solve G2.

solve (R -o G) :-
      makeClause R R1,
      R1 -o solve G.

solve (forall G) :-
      forall X \
        solve (G X).

solve A :-
      clause D,
      solveAtomic A D.
```

```
solve erase :- erase.

solve (G1 & G2) :-
      solve G1 &
      solve G2.

solve ({G}) :-
      {solve G}.

solve (R => G) :-
      makeClause R R1,
      R1 => solve G.

solve (exists G) :-
      exists X \
        solve (G X).
```

```
makeClause (forall R) (forall R1) :-
      makeClause (R X) (R1 Y).

makeClause R (clause R).
```

```
solveAtomic A A.

solveAtomic A (R1 & R2) :-
      solveAtomic A R1;
      solveAtomic A R2.
```

```
solveAtomic A (A :- G) :-
      solve G.

solveAtomic A (A <= G) :-
      solve ({G}).
```

We call this program μ

Soundness

Soundness theorem

Let (Γ,Δ) be a restricted Lolli program and G a G-formula. Let $\Gamma^{\tau+}$ be $\Gamma^{\tau} \cup \mu$, then

$$\Gamma;\Delta \longrightarrow G \quad \text{iff} \quad \Gamma^{\tau+};\Delta^{\tau} \longrightarrow G^{\tau}$$

Proof

By induction on the structure of a deduction of

$$\Gamma;\Delta \longrightarrow G$$

Example: the last rules applied were *id* and $\otimes R$

Notice: the proof gives us some insight about the simulation overhead

Soundess proof: *id*

File *iclp_aux.ppt*
–
slide 17

Soundess proof: $\otimes R$

File *iclp_aux.ppt*

–

slide 18

Extensions

Extra-logical features of Lolli

- guarded goals (negation as failure)
- I/O
- arithmetics

Para-logical features of Lolli

- extended syntax
- modules

The meta-interpreter can be easily turned into something useful:

- execution tracer
- debugger
- cost analyser

Examples of use

```
clause (toggle G :-
            on, (off -o G)).
clause (toggle G :-
            off, (on -o G)).
LINEAR clause (on).

?- solve (toggle (off)).
+> toggle off
+* toggle off
|   +> on
|   +- on
|   +> off
|   +- off
+- toggle off
solved
yes
?- solve (toggle (on)).
+> toggle on
+* toggle on
|   +> on
|   +- on
|   +> on
|   +# on        HAS FAILED.
|   +# on        HAS FAILED.
+* toggle on
|   +> off
|   +# off       HAS FAILED.
+# toggle on     HAS FAILED.
no
```

Conclusions and future work

Conclusions

- it is possible to write meta-interpreters for the new generation logic programming languages
- this task is easy
- the underlying proof theory provides a powerful tool for meta-theoretical investigations

Future work

- attempt a proof-theoretical approach to the complexity of meta-interpreters
- apply the technique used to typed languages (λProlog, Elf, ...)