Fondements de la Programmation: Introduction to Logic Programming

Giovanni Coray - Vincenzo Pallotta

June 25, 2000

References:

- "From Logic Programming to PROLOG" - K. Apt, Prentice Hall, 1997
- "Symbolic Logic and Mechanical Theorem Proving" - Chang & Lee, Academic Press, 1973
- "The Art of Prolog" - Sterling & Shapiro, MIT Press, 1986
- "Constraint Logic Programming: A survey" - J. Jaffar and M. Maher, The Journal of Logic Programming Vol. 19 & 20, pp. 503-582, May 1994.

Declarative Programming

- Two equivalent interpretations of programs:
  - declarative = meaning
  - procedural = method
- Specification vs. Algorithms
  - Wirth: Algoritm + Data structures = programs
  - Kowalski: Logic + control = programs
- Towards real software engineering
  - programs are easy to understand and to develop
  - executable specification => rapid prototyping
  - write always "correct" programs!
  - correct prototypes can turned into efficient applications
- Knowledge Representation
  - expert systems
  - inference engines
  - deductive databases

Logic Programming paradigm

Example 1 (Flight connections)

```
direct(amsterdam,seattle).
direct(amsterdam,paramaribo).
direct(seattle,anchorage).
direct(anchorage,fairbank).
connection(X,Y) ← direct(X,Y).
connection(X,Y) ← direct(X,Z), connection(Z,Y).
```

is there any connection from Amsterdam to Fairbanks?

```
?- connection(amsterdam, fairbank).
yes
```

where can one fly from Seattle?

```
?- connection(seattle,X).
X = anchorage;
X = fairbanks;
no
```

...and from Fairbanks?

```
?- connection(fairbanks,X).
no
```


Logic Programming paradigm

Example 2 List Membership

```
member(X,[X|Rest]).
member(X,[Y|Rest]) ← member(X,Rest).
```

is there an apple in the list [apple,pear,plum]?

```
?- member(apple,[apple,pear,plum]).
yes
```

tell me the elements of the list [apple,pear,plum]...

```
?- member(X,[apple,pear,plum]).
X = apple;
X = pear;
X = plum;
no
```

which are the common elements of the two lists [1,2,3,4] and [4,7,1]?

```
common_member(X,L1,L2) ← member(X,L1), member(X,L2).
?- common_member(X,[1,2,3,4],[4,7,1]).
X = 1;
X = 4;
no
```

Logic vs. Imperative Programming

Example 2.1 List Membership

type List = **array**[1..n] **of** integers;
procedure common_members(a,b:List; **var** c:List; **var** x: integer);
var i,j,k: integer;
begin
  k:=1;
  for i:=1 **to** n **do**
    for j:=1 **to** n **do**
      if a[i] = b[j] **then begin**
        c[k]:=a[i]; k:=k+1
      end;
    x:=k-1
end;
a[1]:=1;a[2]:=2;a[3]:=3;a[4]:=4;
b[1]:=4;b[2]:=7;b[3]:=1;
x:=1;
common_members(a,b,c,x);

Advantages of declarative programming

- Specifications are programs
- The output is a logical consequence of the program
- The computation mechanism is not part of the program
- "Thinking" declaratively is easier than "thinking" procedurally (at least for beginners...)
- Declarative programs are therefore much simpler to understand and to develop.

Shortcomings of PROLOG

Termination: PROLOG programs have a strong procedural interpretation. The programmer must be aware of it since:

- PROLOG follows a left-to-right policy for selecting sub-goals (sub-procedure calls).
  Suppose to add the fact `direct(seattle,seattle)` to the program of example 1. If we ask for `?-connect(seattle,X)`, we will have an infinite of answer! This would imply that it is not possible to compute the reflexive and transitive closure of a relation! Solution: DATALOG!
- PROLOG looks up the program data-base sequentially.
  Suppose we change the order of the clauses, thus we may obtain a non terminating program.
- PROLOG does not perform occur-check:

```
?-member(X,[f(X),X]).
X= f(f(f(f(f(f(f(f.....

miracle ← same_person(X,mother_of(X)).
same_person(Z,Z).

?-miracle.

yes
```


Shortcomings of PROLOG

Arithmetic: In PROLOG one cannot perform arithmetic symbolic computations (variable must be instantiated).

Example: (Strange Squares) The square of 45 is 2025. If we split this in two, we get 20 and 25. 20+25 = 45. Find two other number with 4 digits that exhibits the same peculiarity.

```
sol(N,Z) ←
    between(10,99,N),
    Z is N*N,
    Z ≥ 1000,
    Z // 100 + Z mod 100 =:= N.

?- sol(N,Z).

N=45,
Z=2025;

N=55,
Z=3025;

N=99;

Z=9801;
no
```

How to define the predicate between? If we define:

```
between(X,Y,Z) ← X ≤Z, Z ≤Y.
```

then it works only for instantiated variables. We want instead to generate all the integer numbers between X and Y. So we can define a less intuitive program:

```
between(X,Y,Z) ← X ≤Y, Z is X.
between(X,Y,Z) ← X < Y,X1 is X + 1, between(X1,Y,Z).
```

Anyway PROLOG does not answer the query:

```
?- between(X,X+10,Z).
{INSTANTIATION ERROR: in expression}
```

The Language

Alphabet:

Variables: *X,Y,Z,...*
Function symbols: *f,g,h,...*
Constant symbols: *a,b,c,...*
Predicate symbols: *p,q,r,s,...*
Parentheses and connectives: '(', ')', ',' and '$\leftarrow$'

Terms:

- a variable is a term
- $f(t_1,\ldots,t_n)$ is a term iif f is a n-ary function symbol and $t_1,\ldots,t_n$ are terms.
- $Var(t) = \begin{cases} X & if\, t \equiv X \\ \bigcup_{s\in\{t_1,\ldots,t_n\}} Var(s) & if\, t = f(t_1,\ldots,t_n) \end{cases}$
- $ground(t) \Leftrightarrow Var(t) = \emptyset$

Atoms: $p(t_1,\ldots,t_n)$ is an atom iif p is a predicate symbol and $t_1,\ldots,t_n$ are terms.

Queries: $p_1,\ldots,p_n$ is a (conjuntive) query iif $p_1,\ldots,p_n$ are atoms and $0 \leq n < \infty$.

Definite Clauses: $H \leftarrow B$ is a clause iif H (the head) is an atom and B (the body) is a (possibly empty) query. Clauses with empty body are called *facts* or *unitary clauses* and clauses with nonempty body are called *rules*.

Positive Logic Programs: are finite sets of clauses. Denoted by $P, Q, R, \ldots$

Intuitive Meaning of Positive Logic Programs

- A clause $H \leftarrow B_1,\ldots,B_n$ can be understood as the logical formula:

$$\forall X_1,\ldots,X_k : (B_1 \wedge \ldots \wedge B_n \rightarrow H)$$

where $\{X_1,\ldots,X_k\} = Var(H \leftarrow B_1,\ldots,B_n)$.

- Unit clauses $H \leftarrow$ or simply H can be understood as:

$$\forall X_1,\ldots,X_n : H.$$

- A query $A_1,\ldots,A_n$ can be understood as the formula:

$$\exists X_1,\ldots,X_k : (A_1 \wedge \ldots \wedge A_n).$$

- The empty query is equivalent to $true$.

Substitutions

Substitutions: $\theta = \{X_1/t_1,\cdots,X_n/t_n\}$ where

- $n \geq 0$
- $X_1,\cdots,X_n$ are pairwise distinct variables
- $t_1,\ldots,t_n$ are terms such that: $\forall i \in \{1,\ldots,n\} : X_i \neq t_i$
- the *empty substitution* ε is a subistution of length $n = 0$
- θ is *grounding* iif $t_1,\ldots,t_n$ are ground terms
- θ is a *renaming* iff $t_1,\ldots,t_n$ are all variables.
- $Dom(\theta) = \{X_1,\cdots,X_n\}$ and $Range(\theta) = \{t_1,\ldots,t_n\}$

Application of substitutions:

- if $X \in Dom(\theta)$ then $X\theta = \theta(X)$
- if $X \notin Dom(\theta)$ then $X\theta = X$
- $f(t_1,\ldots,t_n)\theta = f(t_1\theta,\ldots,t_n\theta)$
- t is an *instance* of s iff $\exists\theta : t = s\theta$
- s is *more genera*l than t iff t is an instance of s (i.e. $t \sqsubseteq s$)
- t is a *variant* of s iff $s\theta = t$ and θ is a renaming.

Unification

Composition of subsitution: $\theta\eta$ is defined as $(\theta\eta)(X) = (X\theta)\eta$ for all variable X

- $\theta = \{X_1/t_1,\cdots,X_n/t_n\}$, $\eta = \{Y_1/s_1,\cdots,Y_n/s_n\}$, then

$$\theta\eta = \{X_1/t_1\eta,\cdots,X_n/t_n\eta, Y_1/s_1,\cdots,Y_n/s_n\} \setminus \gamma$$

$\gamma = \{X_i/t_i\eta \,|\, X_i = t_i\eta\} \cup \{Y_i/s_i \,|\, Y_i \in \{X_1,\ldots,X_n\}\}$

- θ is *more general* than τ ($\tau \sqsubseteq \theta$) iff $\exists\eta : \tau = \theta\eta$

Unifiers:

- θ is a *unifier* for s and t iff $s\theta = t\theta$
- s and t are *unifiable* iff $\exists\theta : s\theta = t\theta$
- θ is the *most general unifier* (e.g. $mgu(s,t)$) of s and t iff $\forall\tau : s\tau = t\tau \Rightarrow \tau \sqsubseteq \theta$.

Computing the MGU:

There exists an algorithm (Martelli-Montanari) which computes in linear time (if there exists) the $mgu(s,t)$.

Declarative Interpretation

Interpretation: $I_{\mathcal{L}} = \langle D, F, R, \sigma \rangle$ where
$\mathcal{L} = Terms \cup Atoms \cup Queries \cup Clauses \cup Programs$

domains: non empty set D

assignment of mapping: for each n-ary function symbol in the term language f there exists a mapping $f_I : D^n \rightarrow D$ (constant are 0-ary functions):
$F = \{f_I : D^n \rightarrow D | f \in Terms\}$

state: $\sigma : Term \rightarrow D$ is a mapping from terms to the domain such that:

$$\sigma(f(t_1, \ldots, t_n)) = f_I(\sigma(t_1), \ldots, \sigma(t_n))$$

assignment of relations: for each n-ary predicate symbol in the atom language p there exists a relation $p_I \subseteq D^n$ (propositions are 0-ary functions):
$R = \{p_I : D^n | p \in Atoms\}$

Logical truth of logical expression:

- $I \models_\sigma E$:
  - $I \models_\sigma p(t_1, \ldots, t_n)$ iff $\langle \sigma(t_1), \ldots, \sigma(t_n) \rangle \in p_I$
  - $I \models_\sigma A_1, \ldots, A_n$ iff $I \models_\sigma A_i$ for all $i \in 1, \ldots, n$
  - $I \models_\sigma A \leftarrow B_1, \ldots, B_n$ iff $I \models_\sigma B_1, \ldots, B_n$ implies $I \models_\sigma A$
  - $I \models_\sigma A_1, \ldots, A_n \leftarrow B_1, \ldots, B_n$ iff $I \models_\sigma B_1, \ldots, B_n$ implies $I \models_\sigma A_1, \ldots, A_n$
- $I \models E$ iff for all states σ: $I \models_\sigma E$

- I is a model of a set of clauses (a program) S iff for all $E \in S$: $I \models E$

- T is a logical consequence of S (i.e. $S \models T$) iff for all I such that $I \models S$: $I \models T$ (e.g. every model of S is a model of T).

Satisfaction as inconsistency

- Let A be a query and P is a program (a set of clauses).

- $P \models \exists A$ iff there exists a grounding substitution θ such that $P \models A\theta$ iff $P \cup \{\neg A\theta\}$ is inconsistent (e.g. there exists no I such that $I \models P \cup \{\neg A\theta\}$).

- A program $P = \{C_1, \ldots, C_n\}$ can be considered as a universally quantified disjunction of clauses

$$P' = \forall X_1, \ldots, X_m : C_1 \vee \cdots \vee C_n$$

where each
$C_i = H_i \leftarrow B_1^i, \ldots, B_{k_i}^i = \neg B_1^i \wedge \ldots \wedge \neg B_{k_i}^i \wedge H_i$ and $\{X_1, \ldots, X_m\} = Var(P)$.

- P' can be skolemized and put in conjunctive normal form obtaining: $P' = D_1 \wedge \ldots \wedge D_l$ where each $D_j = A_1^j \vee \cdots \vee A_{k_j}^j$.

- **Herbrand method:** Let $P' = \{D_1, \ldots, D_l\}$. $P \models \exists A$ iff $P' \cup \{\neg A\theta\}$ is inconsistent.

- We consider special interpretations where terms and predicate symbols are interpreted on themselves (e.g. $D = Term \cup Atoms$). These interpretations are called *Herbrand* interpetations.

- $P' \cup \{\neg A\theta\}$ is inconsistent iff $P' \cup \{\neg A\theta\}$ is false under all Herbrand interpretations (e.g. it has no Herbrand models).

Herbrand Interpretations

Herbrand universe: $HU_{\mathcal{L}}$ is the set of all ground terms of language $\mathcal{L}$

Herbrand base: $HB_{\mathcal{L}}$ is the set of all ground atoms of language $\mathcal{L}$

Observe that given a logic program P it is possible to define the language $\mathcal{L}(P)$of terms and atoms generated by P. We write HU_P and HB_P instead of $HU_{\mathcal{L}(P)}$ and $HB_{\mathcal{L}(P)}$.

Herbrand interpretation: is an interpretation $H_{\mathcal{L}} = \langle HU_{\mathcal{L}}, F, R, \sigma \rangle$ where:

- $F = \{f_H : HU_{\mathcal{L}}^n \rightarrow HU_{\mathcal{L}} | f \in HU_{\mathcal{L}}\}$

- $\sigma : HU_{\mathcal{L}} \rightarrow HU_{\mathcal{L}}$ is a mapping from over the Herbrand universe such that:

$$\sigma(f_H(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$$

- $R = \{r | r \subseteq HB_{\mathcal{L}}^n \wedge n \geq 0\}$

It is easy to see that a Herbrand interpretation is univocally determined by the set R of atoms interpretations. With abuse of notation we identify H with R. It is also apparent that if A is a ground atom then $H \models A$ iif $A \in H$ and that $H \models P$ iff for every clause $H \leftarrow B_1, \ldots, B_n \in ground(P)$, $I \models B_1, \ldots, B_n$ implies $I \models H$ where:
$ground(P) = \{C\theta \,|\, C \in P \wedge \theta\, is\, grounding\}$.

The Least Herbrand Model

Least Herbrand model: H is the least Herbrand model of a set of clauses S iff H is an Herbrand model and $H \subseteq H'$ for all Herbrand models H' of S iff

$$H = \bigcap_{H_i \models S} H_i.$$

Theorem. *Let P be a logic program and let M_P be the least Herbrand model of P then:*

$$M_P = \{A \in HB_P \,|\, P \models A\}$$

Proof. Let $A \in HB_P$:
$P \models A$
iff {by the deduction theorem}
$P \cup \{\neg A\}$ is unsatisfiable
iff
$P \cup \{\neg A\}$ has no Herbrand models
iff
$\neg A$ is false in all Herbrand models of P
iff
A is true in all Herbrand models of P
iff
$A \in M_P$ (A is true in the least Herbrand model of P). $\square$

Fix-point Semantics

Immediate consequence operator [Kowalski76]

Let P a positive logic program and I a Herbrand interpretation. Then

$$
\begin{aligned}
T_P(I) &= \{H \mid H \leftarrow B_1, \ldots, B_n \in ground(P) \wedge \\
&\qquad I \models B_1 \wedge \ldots \wedge B_n\} \\
&= \{H \mid H \leftarrow B_1, \ldots, B_n \in ground(P) \wedge \\
&\qquad \{B_1, \ldots, B_n\} \subseteq I\}
\end{aligned}
$$

Lemma. *A Herbrand interpretation I is a model of a positive logic program P iff $T_P(I) \subseteq I$ (e.g. I is a pre-fix point of T_P.*

Proof. $I \models P$ iff {definition of models of a set of clauses} for every clauses $H \leftarrow B_1, \ldots, B_n \in ground(P)$, $I \models B_1, \ldots, B_n$ implies $H \in I$ iff {definition of T_P} $T_P(I) \subseteq I$

□

Lemma. *T_P is a monotonic and continuous operator over the cpo $\langle HB_P, \subseteq \rangle$.*

Proof. Monotonicity follows by the definition. For the continuity consider a sequence $I_0 \subseteq I_1 \subseteq \ldots$ of Herbrand interpretations and suppose that $H = T_P(\bigcup_{n \in \omega} I_n)$. This means that there exists $\{B_1, \ldots, B_k\}$ such that the clause $H \leftarrow B_1, \ldots, B_k \in ground(P)$ and that $\bigcup_{n \in \omega} I_n \models B_1, \ldots, B_k$ iff $\{B_1, \ldots, B_k\} \subseteq \bigcup_{n \in \omega} I_n$. This implies that there exists a m such that $\{B_1, \ldots, B_k\} \subseteq I_m$ and hence $H \in T_P(I_m) \subseteq \bigcup_{n \in \omega} T_P(I_n)$. This condition is equivalent to continuity.

□

Theorem. *[Kleene] There exists the least fix point $T_P^\omega = \bigcup_{n=0}^{\infty} T_P^n(\emptyset)$ of T_P which is also the least pre-fix point that is $T_P(T_P^\omega) \subseteq T_P^\omega$.*

Corollary. *T_P^ω is the least Herbrand model of P.*

The Resolution Principle

SLD refutation: Let P a positive logic program and $G \equiv \leftarrow A_1, \ldots, A_n$ a ground goal. A SLD-refutation for $P \cup \{G\}$ is a proof of $\neg G \equiv A_1, \ldots, A_n$ obtained by applying a finite number of times the following inference rules:

$$\frac{}{P \vdash empty}$$

$$\frac{P \vdash G_1 \wedge P \vdash G_2}{P \vdash G_1, G_2}$$

$$\frac{P \vdash (A \leftarrow G) \wedge P \vdash G}{P \vdash A}$$

$$\frac{(A \leftarrow G) \in ground(P)}{P \vdash (A \leftarrow G)}$$

Success Set The set of groud atoms A such that there exists a SLD-refutation for $P \cup \{A\}$, is called $success\ set$ of P and it is defined by:

$$
\begin{aligned}
SS(P) &= \{A \mid A : ground \wedge \\
&\qquad P \vdash A\}.
\end{aligned}
$$

The three equivalent semantics of Logic Programming

Theorem. *[vanEmden e Kowalski]*

Let P a positive logic program. Then P has a Herbrand model M such that:

1. M is the $Least\ Herbrand\ Model$ of P.
2. M is the least fix point of T_P and $M = T_P^\omega$
3. $M = SS(P)$

Non-ground SLD-refutation

SLD stands for "Linear resolution for Definite Clauses with Selection function".

Let P be a logic program and $G = \leftarrow G_1, \ldots, G_n$ a goal. Then a SLD-resolution step for $P \cup \{G\}$ is given by the application of the following rule:

$$\frac{(A \leftarrow B_1, \ldots, B_m) \in P \wedge \exists \vartheta = mgu(A, G_k)}{P \vdash \leftarrow (G_1, \ldots, G_{k-1}, B_1, \ldots, B_m, G_{k+1}, \ldots, G_n)\vartheta}$$

- The choice of the k-th atom in the goal to be resolved is made by a given selection rule R. With $mgu(A, A')$ we denote the most general unifier for A and A'.
- A substitution ϑ is correct for G in P if $P \models \forall \overline{X}(G)\vartheta$, where $\overline{X} \in FV(G)$.
- A SLD derivation via R from $P \cup \{G\}$ is a finite sequence of goals starting with G where each G_{i+1} is obtained from G_i by application of the SLD-resolution rule via R using (C_i, G_i) with C_i variant of a clause in P and ϑ_i is the mgu computed.
- A SLD refutation of $P \cup \{G\}$ via R is a finite derivation of $P \cup \{G\}$ where the last goal is the empty clause (succesful derivation).

Semantic results

- The $\vartheta = \vartheta_1 \ldots \vartheta n$ substitution obtained composing all the substitutions computed at each resolution step is called *computed answer substitution* for the goal G.

- Let be σ and σ' two computed answer substitutions obtained by two SLD-refutation of $P \cup \{G\}$ via two different selection rules R and R' then $G\sigma$ is a variant of $G\sigma'$.

- For each correct substitution ϑ for G in P there exists a SLD refutation for $P \cup \{G\}$ via R which computes an answer substitution σ such that $\sigma = \vartheta\gamma$ for some γ.

- The same result of the above theorem are proved introducing the *S-semantics* as a generalization of the Least Herbrand Model semantics.