Declarative Programming

Functional Programming
  LISP
    Scheme
    Common Lisp
    ML
    Haskell

Logic Programming
  PROLOG

based on the λ-calculus
based on the first-order predicate calculus
Logic Programming

• based on the First-Order Predicate Calculus
• computing with relations and queries
• backtracking is the main paradigm for finding answers.
• input-output relation (often) blurred
There may be infinitely many

Relations are defined using clauses

direc(dran, 518-442-3387).
direc(berg, 518-442-4267).
direc(ravi, 518-442-4278).
eve% which prolog
/usr/local/bin/prolog
eve% prolog
booting SICStus...please wait
SICStus 2.1 #8: Mon Sep 27 16:20:51 EDT 1993
| ?- [user].
| direc(dran, 518-442-3387).
| direc(berg, 518-442-4267).
| direc(ravi, 518-442-4278).
| user consulted, 10 msec 608 bytes

yes
| ?- direc(dran, X).

X = 518-442-3387 ? ;

no
| ?- direc(X, 518-442-4267).

X = berg ? ;

no
Another example

direct(albany, pittsburg).
direct(albany, chicago).
direct(albany, philadelphia).
direct(chicago, seattle).
direct(chicago, philadelphia).
direct(chicago, los_angeles).
direct(seattle, los_angeles).
direct(memphis, peoria).
fly(X,Y) :- direct(X,Y).
fly(X,Y) :- direct(X,Z), fly(Z,Y).

direct and fly are predicates.

:- stands for “if”

reverse implication (←)
First-Order Predicate Calculus

- quantifiers $\forall$, $\exists$

$$\forall X : p(X) \equiv \neg (\exists X : \neg(p(X)))$$

- variables

- predicate and function symbols
  - every symbol has an *arity*
  - functions of arity 0 are *constants*
  - predicates of arity 0 are *propositions*

- boolean operators
Terms, Atoms

- Every variable is a term

- if $f$ is a function symbol of arity $n$ and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term

- if $p$ is a predicate symbol of arity $m$ and $s_1, \ldots, s_m$ are terms, then $p(s_1, \ldots, s_m)$ is an atom

$Var(t)$: the set of variables occurring in a term (atom) $t$

$t$ is a **ground** term iff $Var(t) = \emptyset$
Definite clauses

Definite clauses are either \textit{facts} or \textit{rules}

\[
\text{<fact>} ::= \text{<atom>}. \\
\text{<rule>} ::= \text{<atom>} :- \text{<atoms>}. \\
\text{<atoms>} ::= \text{<atom>} \mid \text{<atom>} , \text{<atoms>}
\]

Variables in a definite clause are implicitly universally quantified.

\[
\text{<atom>} ::= \text{<atoms>}
\]

\textit{Head} of the rule

\textit{Body}
Queries

?- <atoms>.

Variables in a query are implicitly existentially quantified.

Atoms in a query are often called goals.
Example

Natural numbers represented by

\[ 0, s(0), \ldots, s^i(0), \ldots \]

(1) \( \text{add}(0, Y, Y) \).

(2) \( \text{add}(s(X), Y, s(Z)) \) :- \( \text{add}(X, Y, Z) \).

(1) means \( \forall Y : \text{add}(0, Y, Y) \)

(2) means

\[ \forall X \forall Y \forall Z : \text{add}(X, Y, Z) \rightarrow \text{add}(s(X), Y, s(Z)) \]

(2) is equivalent to

(2’) \( \text{add}(s(U), V, s(W)) \) :- \( \text{add}(U, V, W) \).
Substitutions

mappings from variables to terms

\[ \theta = [s(0)/X, s(s(0))/Y, U/Z] \]

replacements done simultaneously (in parallel)

\[ f(X, Y)\theta = f(s(0), s(s(0))) \]

— sometimes written as \( \theta(f(X, Y)) \)

A term (atom) \( s \) is an instance of a term (atom) \( t \) iff there is a substitution \( \sigma \) such that

\[ s = t\sigma \]

A term (atom) \( s \) is a variant of a term (atom) \( t \) iff there is a 1-1 substitution \( \eta \) from variables to variables such that

\[ s = t\eta \]
Composition of substitutions

\( \sigma \circ \theta \) — composition of substitutions \( \sigma \) and \( \theta \)

\[ \delta =_{V} \sigma \circ \theta \) (or \( \sigma \theta \)) \iff \]

\[ x\delta = (x\sigma)\theta \quad (\forall x \in V) \]

A substitution \( \theta \) is idempotent if and only if

\[ \theta = \theta\theta \]

\[ \sigma = [Y/X, X/Y] \]

is not idempotent since

\[ X\sigma\sigma = X \neq X\sigma \]
Restriction of a substitution

Let $V$ be a set of variables

$$\delta = \theta|_V \quad (\theta \text{ restricted to } V)$$

if and only if

$$\delta(x) = \begin{cases} \theta(x) & \text{if } x \in V \\ x & \text{otherwise} \end{cases}$$
Unification

A substitution $\theta$ **unifies** terms $s$ and $t$ iff

$$s\theta = t\theta$$

$\theta$ is a **unifier** of $s$ and $t$.

**Examples:**

1. $s = f(X, s(X)), t = f(s(0), Y),$
   $$\theta = [s(0)/X, s(s(0))/Y]$$

2. $s = f(X, X), t = f(s(0), Y),$
   $$\theta = [s(0)/X, s(0)/Y]$$

3. $s = f(X, X), t = f(s(W), Y),$
   $$\theta = [0/W, s(0)/X, s(0)/Y]$$
   $$[U/W, s(U)/X, s(U)/Y]$$ is also a unifier.
   It is also **more general** than $\theta$. 

_CSI 311_
Non-unifiability

$f(X, Y)$ and $s(0)$ are not unifiable because $f$ and $s$ (the “root symbols”) are different (function clash)

$X$ and $f(X, Y)$ are not unifiable, since $X$ "occurs in" $f(X, Y)$: so no matter what one substitutes for $X$, $f(X, Y)$ will properly contain it (occur-check failure)

More examples:

1. $s = f(X, X), t = f(s(Y), Y)$
2. $s = f(X, X), t = f(s(0), f(0, Y))$
3. $s = add(0, Y, Y), t = add(X, X, s(s(0)))$
Unification problem

Input: A set of equations over terms

\[ S = \{ s_1 =? t_1, \ldots, s_k =? t_k \} \]

Output: A most general unifier (mgu) \( \theta \) for \( S \) if \( S \) is unifiable; otherwise, output “Not Unifiable”

In other words, \( \theta \) should be most general simultaneous unifier for all the equations in \( S \).
Unification algorithm

Given in terms of steps

- each step considers one equation from the set
- steps performed in any order
- until finished: i.e., until no more steps can be applied
- results merged back into the set after each step

“x occurs in t”: $x \neq t$ and $x \not\in Var(t)$
Terms as trees

\[ f(X, f(f(X, a), f(a, b))) \]
SLD-resolution

?- G₁, G₂, ..., Gₖ

H' :- B₁', ..., Bₘ'  \[ \beta = \text{mgu}(G₁, H') \]

?- (B₁', ..., Bₘ', G₂, ..., Gₖ) \[ \beta \]

variant of H :- B₁, ..., Bₘ

with fresh new variables
SLD-derivation

a finite sequence of SLD-resolution steps

Let \( Q \) be the original query

\[
Q_0 = Q \Rightarrow_{c_1} Q_1 \Rightarrow_{c_2} \ldots \Rightarrow_{c_n} Q_n
\]

The derivation is successful iff it ends with the empty clause (i.e., if \( Q_n = \square \))

The answer substitution is

\[
(\theta_1 \theta_2 \ldots \theta_n)|_{Var(Q)}
\]
Variants with new variables each time

\[
\begin{align*}
Q_0 &= Q \\
Q_1 &= Q_0 \circ c_1 \theta_1 \\
&\vdots \\
Q_{n-1} &= Q_0 \circ c_{n-1} \theta_{n-1} \circ c_n \theta_n
\end{align*}
\]
\[
\text{add}(X, s(0), s(s(0)))
\]

\[
\text{add}(s(U^1), V^1, s(W^1)) :\text{[}s(U^1)/X, s(0)/V^1, s(0)/W^1\text{]}
\]

\[
\text{add}(U^1, V^1, W^1)
\]

\[
\text{add}(U^1, s(0), s(0))
\]

\[
\text{add}(0, Y^2, Y^2) :\text{[}0/U^1, s(0)/Y^2\text{]}
\]

\[
\text{add}(0, Y^2, Y^2)
\]

CSI 311
\[
\text{add}(U^1, s(0), s(0))
\]

\[
\text{add}(s(U^2), V^2, s(W^2)) \leftarrow [s(U^2)/U^1, s(0)/V^2, 0/W^2]
\]

\[
\text{add}(U^2, s(0), 0)
\]

**FAILURE**
\[
\text{add}(X, X, s(s(0)))
\]

\[
\text{add}(s(U), V, s(W)) \quad \text{:- add}(U, V, W).
\]

\[
\text{add}(0, Y_1, Y_1).
\]

\[
\text{add}(U, s(U), s(0))
\]

\[
2 \quad [s(U)/X, s(U)/V, s(0)/W]
\]

\[
1 \quad [0/U, s(0)/Y_1]
\]
\begin{align*}
\text{add}(X, X, Y), \text{add}(Y, Z, s(s(0)))
\end{align*}

\begin{align*}
1 & \rightarrow [0/X, 0/Y, 0/Y'] \\
\text{add}(0, Z, s(s(0)))
\end{align*}

\begin{align*}
1 & \rightarrow [s(s(0))/Z, s(s(0))/Y’’] \\
\square
\end{align*}
\[\text{add}(X, X, Y), \text{add}(Y, Z, s(s(0)))\]

\[\text{add}(U, s(U), W), \text{add}(s(W), Z, s(s(0)))\]

\[\text{add}(s(s(0)), Z, s(s(0)))\]

\[\text{add}(s(0), Z, s(0))\]
SLD-derivation tree

?- \( G_1, G_2, \ldots, G_n \)
add(X, X, Y), add(Y, Z, s(s(0)))

1 [0/X, 0/Y, 0/Y'']
add(0, Z, s(s(0)))

2 [s(U)/X, s(W)/Y, s(U)/V]
add(U, s(U), W), add(s(W), Z, s(s(0)))

1 [s(s(0))/Z, s(s(0))/Y'']

add(0, Z, s(s(0)))

2 [s(0)/U, Z/V, s(0)/W]
add(s(s(0)), Z, s(s(0)))

1 [0/U, s(0)/W, s(0)/Y']
add(s(s(0)), Z, s(s(0)))

2 [s(0)/U, Z/V, s(0)/W]
Infinite derivations

Infinite (unsuccessful) derivations are possible:

\[
\begin{align*}
\text{add}(X, s(0), X) \\
\text{add}(U_1, s(0), U_1) \\
\text{add}(U_2, s(0), U_2)
\end{align*}
\]

2 \quad \frac{[s(U_1)/X, s(0)/V_1, U_1/W_1]}{[s(U_2)/U_1, s(0)/V_2, U_2/W_2]}

2