Declarative Programming

Functional Programming

LISP
  - Scheme
  - Common Lisp
  - ML
  - Haskell
  based on the \( \lambda \)-calculus

Logic Programming

PROLOG
  based on the first-order predicate calculus
Logic Programming

- Automated reasoning
- Relational databases

- 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, pittsburgh).
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 \textit{arity}
  - functions of arity 0 are \textit{constants}
  - predicates of arity 0 are \textit{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

\( \text{Var}(t) \): the set of variables occurring in a term (atom) \( t \)

\( t \) is a **ground** term iff \( \text{Var}(t) = \emptyset \)
Definite clauses

Definite clauses are either *facts* or *rules*

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

Variables in a definite clause are implicitly universally quantified.

![Diagram of a definite clause showing head and body]

*Head* of the rule

*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 \( \theta(f(X, Y)) \)
Instance, Variant

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

\[ \sigma \circ \theta - \text{composition of substitutions } \sigma \text{ and } \theta \]

\[ \delta \equiv_V \sigma \circ \theta \text{ (or } \sigma \theta) \text{ iff } \]

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

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:

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

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

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

$[U/W, s(U)/X, s(U)/Y]$ is also a unifier. It is also more general than $\theta$. 
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)

**Examples:**

- $s = f(X, X), t = f(s(Y), Y)$
- $s = f(X, X), t = f(s(0), f(0, Y))$
- $s = add(0, Y, Y), t = add(X, X, s(s(0)))$
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 \( x \not\in \text{Var}(t) \)
Terms as trees

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

?- \( G_1, G_2, ..., G_k \)

\( H' \leftarrow B'_1, ..., B'_m \) \[ \beta = \text{mgu}(G_1, H') \]

?- \( (B'_1, ..., B'_m, G_2, ..., G_k) \beta \)

variant of \( H \leftarrow B_1, ..., B_m \)

with fresh new variables
?- $G_1, \ldots, G_m$

$H' : - B'_1, \ldots, B'_k$

$\theta = \text{mgu}(G_1, H')$

?- $(B'_1, \ldots, B'_k, G_2, \ldots, G_m)\theta$

variant of $H : - B_1, \ldots, B_k$

with fresh new variables
SLD-derivation

a finite sequence of SLD-resolution steps

Let $Q$ be the original query

$$ Q_0 = Q \Rightarrow_{c_1}^\theta Q_1 \Rightarrow_{c_2}^\theta \ldots \Rightarrow_{c_n}^\theta Q_n $$

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

The answer substitution is

$$ (\theta_1 \theta_2 \ldots \theta_n)|_{Var(Q)} $$
\[ Q_0 = Q \]

\[ Q_1 \]

\[ Q_{n-1} \]

\[ Q_n \]

Variants of program clauses with new variables each time
The answer substitution is \([s(0)/X]\)
add($U^1$, $s(0)$, $s(0)$)

$$add(s(U^2), V^2, s(W^2)) :\quad add(U^2, V^2, W^2)$$

$$\rightarrow [s(U^2)/U^1, s(0)/V^2, 0/W^2]$$

add($U^2$, $s(0)$, 0)

FAILURE
The answer substitution is \([s(0)/X]\)
add(X, X, Y), add(Y, Z, s(s(0)))

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

The answer is \([0/X, \ 0/Y, \ s(s(0))/Z]\)
The answer is \[ s(0)/X, s(s(0))/Y, 0/Z \]
SLD-derivation tree

?- $G_1, G_2, \ldots, G_n$

\[ \ldots \]

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

1. \[\left[\frac{0}{X}, \frac{0}{Y}, \frac{0}{Y'}\right]\]
2. \[\left[\frac{s(U_1)}{X}, \frac{s(W_1)}{Y}, \frac{s(U_1)}{V_1}\right]\]

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

1. \[\left[\frac{s(s(0))}{Z}, \frac{s(s(0))}{Y''}\right]\]
2. \[\left[\frac{s(0)}{U_2}, \frac{Z}{V_2}, \frac{s(0)}{W_2}\right]\]

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

1. \[\left[\frac{0}{U_1}, \frac{s(0)}{W_1}, \frac{s(0)}{Y_1}\right]\]

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

1. \[\left[\frac{s(0)}{U_2}, \frac{Z}{V_2}, \frac{s(0)}{W_2}\right]\]

\text{add}(s(0), Z, s(0))
Infinite derivations

Infinite (unsuccessful) derivations are possible:

\[ \text{add}(X, s(0), X) \]
\[ 2 \rightarrow [s(U_1)/X, s(0)/V_1, U_1/W_1] \]
\[ \text{add}(U_1, s(0), U_1) \]
\[ 2 \rightarrow [s(U_2)/U_1, s(0)/V_2, U_2/W_2] \]
\[ \text{add}(U_2, s(0), U_2) \]
\[ 2 \]