Prolog

Alan Mao

College of Arts and Science

CSCI-GA 3110-001, Honors Programming Languages, Fall 2017
Logic Programming
<table>
<thead>
<tr>
<th>Programming languages</th>
<th>Formal language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional programming</td>
<td>(Typed) λ-calculus</td>
</tr>
<tr>
<td>Logic programming</td>
<td>First-order predicate calculus</td>
</tr>
</tbody>
</table>
Programming languages  Formal language

Functional programming
<table>
<thead>
<tr>
<th>Programming languages</th>
<th>Formal language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional programming</td>
<td>(Typed) $\lambda$-calculus</td>
</tr>
<tr>
<td>Programming languages</td>
<td>Formal language</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>Functional programming</td>
<td>(Typed) $\lambda$-calculus</td>
</tr>
<tr>
<td>Logic programming</td>
<td></td>
</tr>
</tbody>
</table>
Programming languages
  Functional programming
  Logic programming

Formal language
  (Typed) λ-calculus
  First-order predicate calculus
Basic Concepts
Axioms
Axioms

Goals
Axioms

Goals

Horn logic
Axioms

Goals

Horn logic

- resolution
Axioms

Horn clause
Axioms

Horn clause

- head or consequent $H$
- body $\{B_i\}_{i \in \mathbb{N} < n}$
Axioms

Horn clause

- head or consequent \( H \)
- body \( \{B_i\}_{i \in \mathbb{N}_0} \)

\[ H \leftarrow B_0, B_1, \ldots, B_{n-1} \]
A term is a constant, a variable, or a function whose arguments are themselves terms: $a, x, f(x)$, or $h(c, f(z), y)$
A term is a constant, a variable, or a function whose arguments are themselves terms: $a$, $x$, $f(x)$, or $h(c, f(z), y)$

A literal is either an atomic formula or the negation of an atomic formula: $F(x)$ or $\neg R(x, f(a))$
Language Representation

Two literals are complementary if one is the negation of the other

A clause is a finite disjunction of literals

Ground terms, ground literals, and ground clauses have no variables
Example

\[ C \leftarrow A, B \]
\[ D \leftarrow C \]
\[ D \leftarrow A, B \]
Example

\[ C \leftarrow A, B \]
\[ D \leftarrow C \]

\[ D \leftarrow A, B \]
Resolution

Unification
Resolution

Unification

cold(X) ← snowy(X)
snowy(New York)

__________
cold(New York)
Prolog
Functional programming languages    Prolog
Functional programming languages  Prolog

Referencing environment
Functional programming languages

Referencing environment

Prolog

Database
Functional programming languages

Referencing environment
Functions, constants, etc.

Prolog

Database
Functional programming languages       Prolog

Referencing environment   Database
Functions, constants, etc.   Horn clauses
Terms

Constants

Variables

Structures
Constants

Atoms

Numbers
Atoms

\{a, \ldots, z\}{a, \ldots, z, A, \ldots, Z, 1, \ldots, 0, _\}^*

\{#, \&, *, +, -, ., /, :, <, =, >, ?, @, ^, ~, \ldots\}^*

"\{c : c \text{ is any character}\}^*"
Variables

\{A, \ldots, Z\}\{a, \ldots, z, A, \ldots, Z, 1, \ldots, 0, _\}^*
Variables

Instantiated at run time

Scope limited to clause

No declarations

Dynamic type checking
Structures

Functor: an atom

List of arguments: constants, variables, or structures
Structures

Functor: an atom

List of arguments: constants, variables, or structures

snowy(new_york)
takes(alan, csci3110)
bin_tree(foo, bin_tree(bar, baz))
Clauses

Facts

Rules

Queries
Clauses

Facts

Rules

Queries

Ending with periods
Facts

Heads of Horn clauses

snowy(new_york).
Facts

Heads of Horn clauses

snowy(new_york).
Rules

Horn clauses

snowy(X) :- rainy(X), freezing(X).

Universal quantification for variables in the head
Rules

Horn clauses

\[ \text{snowy}(X) :\!-\! \text{rainy}(X), \text{freezing}(X). \]
Rules

Horn clauses

\( \text{snowy}(X) :\!-\! \text{rainy}(X), \text{freezing}(X). \)

Universal quantification for variables in the head
Queries

Horn clauses with empty heads

To be answered by the interpreter
Example

snowy(manhattan).
snowy(brooklyn).
?- snowy(B).
Example

Interpreting result

B = manhattan
Example

Interpreting result

B = manhattan ;
B = brooklyn.
Resolution
Example

takes(a, math_101).
takes(a, csci_101).
takes(b, phil_101).
takes(b, csci_101).
classmates(X, Y) :- takes(X, Z), takes(Y, Z).
Example

takes(a, math_101).
takes(a, csci_101).
takes(b, phil_101).
takes(b, csci_101).
classmates(X, Y) :- takes(X, Z), takes(Y, Z).

Existential quantification for variables in the body
Example

classmates(a, Y) :- takes(Y, csci_101).
Example

classmates(a, Y) :- takes(Y, csci_101).

Unification and instantiation
Unification Rules

- A constant unifies only with itself.
Unification Rules

- A constant unifies only with itself.
- Two structures unify if and only if they have the same functor and the same arity and the corresponding arguments unify recursively.
Unification Rules

- A constant unifies only with itself.
- Two structures unify if and only if they have the same functor and the same arity and the corresponding arguments unify recursively.
- A variable unifies with anything.
Unification Rules

- A constant unifies only with itself.
- Two structures unify if and only if they have the same functor and the same arity and the corresponding arguments unify recursively.
- A variable unifies with anything.
  - If the other thing has a value, then the variable is instantiated.
  - If the other thing is an uninstantiated variable, then the two variables are associated in such a way that, if either is instantiated later, the value will be shared by both.
Equality

Unifiability
Equality

Unifiability

$= (A, B)$ or $A = B$ if and only if $A$ and $B$ are unifiable.
Example

?- a = a.
true. % constant unifies with itself
?- a = b.
false. % but not with another constant
?- foo(a, b) = foo(a, b).
true. % structures are recursively identical
?- X = a.
X = a. % variable unifies with constant
?- foo(a, b) = foo(X, b).
X = a. % arguments must unify
Example

?- A = B.
A = B. % unification without instantiation
?- A = B, A = a, B = Y.
A = B, B = Y, Y = a.
Limitations
Horn clauses do not capture all of first-order predicate calculus
The End