Principles of Programming Languages
Lecture 3

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgment: parts of these slides are based on material by Diane Horton & Eric Joanis @ UoT

References: Scheme by Dybvig
Concepts of PL by Sebesta
Prog. in Prolog by Clocksin and Mellish
PL Concepts and Constructs by Sethi
ML for the Working Prog. By Paulson
PL Pragmatics by Scott

Today

- **Logic Programming**

- **Prolog I**

LP: resolution in predicate calculus

- **We would like to infer new propositions (e.g. facts) from some existing set of propositions.**

- **An inference rule that can be applied atomically is called a *resolution***
  - E.g.

    Given: P1 ← P2 , Q1 ← Q2
    P1 ≡ Q2
    Alternatively: T ← P2 , Q1 ← T
    New fact: Q1 ← P2
    New Set of Rules: P1 ← P2 , Q1 ← Q2 , Q1 ← P2

- **Resolution gets more complex if variables/values are involved:**
  - To use resolution with variables, we will need to find values for variables that allow matching to proceed.
  - E.g.

    Given: F(X,Y) ← P2(Y,X)
    Q1(foo) ← F(foo, bar)
    New fact: Q1(foo) ← P2(bar, foo)
    New Set of Rules: F(X,Y) ← P2(Y,X) , Q1(foo) ← F(foo, bar) ,
    Q1(foo) ← P2(bar, foo)

LP: horn clause

- **Logic programming is heavily based on horn clauses:**

$$c \leftarrow h_1 \wedge h_2 \wedge h_3 \wedge \ldots \wedge h_n$$

  - Antecedents (h's): conjunction of zero or more conditions which are atomic constructs in predicate logic.
  - Consequent(c): an <u>atomic</u> construct in predicate logic

- **Meaning of a horn-clause:**
  - The consequent is true if the antecedents are all true
  - c is true if h_1, h_2, h_3,.... are all true

- **A horn clause can capture most, but not all, logical statements/implications, why?**

- **Additional reading: http://en.wikipedia.org/wiki/Horn_clause**

LP: horn clause made easy!

- **Horn clause can include more complex terms:**

p(X) ← q(X,Y) Λ r(X,Y) Λ s(X,Y)
p(X) ← K(M) Λ i(T)
q(X,Y) ← ……
r(X,Y) ← ……
s(X,Y) ← ……
k(M) ← ……
i(T) ← ……
…..

- **We can assume the following when writing horn-clauses:**
  - p is the program name
  - q,r,s are the subprogram names
  - X is a parameter of the program
  - Y is a local variable

LP: horn clause

$$c \leftarrow h_1 \wedge h_2 \wedge h_3 \wedge \ldots \wedge h_n$$

- Antecedents (h's): conjunction of zero or more conditions which are atomic constructs in predicate logic.
- Consequent(c): an atomic construct in predicate logic

- **Examples of horn clauses:**
  - Father (X, Y) 🡐 Child (Y, X) Λ Male (X).
  - Student(X) 🡐 Undergraduate(X).

- **Example of Non-horn clauses:**
  - Student(X) 🡐 Undergraduate(X) V Graduate(X)
  - ¬ (Student(X)) 🡐 Deregistered(X).

LP: specifying non-horn rules

- **Many non-horn rules can be transformed to horn form using one of two methods:**
  - logical equivalence
  - Skolemization

- **Logical equivalence:**
  - Uses the following logical laws:
    - Negation $\neg\neg A \equiv A$
    - De Morgan's Law $\neg(A \vee B) \equiv \neg A \wedge \neg B$
      $\neg(A \wedge B) \equiv \neg A \vee \neg B$
    - Distributive Property $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
      $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
    - Absorption Law $A \vee (A \wedge B) \equiv A$
      $A \wedge (A \vee B) \equiv A$
    - Implication Laws $A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$
      $A \leftarrow B \equiv A \vee \neg B$

LP: specifying non-horn rules

- **Logical equivalence rules:**
  - Negation $\neg\neg A \equiv A$
  - De Morgan's Law $\neg(A \vee B) \equiv \neg A \wedge \neg B$
    $\neg(A \wedge B) \equiv \neg A \vee \neg B$
  - Distributive Property $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
    $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
  - Absorption Law $A \vee (A \wedge B) \equiv A$
    $A \wedge (A \vee B) \equiv A$
  - Implication Laws $A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$
    $A \leftarrow B \equiv A \vee \neg B$

- **Examples:**
  - $\underline{\neg A \leftarrow \neg B} \equiv \neg A \vee \neg(\neg B)$
    $\equiv \neg A \vee B$
    $\equiv B \vee \neg A$
    $\equiv \underline{B \leftarrow A}$ (*horn-clause*)
  - $\underline{A \leftarrow (B \vee C)} \equiv A \vee \neg(B \vee C)$
    $\equiv A \vee (\neg B \wedge \neg C)$
    $\equiv (A \vee \neg B) \wedge (A \vee \neg C)$
    $\equiv (A \leftarrow B) \wedge (A \leftarrow C)$ (*horn-clauses*)

LP: specifying non-horn rules – cont'd

- **Logical equivalence rules:**
  - Negation: $\neg\neg A \equiv A$
  - De Morgan's Law: $\neg(A \vee B) \equiv \neg A \wedge \neg B$
    $\neg(A \wedge B) \equiv \neg A \vee \neg B$
  - Distributive Property: $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
    $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
  - Absorption Law: $A \vee (A \wedge B) \equiv A$
    $A \wedge (A \vee B) \equiv A$
  - Implication Laws: $A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$
    $A \leftarrow B \equiv A \vee \neg B$
- **Examples:**
  - $\underline{A \leftarrow (B \leftarrow C)} \equiv A \vee \neg(B \leftarrow C)$

$$\equiv A \vee \neg(B \vee \neg C)$$
$$\equiv A \vee (\neg B \wedge \neg\neg C)$$
$$\equiv A \vee (\neg B \wedge C)$$
$$\equiv (A \vee \neg B) \wedge (A \vee C)$$
$$\equiv (A \leftarrow B) \wedge (A \vee C)$$

(*non-horn*)

LP: specifying non-horn rules – cont'd

- **Skolemization:**
  - Non horn formulas like (∃X) A(X) can be converted to horn-clause by introducing a ***skolem constant*** and/or ***skolem function.*** The resulting clause is *almost* the same thing.

- **Why does skolemization works?**
  - We only need ∃X because we don't have have a name for X. By creating artificial names (*skolem names*), we can eliminate many ∃'s and convert many formulas to horn clause.

LP: specifying non-horn rules – cont'd

- **Horn clause** $c \leftarrow h_1 \wedge h_2 \wedge h_3 \wedge \ldots \wedge h_n$
  - What are we going to do about quantifiers? (∃X) and (∀X)

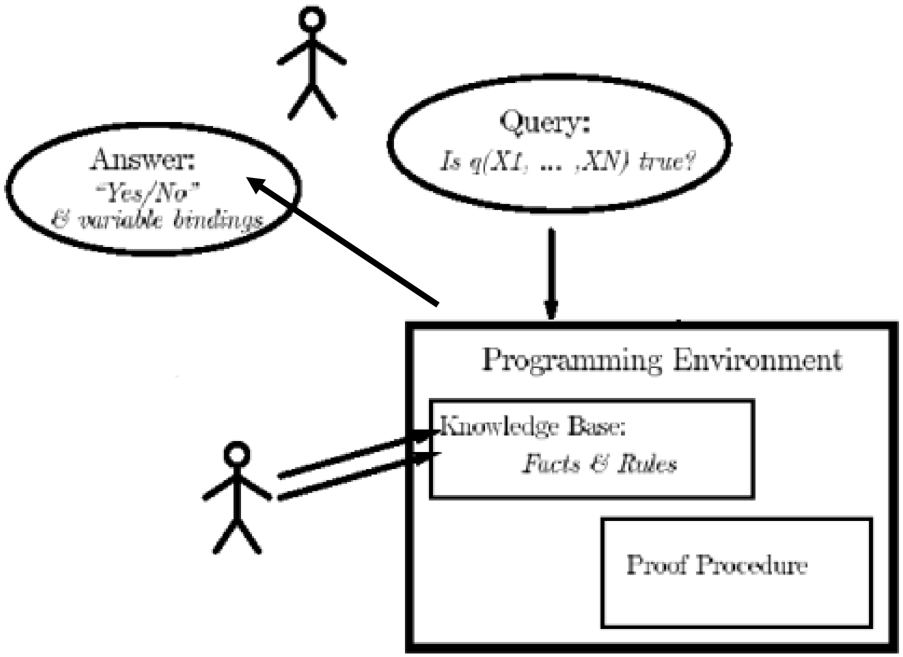
- **Skolemization:**
  - Variables bound by existential (∃X) quantifiers which are not inside the scope of universal quantifiers can simply be replaced by constants:
    - (∃X) mother(john,X) becomes mother(john,m)
  - When the existential quantifier (∃Y) is inside a universal quantifier (∀X) , the bound variable must be replaced by a *function* of the variables bound by universal quantifier (∀X) .
    - (∀X) [person(X) → (∃Y) mother(X,Y)]
      becomes
      (∀X) [person(X) → mother(X, m(X))]

Prolog I

Prolog:- Programmation en logique

- **The first and most popular logic programming language**
  - Invented by Alain Colmerauer and Phillipe Roussel at the University of Aix-Marseille in 1971 (France)

- **Characteristics:**
  - Is very weakly typed
  - Has no data abstraction
  - Has no functional abstraction!
  - Has no mutable state
  - Has no explicit control flow



- **So, how do you program?**
  - Load facts/rules into interpreter
  - Make queries to see if a fact is:
    - in the knowledge-base or
    - can be implied from existing facts or rules

- **Prolog is really an engine to *prove theorems***

Prolog: data types – quick intro

- **Simple**
  - Constants :
    - Numbers: integer, floating point,…
    - Atoms: alphabetic sequence starting with a lower case letter (e.g. apple)
  - Variables:
    - Variables start with capital letters or underscore
- **Complex**
  - Lists
  - Structures

Prolog: horn clauses

- **Recall** $c \leftarrow h_1 \wedge h_2 \wedge h_3 \wedge \ldots \wedge h_n$

- **Syntax:** <head> :- <body>.
  - You can conclude that <head> is true, if you can prove that <body> is true
  - The symbol :- is read as ***if***

- **3 types of clauses:**
  - Facts
  - Rules
  - Queries

Prolog: facts

- **A fact is a horn clause with an empty body (nothing to prove).**
- **Syntax**

  <head>.

- **What makes a fact a fact?**
- **Examples**
  - Exams — exams.
  - Assignments — assignments.
  - Taxes — taxes.
  - The earth is round. — round(earth).
  - The sky is blue. — blue(sky).
  - The sun is hot. — hot(sun).
  - Mary is a female. — female(mary).
  - Beethoven lived between 1770 & 1827. — person(beethoven,1770,1827).

Prolog: facts – cont'd

- **Facts about facts:**
  - Full stop "." at the end of every fact.
  - The number of arguments in a fact is called arity.
    - E.g. female(mary). is an instance of female/1 (functor female, arity 1)
  - Facts with different number of arguments are distinct
    - E.g. female(mary,may). is different from female(mary).

Prolog: rules

- **A rule in Prolog is a full horn clause:**

$$c \leftarrow h_1 \wedge h_2 \wedge h_3 \wedge \ldots \wedge h_n$$

- **Syntax:**

$$\underbrace{\texttt{rel}_1}_{\text{head}} \texttt{ :- } \underbrace{\texttt{rel}_2\texttt{, rel}_3\texttt{, ... rel}_n\texttt{.}}_{\text{body}}$$

  - *If I know that all those RHS relations (those in the body) hold, then I also know that this LHS relation (in the head) holds.*

- **Examples:**
  - If there is smoke there is fire

    fire :- smoke.

  - If the course is boring, I leave

    leave(i) :- boring(course).

  - Joe is going to kill the teacher if he fails CSC324.

    kills(joe, X) :- fails(joe,csc324), teaches(X,csc324).

Prolog: rules – cont'd

- **Examples:**
  - X is female if X is the mother of anyone.

    `female(X) :- mother(X,_).`

  - X is the sister of Y, if X is female and X's parents are M and F, and Y's parents are M and F

    `sister_of(X,Y):- female(X),parents(X,M,F),parents(Y,M,F).`

- **When to use rules?**
  - Use rules to say that a particular fact depends on a group of facts.
  - Use rules to deduce new facts from existing ones.

- **Rules of rules:**
  - The head of the rule consist of at most one predicate
  - The body of the rule is a finite sequence of literals separated by , or conjunction *(and)*
  - Rules always end with a period "."

Prolog: queries

- **A query is a clause with an empty head.**

$$\leftarrow h_1 \wedge h_2 \wedge h_3 \wedge \ldots \wedge h_n$$

- **Syntax**

|? <body>.

  - Try to prove that <body> is true
  - The goal is represented to the interpreter as a question.

- **Examples**

|?-round(earth). - is it true that the earth is round?
(or simpler than that: is the earth round?)

|?-round(X). - is it true that there are entities which are round?
(or simpler than that: what entities are round?)

Prolog: queries – cont’d

- **Examples**

|?-composer(beethoven,1770,1827).	- is it true that beethoven was a composer who lived between 1770 and 1827
|?-owns(john,book).	- is it true that john owns a book? (simpler: does john own a book?)
|?-owns(john,X).	- is it true that john owns something? (simpler: does john own something? or what does John own?)

Prolog: simple types - constants

- **There are two types of constants: *atoms* and *numbers*.**

- **Atoms:**
  - Alphanumeric atoms: *alphabetic sequence starting with a lower case letter*
    - E.g.: apple a1 apple_cart
  - Special atoms
    - E.g ! ; [ ] {} ,
  - Symbolic atoms
    - E.g. & < > * - + >>
  - Quoted atoms: *sequence of characters surrounded by single quotes*
    - Can make anything an atom by enclosing it in single quotes.
    - E.g 'Apple' 'hello world'

- **Numbers:**
  - Integers and Floating Point numbers
    - E.g. 0 1 9821 -10 1.3 -1.3E102

Prolog: simple types - variables

- **Variables start with <u>capital letters</u> or <u>underscore</u>**

- **There are *no global variables* (***assert and retract, will see them later…***)**

- **Instantiated vs. un-instantiated:**
  - if the object a variable stands for is already determined, var is ***instantiated***
  - if the object a variable stands for is not yet determined, var is ***un-instantiated***

- **An instantiated variable in Prolog cannot change its value**

- **Variables are limited in scope to the clause they appear in** *(local vars)*
  - E.g.

```
grandParent(X,Z) :- parent(X,Y), parent(Y,Z).    % The Xs here are the same var
sister_of(X,Y):- female(X),parents(X,M,F),parents(Y,M,F).  % But not the same
                                                            % as those here
```

- **There is a special anonymous variable "_" which is used to denote "don't care"**
  - E.g

```
Parent(X)   :- mother(X,_).
married(X) :- husband(X,_).
```

  - Note that every use of _ is considered a separate variable

Prolog: example 1

Facts

```
likes(eve, pie).      food(pie).
likes(al, eve).       food(apple).
likes(eve, tom).      person(tom).
likes(eve, eve).
```

query

```
?-likes(al, pie).
no
```

answer

```
?-likes(al,eve).
yes
?-likes(eve,al).
no
?-likes(person,food).
no
```

variable

```
?-likes(al,Who).
Who=eve
?-likes(eve,W).
W=pie   ;
W=tom   ;
W=eve   ;
no
```

answer with variable binding

force search for more answers