Logic Programming (PLP 11)
Predicate Calculus, Horn Clauses, Clocksin-Mellish Procedure

Carlos Varela
Rennselaer Polytechnic Institute

November 10, 2015

An Early (1971) "Conversation"

USER:

Cats kill mice.

Tom is a cat who does not like mice who eat cheese.

Jerry is a mouse who eats cheese.

Max is not a mouse.

What does Tom do?

COMPUTER:

Tom does not like mice who eat cheese.

Tom kills mice.

USER:

Who is a cat?

COMPUTER:

Tom.

USER:

What does Jerry eat?

COMPUTER:

Cheese.

USER:

Who does not like mice who eat cheese?

COMPUTER:

Tom.

USER:

What does Tom eat?

COMPUTER:

What cats who do not like mice who eat cheese eat.

Another Conversation

USER:

Every psychiatrist is a person.
Every person he analyzes is sick.
Jacques is a psychiatrist in Marseille.
Is Jacques a person?
Where is Jacques?
Is Jacques sick?

COMPUTER:

Yes.
In Marseille.
I don't know.

Logic programming

- A program is a collection of *axioms*, from which theorems can be proven.
- A *goal* states the theorem to be proved.
- A logic programming language implementation attempts to satisfy the goal given the axioms and built-in inference mechanism.

Propositional Logic

- Assigning truth values to logical propositions.
- Formula syntax:

f	::=	v	symbol
	|	$f \wedge f$	and
	|	$f \vee f$	or
	|	$f \Leftrightarrow f$	if and only if
	|	$f \Rightarrow f$	implies
	|	$\neg f$	not

Truth Values

- To assign a truth values to a propositional formula, we have to assign truth values to each of its atoms (symbols).
- Formula semantics:

a	b	a $\wedge$ b	a $\vee$ b	a $\Leftrightarrow$ b	a $\Rightarrow$ b	$\neg$ a
False	False	F	F	T	T	T
False	True	F	T	F	T	T
True	False	F	T	F	F	F
True	True	T	T	T	T	F

Tautologies

- A *tautology* is a formula, true for all possible assignments.
- For example: $\neg\neg p \Leftrightarrow p$
- The contrapositive law:

$$(p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$$

- De Morgan's law:

$$\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$$

First Order Predicate Calculus

- Adds variables, terms, and (first-order) quantification of variables.
- Predicate syntax:

a	::=	$p(v_1, v_2, \ldots, v_n)$	predicate
f	::=	a	atom
	|	$v = p(v_1, v_2, \ldots, v_n)$	equality
	|	$v_1 = v_2$	
	|	$f \wedge f$ | $f \vee f$ | $f \Leftrightarrow f$ | $f \Rightarrow f$ | $\neg f$	
	|	$\forall v.f$	universal quantifier
	|	$\exists v.f$	existential quantifier

Predicate Calculus

- In mathematical logic, a *predicate* is a function that maps constants or variables to true and false.
- Predicate calculus enables reasoning about propositions.
- For example:

$$\forall C\ [\mathrm{rainy}(C) \land \mathrm{cold}(C) \Rightarrow \mathrm{snowy}(C)]$$

Quantifier ($\forall$, $\exists$)

Operators ($\land, \lor, \Leftrightarrow, \Rightarrow, \neg$)

Quantifiers

- *Universal* (∀) quantifier indicates that the proposition is true for **all** variable values.
- *Existential* (∃) quantifier indicates that the proposition is true for **at least one** value of the variable.
- For example:

```
∀A ∀B [(∃C [ takes(A,C) ∧ takes(B,C)]) ⇒
                classmates(A,B) ]
```

Structural Congruence Laws

$P_1 \Rightarrow P_2 \equiv \neg P_1 \vee P_2$

$\neg \exists X\ [P(X)] \equiv \forall X\ [\neg P(X)]$
$\neg \forall X\ [P(X)] \equiv \exists X\ [\neg P(X)]$

$\neg (P_1 \wedge P_2) \equiv \neg P_1 \vee \neg P_2$
$\neg (P_1 \vee P_2) \equiv \neg P_1 \wedge \neg P_2$
$\neg \neg P \equiv P$

$(P_1 \Leftrightarrow P_2) \equiv (P_1 \Rightarrow P_2) \wedge (P_2 \Rightarrow P_1)$

$P_1 \vee (P_2 \wedge P_3) \equiv (P_1 \vee P_2) \wedge (P_1 \vee P_3)$
$P_1 \wedge (P_2 \vee P_3) \equiv (P_1 \wedge P_2) \vee (P_1 \wedge P_3)$

$P_1 \vee P_2 \equiv P_2 \vee P_1$

Clausal Form

- Looking for a *minimal kernel* appropriate for theorem proving.
- Propositions are transformed into normal form by using structural congruence relationship.
- One popular normal form candidate is *clausal form*.
- Clocksin and Melish (1994) introduce a 5-step procedure to convert first-order logic propositions into clausal form.

Clocksin and Melish Procedure

1. Eliminate implication (⇒) and equivalence (⇔).
2. Move negation (¬) inwards to individual terms.
3. *Skolemization*: eliminate existential quantifiers (∃).
4. Move universal quantifiers (∀) to top-level and make implicit , i.e., all variables are universally quantified.
5. Use distributive, associative and commutative rules of ∨, ∧, and ¬, to move into *conjuctive normal form*, i.e., a conjuction of disjunctions (or *clauses*.)

Example

```
∀A [¬student(A) ⇒ (¬dorm_resident(A) ∧
                    ¬∃B [takes(A,B) ∧ class(B)])]
```

1. Eliminate implication (⇒) and equivalence (⇔).

```
∀A [student(A) ∨ (¬dorm_resident(A) ∧
                  ¬∃B [takes(A,B) ∧ class(B)])]
```

2. Move negation (¬) inwards to individual terms.

```
∀A [student(A) ∨ (¬dorm_resident(A) ∧
                  ∀B [¬(takes(A,B) ∧ class(B))])]

∀A [student(A) ∨ (¬dorm_resident(A) ∧
                  ∀B [¬takes(A,B) ∨ ¬class(B)])]
```

Example Continued

```
∀A [student(A) ∨ (¬dorm_resident(A) ∧
                  ∀B [¬takes(A,B) ∨ ¬class(B)])]
```

3. *Skolemization*: eliminate existential quantifiers (∃).
4. Move universal quantifiers (∀) to top-level and make implicit , i.e., all variables are universally quantified.

```
student(A) ∨ (¬dorm_resident(A) ∧
             (¬takes(A,B) ∨ ¬class(B)))
```

5. Use distributive, associative and commutative rules of ∨, ∧, and ¬, to move into *conjuctive normal form*, i.e., a conjuction of disjunctions (or *clauses*.)

```
(student(A) ∨ ¬dorm_resident(A)) ∧
(student(A) ∨ ¬takes(A,B) ∨ ¬class(B))
```

Horn clauses

- A standard form for writing axioms, e.g.:

```
father(X,Y) ⇐ parent(X,Y), male(X).
```

- The Horn clause consists of:
  - A *head* or consequent term H, and
  - A *body* consisting of terms B_i

$$H \Leftarrow B_0 , B_1 , \ldots, B_n$$

- The semantics is:

« If B_0, B_1, …, and B_n, then H »

Clausal Form to Prolog

```
(student(A) ∨ ¬dorm_resident(A)) ∧
(student(A) ∨ ¬takes(A,B) ∨ ¬class(B))
```

6. Use commutativity of ∨ to move negated terms to the right of each clause.
7. Use $P_1 \vee \neg P_2 \equiv P_2 \Rightarrow P_1 \equiv P_1 \Leftarrow P_2$

```
(student(A) ⇐ dorm_resident(A)) ∧
(student(A) ⇐ ¬(¬takes(A,B) ∨ ¬class(B)))
```

8. Move Horn clauses to Prolog:

```
student(A) :- dorm_resident(A).
student(A) :- takes(A,B),class(B).
```

Skolemization

```
∃X [takes(X,cs101) ∧ class_year(X,2)]
```

introduce a Skolem constant to get rid of existential quantifier (∃):

```
takes(x,cs101) ∧ class_year(x,2)
```

```
∀X [¬dorm_resident(X) ∨
     ∃A [campus_address_of(X,A)]]
```

introduce a Skolem function to get rid of existential quantifier (∃):

```
∀X [¬dorm_resident(X) ∨
     campus_address_of(X,f(X)]
```

Limitations

- If more than one non-negated (positive) term in a clause, then it cannot be moved to a Horn clause (which restricts clauses to only one head term).
- If zero non-negated (positive) terms, the same problem arises (Prolog's inability to prove logical negations).
- For example:
  - « every living thing is an animal or a plant »

```
animal(X) ∨ plant(X) ∨ ¬living(X)
animal(X) ∨ plant(X) ⇐ living(X)
```

Exercises

72. What is the logical meaning of the second Skolemization example if we do not introduce a Skolem function?
73. Convert the following predicates into Conjunctive Normal Form, and if possible, into Horn clauses:
    a) `∀C [rainy(C) ∧ cold(C) ⇒ snowy(C)]`
    b) `∃C [¬snowy(C)]`
    c) `¬∃C [snowy(C)]`
74. PLP Exercise 11.5 (pg 571).
75. PLP Exercise 11.6 (pg 571).