1

Logical Foundations

How do we represent logical formulas in Prolog? Are all formulas representable? What does Prolog computation amount to?

Logical formulas (or, well formed formulas, or wffs) in First Order Predicate Calculus can be used to represent facts about the world:

"John is the father of George"

```
fatherOf(george, john)          or
father(john, george)
```

"Either Maria or Jill own a car"

```
∃X(own(maria, X) ∨ own(jill, X))
```

"Every dog is liked by someone"

```
∀X(dog(X) ⇒ ∃Y(likes(Y,X))
```

(or, every dog is liked by a single being:

```
∃Y∀X(dog(X) ⇒ likes(Y,X))        )
```


2

Horn Clause Derivation: Placing a wff into Clause Normal Form

An algorithm for placing an arbitrary formula into clause normal form

- Remove $\equiv$ and $\Rightarrow$ by using the following tautologies:

$\alpha \equiv \beta$ is the same as $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
$\alpha \Rightarrow \beta$ is the same as $\neg\alpha \vee \beta$

A formula is now left with $\vee$, $\wedge$, $\neg$ along with existential and universal quantifiers

3

• Move negation inward so it operates on an atomic proposition:

$\neg \neg \alpha$ is the same as α
$\neg (\alpha \vee \beta)$ is the same as $\neg \alpha \wedge \neg \beta$
$\neg (\alpha \wedge \beta)$ is the same as $\neg \alpha \vee \neg \beta$
$\neg (\exists X(\alpha(X)))$ is the same as $\forall X(\neg\alpha(X))$
$\neg (\forall X(\alpha(X)))$ is the same as $\exists X(\neg\alpha(X))$

For example,

$\neg (\forall X \exists Y (human(X) \wedge loves(X,Y)))$
$\exists X \neg (\exists Y (human(X) \wedge loves(X,Y)))$
$\exists X \forall Y (\neg (human(X) \wedge loves(X,Y)))$
$\exists X \forall Y (\neg human(X) \vee \neg loves(X,Y))$

4

• Remove existential variables

Existential variables introduced by $\exists$ can be replaced with **Skolem constants or functions.**

$\exists X((female(X) \wedge motherOf(X,john))$ (1)
'There is some X that is female and is John's mother'

$female(G1) \wedge motherOf(G1,john)$ (1')
'G1 is female and is John's mother'

What is the logical relationship between (1) and (1')? Are they logically equivalent?

$\forall Y(human(Y) \Rightarrow \exists X(motherOf(X,Y)))$
'For every Y, there is some X that is Y's mother'

$\forall Y(human(Y) \Rightarrow motherOf(g2(Y),Y))$
'For every Y, there is some g2(Y) that is Y's mother'

5

Horn Clause Derivation

• Remove universal quantifiers

Universal quantifiers can be removed by moving all to the very left of the formula, and then omitting them. This means that all variables in a formula are assumed to be universally quantified.

$\forall Y(human(Y) \Rightarrow motherOf(g2(Y),Y))$
'For every Y, there is some g2(Y) that is Y's mother'

$human(Y) \Rightarrow motherOf(g2(Y),Y)$
'For every Y, there is some g2(Y) that is Y's mother'

Some renaming of variables may be required for this:

$\forall X(human(X)) \vee \forall X(\neg human(X))$
$\forall X(human(X)) \vee \forall Y(\neg human(Y))$
$\forall X \forall Y(human(X) \vee \neg human(Y))$

(compare this with

$\forall X \forall X(human(X) \vee \neg human(X))$

or

$\forall X(human(X) \vee \neg human(X))$)

©1993 John Mylopoulos

6

Any quantifier-free formula in predicate calculus can be rewritten in conjunctive normal form, i.e., in the form

$conj_1 \wedge conj_2 \wedge ... \wedge conj_n$

where $conj_i = lit_{i1} \vee lit_{i2} \vee ... \vee lit_{im}$

and each literal lit_{ij} has the form $P(x)$ or $\neg P(x)$

• Place formula in conjunctive normal form

The following tautology is used for this step:

$\alpha \vee (\beta \wedge \gamma)$ is the same as $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

©1993 John Mylopoulos

7

What is the relationship between a formula α and the formula α' obtained from α by applying these steps?

α is unsatisfiable iff α' is

Since the conjunction is understood, we can just write the formula as a set of disjunctive clauses. For example:

$(p(X) \vee \neg q(X)) \wedge \neg r(g1) \wedge (s(Z) \vee \neg t(W) \vee p(X))$

becomes:

$\{ p(X) \vee \neg q(X)\}$, $\{\neg r(g1)\}$, $\{s(Z) \vee \neg t(W) \vee p(X)\}$

Since the only connective in each clause is $\vee$, we can leave that out as well:

$\{ p(X), \neg q(X)\}$, $\{\neg r(g1)\}$, $\{s(Z) , \neg t(W), p(X)\}$

©1993 John Mylopoulos

8

Clause Normal Form

The final transformation:

$\{\neg a_1, \neg a_2, \ldots, \neg a_n, b_1, b_2, \ldots, b_m\}$
$\equiv (\neg a_1 \vee \neg a_2 \vee \ldots \vee \neg a_n) \vee (b_1 \vee b_2 \vee \ldots \vee b_m)$
$\equiv \neg(a_1 \wedge a_2 \wedge \ldots \wedge a_n) \vee (b_1 \vee b_2 \vee \ldots \vee b_m)$
$\equiv (a_1 \wedge a_2 \wedge \ldots \wedge a_n) \Rightarrow (b_1 \vee b_2 \vee \ldots \vee b_m)$

Put the positive literals (if any) on the left hand side, - this is the head of the clause, the rest is the body of the clause

$b_1, b_2, \ldots, b_m$:- $a_1, a_2, \ldots, a_n$

This clause is interpreted as "if a_1 and a_2 and ... and a_n then b_1 or b_2 or ... or b_m"

For example:

$\{\neg s(Z), \neg t(W), p(X)\}$	becomes	p(X) :- s(Z), t(W).
$\{\neg a(U), \neg b(X), \neg c(Y)\}$	becomes	:- a(U), b(X), c(Y)
$\{a(U)\}$	becomes	a(U).

This is the clause normal form

©1993 John Mylopoulos

9

Horn Clause Form

A clause is a Horn clause if it has at most one positive literal.

For example:

{¬ s(Z), ¬ t(W), p(X)} is a Horn Clause
{¬ a(U), ¬ b(X), ¬ c(Y)} is a Horn Clause

{a(X), b(X)} is not a Horn Clause

Horn clauses can be placed in the form

b :- $a_1, a_2, \ldots, a_n$

or

:- $a_1, a_2, \ldots, a_n$ (not allowed in Prolog)

or

b :- (written "b." in Prolog)

10

Horn Clauses - Declarative Meaning

• As indicated earlier, we can think of a clause such as

```
a :- b, c, d ..
```

as meaning "`a` whenever `b` and `c` and `d` ..." because if all of the literals on the right hand side hold, then the left hand side must also hold.

• Clauses which involve just a head and no body, i.e.,

```
a :- .
```

or just `a.`

can be interpreted as meaning that `a` holds (no premises are required for `a` to be true). In other words, `a` states a true fact.

• A clause with no head such as:

```
:- a,b,c, ...
```

(not allowed in Prolog)

can be viewed as a set of premises that has no consequences, i.e., a false set of premises.

11

Horn Clause - Procedural Meaning

• A headed clause such as:

```
a :- b,c,d ..
```

can be viewed as specifying a goal, and its subgoals. In order to prove `a`, prove the sub-goals `b,c,d`.

• A clause with no body such as:

```
a.
```

means that if the inference engine is searching for a goal that unifies with `a`, this goal will succeed.

• A clause with no head such as:

```
:-  a,b,c, ...
```

can be viewed as a sequence of goals to be achieved with respect to a given set of clauses.

©1993 John Mylopoulos

12

Logical Formulae and their Interpretation

An interpretation maps a formula to a truth value which can be either **true** or **false**.

Some formulae are true in all interpretations:

true
$A \vee \neg A$

These formulae are called tautologies.

Some formulae are false in all interpretations:

false
$A \wedge \neg A$

These formulae are called inconsistent.

If an interpretation maps a formula into a true statement, then it is called a model of this formula. A model of a set of formulae is an interpretation where all of them are true.

All interpretations are models of tautologies.

Inconsistent formulae have no models.

©1993 John Mylopoulos

13

Formulae and their Interpretation

The proof theory of a logic only takes into account the syntactic form of the formulae, and disregards their interpretations.

We say that A is a logical consequence of B1,B2, ... if all models of the set B1, B2, ... are also models of A

B1,B2, ... |= A

There exist inference rules, which are correct recipes for deriving logical consequences. For example, modus ponens

A, (A $\Rightarrow$ B)
B

A set of formulae (axioms) and a set of inference rules define a theory which is the set of all formulae (called theorems) that can be proved from the axioms using the theory's inference rules.

A formal derivation of a theorem is called its proof.

B1,B2, ... |- A

©1993 John Mylopoulos

14

Proof by Contradiction

This a particular method of proving that A follows from axioms B1, B2,...:

To derive A from a set of axioms B1,B2, B3 .. assume that ¬ A is a theorem of B1,B2, B3 ...

If the resulting theory is inconsistent, then A is a theorem of the theory with axioms B1,B2,....

A theory is inconsistent if it contains an inconsistent (false) formula, denoted by **[]**.

With this method we often show inconsistency by deriving, for some C: C $\vee$ ¬ C

©1993 John Mylopoulos

15

Proof by Resolution

Resolution theorem proving is a method of theorem proving for Horn clauses that is sound and complete, that is, it is guarenteed to find a proof **iff** one exists.

Resolution Inference Rule

"from $\neg A \vee B$ and from $A \vee C$ infer $B \vee C$"

$$\frac{\neg A \vee B \;,\; A \vee C}{B \vee C}$$

Why?

In any model either $\neg A$ is false, or A is false.
In the first case, B must be true.
In the second case C must be true.
Therefore we can infer $B \vee C$.

More generally,

"from $\neg A \vee B$ and from $A' \vee C$ such that A, A' unify with mgu σ, infer $(B \vee C)\sigma$"

$$\frac{\neg A \vee B \;,\; A' \vee C}{(B \vee C)\sigma} \qquad \text{such that A, A' unify with mgu } \sigma$$

©1993 John Mylopoulos

16

Resolution Rule

"from $\neg A \vee B$ and from $A' \vee C$ such that A, A' unify with mgu σ, infer $(B \vee C)\sigma$"

Special Cases of Resolution Rule:

from $\neg A$ and $A' \vee B$ such that A, A' unify with mgu σ, infer $B\sigma$

from $\neg A$ and A' such that A, A' unify, infer $**$ (a contradiction, or, the empty clause)

The special cases of the resolution rule are used to derive shorter clauses with ultimate goal the generation of the empty clause.

©1993 John Mylopoulos

17

Resolution Rule applied to Horn Clauses

Logical Notation	Prolog Notation
A 1 ∧ A 2 ∧ ...?	(initial question)
¬ (A 1 ∧ A 2 ∧ ...)	(try proof by contradiction)
¬ A 1 ∨ ¬ A 2 ∨ ...	:- A1, A2,... goal
B ∨ ¬ A 1 ∨ ¬ A 2 ∨ ...	B :- A1, A2,... axiom/rule
B	B. axiom

For Horn clauses, from a resolvant with negative literals, we can only derive a conclusion with negative literals using the resolution rule.

Each step of the resolution rule consists of cancelling a negative literal ¬ A from the resolvant, and of replacing it with the negative literals of a clause having A as its only positive literal to produce a new current resolvant.

The resolvant shrinks when one of its literals is cancelled with a unit clause. The set of Horn clauses is inconsistent if [] can be inferred (a contradiction)

©1993 John Mylopoulos

18

Proof Strategy Using Contradiction and Resolution

Start with a question consisting of a conjunction of literals. Negate that and turn it into a Prolog goal G. Call G the current resolvant.

Starting with the current resolvant, repeatedly choose axioms, and/or derived clauses and apply resolution to them to produce a new current resolvant.

If the current resolvant reduces to a contradiction, then G is inconsistent with the axioms included in the Prolog program, therefore ¬ G (the initial question) is a theorem that can be inferred from the axioms.

Restating Clausal form using Prolog syntax as:

- from ¬ A ∨ ¬ B and A ∨ ¬ C infer ¬ B ∨ ¬ C
  from :- A, B. and A :- C infer :- B, C.
- from ¬ A and A ∨ ¬ B infer ¬ B
  from :- A and A :- B infer :- B.
- from ¬ A and A infer **
  **from :- A. and A. Infer **

©1993 John Mylopoulos

19

Proof by Contradiction: An Example

Initial question: breathes(ellen)

animal(X) :- human(X). (1)
human(ellen). (2)
breaths(Y) :- animal(Y). (3)

Initial goal G: ¬ breathes(ellen)

In Prolog :- breathes(ellen). (4)

©1993 John Mylopoulos

20

Proof by contradiction (continued)

animal(X) :- human(X). (1)
human(ellen). (2)
breaths(Y) :- animal(Y). (3)

Initial goal: :- breathes(ellen). (4)

Unify (4) with (3) with the substitution {Y/ellen}.
Using the resolution rule [:-A, A :- B |- :- B.] :

:- animal(ellen). (5)

(5) unifies with (1), with substitution {X/ellen} to give

:- human(ellen). (6)

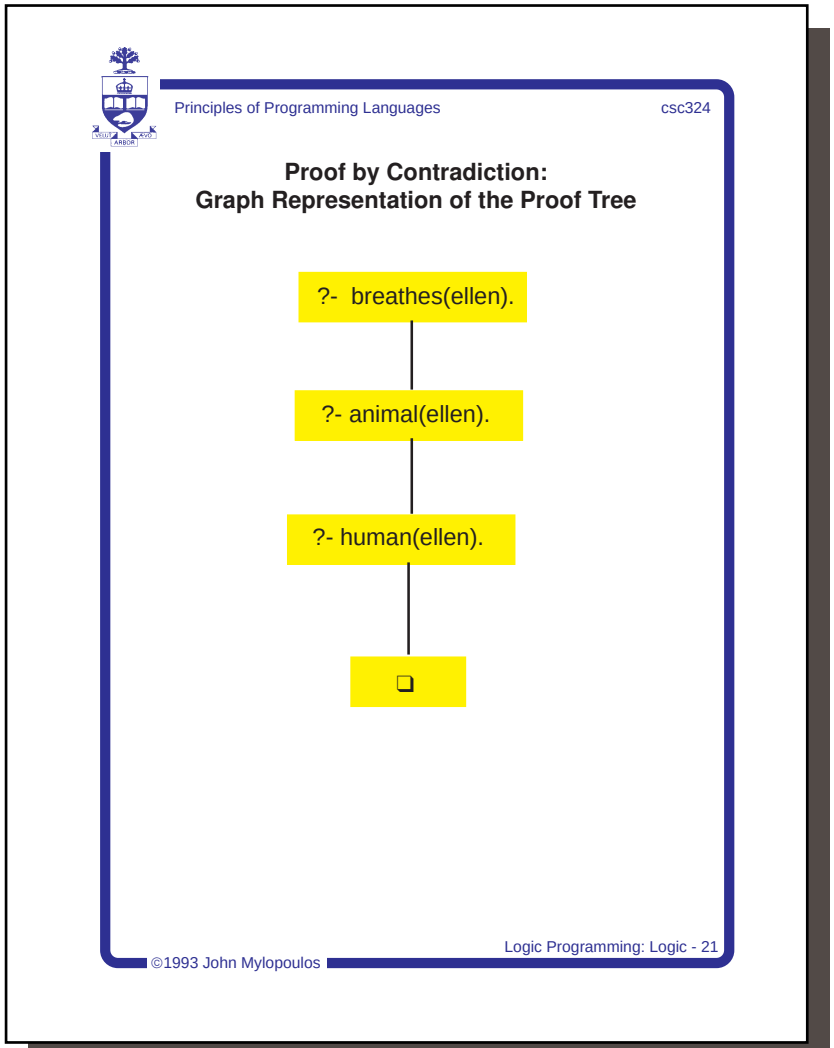
(6) resolves with (2), to give a contradiction **.

**

Hence **G** leads to a contradiction, hence the initial question ¬**G** is proved.

©1993 John Mylopoulos

21


Principles of Programming Languages
csc324
Proof by Contradiction:
Graph Representation of the Proof Tree
?- breathes(ellen).
?- animal(ellen).
?- human(ellen).
❑
Logic Programming: Logic - 21
©1993 John Mylopoulos