Logic in Prolog (2A)

Young Won Lim
4/15/18

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

Predicate Calculus

Atoms, Numbers → Constants

Constants, Variables → Simple Terms

Compound Terms: functor(term$_1$, ... , term$_n$)

Simple Terms, Compound Terms → Predicates

Propositional Logic :
no variable
First Order Logic :
variable → unification

(1) Every Prolog **predicate** : an atomic first-order logic formula

(2) Commas separating subgoals : conjunctions in logic (^)

(3) Prolog rules : implications
(body → head) : (antecedent → consequent)
(head **:-** body) : change the order of head and body

(4) Queries : implications,
(body → ⊥) : (antecedent → consequent ⊥)
(?**-** body)

(5) Every **variable** : universally quantified

Rules

(1) Every Prolog **predicate** : an atomic first-order logic formula

(2) Commas separating subgoals : conjunctions in logic (^)

(3) Prolog rules : implications
(body → head) : (antecedent → consequent)
(head **:-** body) : change the order of head and body

(4) Queries : implications,
(body → ⊥) : (antecedent → consequent ⊥)
(?**-** body)

(5) Every **variable** : universally quantified

head body

is_bigger(X, Y) :- bigger(X, Y).

∀x.∀y.(bigger(x, y) → is_bigger(x, y))

antecedent consequent

Queries

(1) Every Prolog **predicate** : an atomic first-order logic formula

(2) Commas separating subgoals : conjunctions in logic (^)

(3) Prolog rules : implications
(body → head) : (antecedent → consequent)
(head **:-** body) : change the order of head and body

(4) Queries : implications,
(body → ⊥) : (antecedent → consequent ⊥)
(?**-** body)

falsum : contradiction

(5) Every **variable** : universally quantified

empty head body

```
?- is_bigger(elephant, X), is_bigger(X, donkey).
```

∀x.(is_bigger(elephant, x) ^ is_bigger(x, donkey) → ⊥)

antecedent consequent

$(A_1 \wedge A_2 \wedge \cdots \wedge A_n) \rightarrow B$

$\Leftrightarrow \neg(A_1 \wedge A_2 \wedge \cdots \wedge A_n) \vee B$

$\Leftrightarrow \neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_n \vee B$

$B \equiv \bot$

$\neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_n \vee \bot$

$\Leftrightarrow \neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_n$

First Order Logic Formulas

```
bigger(elephant, horse).
bigger(horse, donkey).
is_bigger(X, Y) :- bigger(X, Y).
is_bigger(X, Y) :- bigger(X, Z), is_bigger(Z, Y).
```

{
bigger(elephant, horse),
bigger(horse, donkey),
∀x.∀y.(bigger(x, y) → is_bigger(x, y)),
∀x.∀y.∀z.(bigger(x, z) ^ is_bigger(z, y) → is_bigger(x, y))
}

Horn Formula

$$(A_1 \wedge A_2 \wedge \cdots \wedge A_n) \rightarrow B$$

$$\Leftrightarrow \neg(A_1 \wedge A_2 \wedge \cdots \wedge A_n) \vee B$$

$$\Leftrightarrow \neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_n \vee B$$

$$B \equiv \bot$$

$$\neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_n \vee \bot$$

$$\Leftrightarrow \neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_n$$

$\neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_n$: True when all A_i is false → contradict

This results from the negation of the goal B

Therefore B follows from all A_i

Prolog Query

?- A ⟺ $A \rightarrow \bot$ ⟺ $\neg A$

Putting the negation of the goal in a query into the set of formulas.

Answering means showing that the set of formulas including the translated query is **logically inconsistent**.

to show that A follows from P,
show that adding the negation of A to P will lead to a **contradiction**.

Adding the negation of A to P — contradiction

$P, (A \rightarrow \bot) \vdash \bot$ *iff* $P \vdash A$

the negation of A — A follows from P

Resolution in the Propositional Logic

$$\neg A_1 \lor \neg A_2 \lor \mathbf{B_1}$$
$$\neg \mathbf{B_1} \lor \neg B_2$$
$$\overline{\neg A_1 \lor \neg A_2 \lor \neg B_2}$$

$$\neg A_1 \lor \neg A_2 \lor \boldsymbol{T}$$
$$\boldsymbol{F} \lor \neg B_2$$
$$\overline{\neg B_2 \quad (B_1 = \boldsymbol{T})}$$

$$\neg A_1 \lor \neg A_2 \lor \boldsymbol{F}$$
$$\boldsymbol{T} \lor \neg B_2$$
$$\overline{\neg A_1 \lor \neg A_2 \; (B_1 = \boldsymbol{F})}$$

$\neg A_1 \lor \neg A_2 \lor \mathbf{B_1}$ ➡	$A_1 \land A_2 \rightarrow B_1$ ➡	B1 :- A1, A2
$\neg \mathbf{B_1} \lor \neg B_2$ ➡	$B_1 \land B_2 \rightarrow \bot$ ➡	?- B1, B2
$\neg A_1 \lor \neg A_2 \lor \neg B_2$ ➡	$A_1 \land A_2 \land B_2 \rightarrow \bot$ ➡	?- A1, A2, B2

?- B1, B2

B1 :- A1, A2

?- A1, A2, B2

Find a **fact** or a **rule head** that matches the first **subgoal** b1

Replace the the **subgoal** with the **body** of the found rule

repeated until there are no more subgoals left in the query.

an “empty disjunction” $\bot$

Resolution in the First Order Logic

Propositional Logic : no variable

First Order Logic : variable → unification

Unification :

matching in Prolog

the variable instantiations for successful queries

Negation As Failure – (1)

PLANNER

if (not (goal p)), then (assert ¬p)

If the goal to prove p fails, then assert ¬p

NAF used to derive **not p** (p is assumed not to hold) from failure to derive p

not p can be different from the statement ¬p of the logical negation of p, depending on the **completeness** of the inference algorithm and thus also on the formal logic system

Prolog

NAF literals of the form of not p can occur in the body of clauses

Can be used to derive other NAF literals

p ← q ∧ not r
q ← s
q ← t
t

not p : p is assumed not to hold

¬p : the logical negation of p

completeness of the inference algorithm

semantically complete

every tautology → *theorem*

sound

every theorem → *tautology*

Negation As Failure – (2)

The semantics of NAF remained an open issue until Keith Clark [1978] showed that it is ***correct*** with respect to the **completion** of the logic program, where, loosely speaking,

"**only**" and ← are interpreted as
"**if and only if**", written as "**iff**" or "**≡**".

the completion of the four clauses above is

p ≡ q ∧ not r
q ≡ s ∨ t
t ≡ true
r ≡ false
s ≡ false

interpreted (from)

p ← q ∧ not r
q ← s
q ← t
t ←

Negation As Failure – (3)

the completion of the four clauses above is

p	**≡**	**q ∧ not r**
q	**≡**	**s ∨ t**
t	**≡**	**true**
r	**≡**	**false**
s	**≡**	**false**

interpreted ←

p	**←**	**q ∧ not r**
q	**←**	**s**
q	**←**	**t**
t	**←**	

~~~q~~ ∨ r ∨ p
~s ∨ ~~q~~
~~~t~~ ∨ ~~q~~
~~t~~

The NAF inference rule simulates
reasoning explicitly with the completion,
where <u>*both sides*</u> of the equivalence are <u>*negated*</u>
and *negation on the* <u>*right-hand side*</u> is
<u>*distributed down*</u> *to atomic formulae*.

to show **not p**, NAF simulates
reasoning with the equivalences

not p	**≡**	**not q ∨ r** (≡ false)
not q	**≡**	**not s ∧ not t** (≡ false)
not t	**≡**	**false**
not r	**≡**	**true**
not s	**≡**	**true**

The NAF derives →

not r
not s
t

q
p

Negation As Failure – (4)

In the non-propositional case, (predicate logic with variables)
the completion needs to be augmented with **equality axioms**,
to formalise the assumption that
individuals with distinct names are distinct.
NAF simulates this by **failure of unification**.

For example, given only the two clauses

p(a) ←
p(b) ← t

NAF derives **not p(c)**.

The completion of the program is

p(X) ≡ X=a ∨ X=b **equality axioms**

augmented with **unique names axioms**
and **domain closure axioms**.

The completion semantics is closely related both to
circumscription and to the closed world assumption.

Negation As Failure – (5)

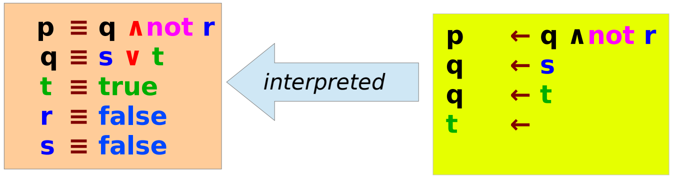
The concept of logical negation in Prolog is problematical,
in the sense that the only method that Prolog can use
to tell if a proposition is false is to try to prove it
(from the facts and rules that it has been told about),
and then if this attempt fails,
it concludes that the proposition is false.

This is referred to as negation as failure.

An obvious problem is that
Prolog may not have been told some critical fact or rule,
so that it will not be able to prove the proposition.

In such a case, the falsity of the proposition
is only relative to the "mini-world-model"
defined by the facts and rules known to the Prolog interpreter.
This is sometimes referred to as the **closed**-world assumption.

A less obvious problem is that,
depending again on the rules and facts
known to the Prolog interpreter, it may take a very long time
to determine that the proposition cannot be proven.
In certain cases, it might "take" infinite time.



Negation As Failure – (6)

Because of the problems of negation-as-failure,
negation in Prolog is represented in modern Prolog interpreters
using the symbol **\+**, which is supposed to be
a mnemonic for **not provable**
with the **** standing for **not** and the **+** for **provable**.
In practice, current Prolog interpreters
tend to support the older operator **not** as well,
as it is present in lots of older Prolog code,
which would break if **not** were not available.

Examples:

```
?- \+ (2 = 4).

true.

?- not(2 = 4).

true.
```

Negation As Failure – (7)

Arithmetic comparison operators in Prolog
each come equipped with a **negation**
which does not have a "negation as failure" problem,
because it is always possible to determine,
for example, if two numbers are equal,
though there may be approximation issues
if the comparison is between fractional (floating-point) numbers.
So it is probably best to use the arithmetic comparison operators
if numeric quantities are being compared.
Thus, a better way to do the comparisons shown above would be:

```
?- 2 =\= 4.

true.
```

Negation As Failure Example 1

```
bachelor(P) :- male(P), not(married(P)).

male(henry).

male(tom).

married(tom).
```

```
?- bachelor(henry).
Yes

?-bachelor(tom).
No

?-bachelor(Who).
Who=henry;
No
```

```
?- not(married(Who)).
No.
```

The first three responses are correct and as expected. The answer to the fourth query might have been unexpected at first. But consider that the goal ?-not(married(Who)) fails because for the variable binding Who=tom, married(Who) succeeds, and so the negative goal fails. Thus, negative goals ?-not(g) with variables cannot be expected to produce bindings of the variables for which the goal g fails.

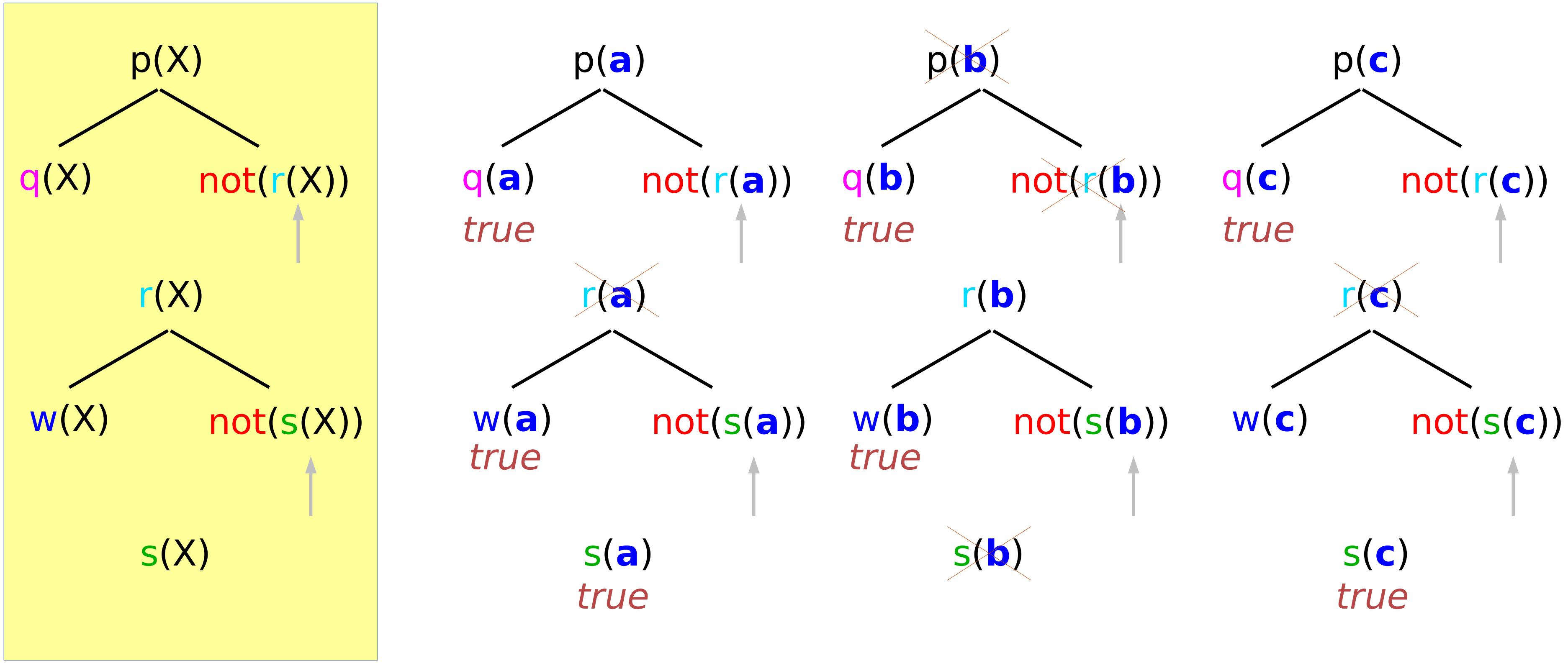
For the variable binding **Who**=tom,
married(**Who**) succeeds
not(married(**Who**)) fails

Negative goals with variables **cannot** be expected to produce **bindings** of the variables for which the goals fails

Right Associative operator

Negation As Failure Example 2

```
p(X) :- q(X), not(r(X)).
r(X) :- w(X), not(s(X)).
q(a). q(b). q(c).
s(a). s(c).
w(a). w(b).
```



References

[1] en.wikipedia.org
[2] en.wiktionary.org
[3] U. Endriss, “Lecture Notes : Introduction to Prolog Programming”
[4] http://www.learnprolognow.org/ Learn Prolog Now!
[5] http://www.csupomona.edu/~jrfisher/www/prolog_tutorial
[6] www.cse.unsw.edu.au/~billw/cs9414/notes/prolog/intro.html
[7] www.cse.unsw.edu.au/~billw/dictionaries/prolog/negation.html