Logic Programming

Theory Lecture 7: Negation as Failure

Richard Mayr

School of Informatics

6th November 2014

Negation as failure: problem 1

Prolog's treatment of *negation* as *failure* is a procedural rather than declarative operation.

For example, the program

```
woman(alice).
man(bob).
```

gives rise to the following queries and responses

```
?- \+man(X), woman(X).
no

?- woman(X), \+man(X).
X = alice
```

So the comma "," can no longer be understood as the commutative operation of logical conjunction.

Negation as failure: problem 1

Prolog's treatment of *negation* as *failure* is a procedural rather than declarative operation.

For example, the program

```
woman(alice).
man(bob).
```

gives rise to the following queries and responses

```
?- \+man(X), woman(X).
no

?- woman(X), \+man(X).
X = alice
```

So the comma "," can no longer be understood as the commutative operation of logical conjunction.

The problem here is the use of negation on non-ground formulas.

Negation as failure: problem 2

Even for ground instances of negation, the procedural behaviour of Prolog programs does not match the declarative reading.

Example program and query.

```
p :- \+p.
```

```
?- p.
```

Prolog goes into a loop on this, but under the declarative reading, `p` is a logical consequence of the theory.

Negation as failure: problem 2

Even for ground instances of negation, the procedural behaviour of Prolog programs does not match the declarative reading.

Example program and query.

```
p :- \+p.

?- p.
```

Prolog goes into a loop on this, but under the declarative reading, `p` is a logical consequence of the theory.

The problem here is the inclusion of negated formulas in the program.

Restricting negation as failure

Today we shall make declarative sense of negation as failure with two major restrictions.

- Only ground atomic formulas appear negated.
- Negations appear only in queries, not in programs.

Even this restrictive setting causes complications.

```
woman(alice).
man(bob).
```

```
?- \+man(alice).
yes
```

However,

$$\texttt{woman(alice)}, \texttt{man(bob)} \not\models \neg\texttt{man(alice)}$$

Thus the conclusion is not a logical consequence of the program.

Even this restrictive setting causes complications.

```
woman(alice).
man(bob).

?- \+man(alice).
yes
```

However,

$$\texttt{woman(alice)}, \texttt{man(bob)} \not\models \neg\texttt{man(alice)}$$

Thus the conclusion is not a logical consequence of the program.

We address this by “completing” the program to a larger logical theory of which the conclusion is a consequence.

Circumscribing knowledge

We view our program as a logical theory expressing knowledge about the world.

In several situations, it is convenient to assume that the program contains complete information about certain kinds of logical statements.

We can then make additional inferences about the world based on the assumed completeness of our knowledge.

The *Closed World Assumption (CWA)* makes the assumption that the program contains complete knowledge about which ground atomic formulas are true.

Consider our simple example

```
woman(alice).
man(bob).
```

If we impose the CWA then we can conclude that neither `woman(bob)` nor `man(alice)` hold, since if either of these were true then our program would not contain complete knowledge about true ground atomic formulas.

Thus the CWA allows us to "complete" our knowledge base to the larger theory

$$\texttt{woman(alice)}, \texttt{man(bob)}, \neg\texttt{woman(bob)}, \neg\texttt{man(alice)}$$

Note that we are here adding negated atomic formulas to the knowledge base.

Closed World Assumption in general

A *theory* T in predicate logic is a set of sentences. (We do not insist that T contain only definite clauses.)

A theory T is said to be *complete for ground atomic formulas* if, for every ground atomic formula A, either $T \models A$ or $T \models \neg A$.

Let T be any *theory* (it does not have to be complete for ground atomic formulas). We define a new theory $CWA(T)$ by:

$$CWA(T) \; = \; T \cup \{\neg A \mid A \text{ is a ground atomic formula and } T \not\models A\}$$

By definition, the theory $CWA(T)$ is complete for ground atomic formulas.

(Formulas that are either atomic or negated atomic are often called *literals*. Thus $CWA(T)$ is complete for ground literals.)

Example 1

Let T be the theory (in definite clause logic)

$$\begin{aligned}
\forall \texttt{X}.\ \texttt{cheap(X)} &\rightarrow \texttt{nasty(X)} \\
\forall \texttt{X}.\ \texttt{free(X)} &\rightarrow \texttt{cool(X)} \\
& \quad \texttt{cheap(windows)} \\
& \quad \texttt{free(linux)} \\
& \quad \texttt{cool(mac)}
\end{aligned}$$

Example 1

Let T be the theory (in definite clause logic)

$$
\begin{aligned}
\forall \mathtt{X}.\ \mathtt{cheap(X)} &\rightarrow \mathtt{nasty(X)} \\
\forall \mathtt{X}.\ \mathtt{free(X)} &\rightarrow \mathtt{cool(X)} \\
&\mathtt{cheap(windows)} \\
&\mathtt{free(linux)} \\
&\mathtt{cool(mac)}
\end{aligned}
$$

Then $CWA(T)$ adds the following ground atomic formulas to T

$\neg\mathtt{free(windows)}$, $\neg\mathtt{cool(windows)}$, $\neg\mathtt{cheap(linux)}$, $\neg\mathtt{nasty(linux)}$, $\neg\mathtt{free(mac)}$, $\neg\mathtt{cheap(mac)}$, $\neg\mathtt{nasty(mac)}$

Closed World Assumption in definite clause logic

Note that the theory $CWA(T)$ is not itself a definite clause theory, even when T is a definite clause theory. However, the assumption that T is a definite clause theory has one important consequence.

Theorem

Let T be a set of definite clauses. Then the theory $CWA(T)$ is consistent (that is, there is no formula F such that $CWA(T) \models F$ and $CWA(T) \models \neg F$).

Proof

For a ground atomic formula A, the minimal Herbrand model $\mathcal{H}$ of T satisfies that $\mathcal{H} \models A$ if and only if $T \models A$ (see Lecture 6 "Importance of minimal Herbrand model"). Hence, if $T \not\models A$ then $\mathcal{H} \models \neg A$. Thus the minimal Herbrand model for T is also a model for $CWA(T)$. Any theory that has a model is consistent.

Soundness of negated queries

Suppose T is a definite clause theory and A is a ground atomic formula. Suppose also that the prolog query

```
?- \+ A.
```

succeeds. Then $CWA(T) \models \neg A$.

Also $\mathcal{H} \models \neg A$, where $\mathcal{H}$ is the minimal Herbrand model for T.

Prolog's behaviour on queries including negated ground queries can thus be considered either as sound relative to the Closed World Assumption, or as sound relative to the minimal Herbrand model.

Non-monotonicity

Logical consequence is *monotonic* in the sense that, given two theories T, T' with $T \subseteq T'$ then it holds that

$$T \models F \quad \text{implies} \quad T' \models F \ ,$$

for all formulas F.

The Closed World Assumption is *non-monotonic* in the sense that, given two theories T, T' with $T \subseteq T'$ then it *does not* hold in general that

$$CWA(T) \models F \quad \text{implies} \quad CWA(T') \models F \ ,$$

for all formulas F.

Example 1 (continued)

Let T' be our example theory extended with the new axiom

$$\texttt{cheap(linux)}$$

Then $CWA(T')$ adds the following ground atomic formulas to T'

$$\neg\texttt{free(windows)}, \neg\texttt{cool(windows)}, \\ \neg\texttt{free(mac)}, \neg\texttt{cheap(mac)}, \neg\texttt{nasty(mac)}$$

We have $T \subseteq T'$ and

$CWA(T) \models \neg\texttt{cheap(linux)}$ but $CWA(T') \not\models \neg\texttt{cheap(linux)}$

where the latter claim holds because $CWA(T') \models \texttt{cheap(linux)}$ and $CWA(T')$ is consistent (since T' is a definite clause theory).

This illustrates non-monotonicity.

Summarizing theorem

Theorem. Let T be a definite clause theory, and let $\mathcal{H}$ be its minimum Herbrand model.

Then the statements below, about any ground atomic formula `A`,

1. The Prolog query `\+ A` succeeds.
2. $CWA(T) \models \neg \mathtt{A}$.
3. $\mathcal{H} \models \neg \mathtt{A}$.

enjoy the following implications

$$1 \implies 2 \iff 3$$

Two issues with the CWA

1. Because of the *undecidability* of definite clause predicate logic (see Lecture 5), it is not possible to compute the theory $CWA(T)$ from the theory T.

2. The CWA over-approximates the behaviour of negation by failure. Consider the propositional theory `T` consisting of a single axiom

   $$\mathtt{p} \rightarrow \mathtt{p}$$

   Then the Prolog query `\+ p` goes into a loop. Nevertheless,

   $$CWA(T) \models \neg \mathtt{p}$$

Clark completion

The *Clark completion* is an alternative completion procedure, used for modelling negation by failure.

In contrast to the CWA, the Clark completion *is* computable.

However, whereas $CWA(T)$ can be defined for any logical theory T, the Clark completion requires T to be a definite-clause theory. (Actually, Clark completion can be defined for a generalization of definite-clause logic including negation. But this is beyond the scope of this lecture.)

Roughly, the Clark completion makes the assumption that the axioms of the definite clause program completely axiomatize all possible reasons for atomic formulas to be true.

Example 2

Suppose we have a theory in which the only clauses with *head predicate* `british` are

$$\begin{aligned} \forall \mathtt{X}.\ \mathtt{english(X)} &\rightarrow \mathtt{british(X)} \\ \forall \mathtt{X}.\ \mathtt{scottish(X)} &\rightarrow \mathtt{british(X)} \\ \forall \mathtt{X}.\ \mathtt{welsh(X)} &\rightarrow \mathtt{british(X)} \end{aligned}$$

Then the Clark formula for the predicate `british` is

$$\forall \mathtt{X}. (\mathtt{british(X)} \leftrightarrow (\mathtt{english(X)} \vee \mathtt{scottish(X)} \vee \mathtt{welsh(X)}))$$

(The *head predicate* in a clause is: the predicate on the right-hand side of the implication, if the clause is an implication; and the only predicate in the formula, if the clause is an atomic formula.)

Example 2 (continued)

Suppose the remaining axioms are

```
english(elizabeth)
scottish(mary)
scottish(james)
```

Then the Clark formulas for the three predicates `english`, `scottish`, `welsh` are

$$\forall \mathtt{X}.\ (\mathtt{english(X)} \leftrightarrow \mathtt{X} = \mathtt{elizabeth})$$
$$\forall \mathtt{X}.\ (\mathtt{scottish(X)} \leftrightarrow (\mathtt{X} = \mathtt{mary} \lor \mathtt{X} = \mathtt{james}))$$
$$\forall \mathtt{X}.\ (\neg \mathtt{welsh(X)})$$

Example 2 (completed)

The Clark completion of the full theory:

$$\forall \mathtt{X}.\ \mathtt{english(X)} \rightarrow \mathtt{british(X)}\ ,$$
$$\forall \mathtt{X}.\ \mathtt{scottish(X)} \rightarrow \mathtt{british(X)}\ ,$$
$$\forall \mathtt{X}.\ \mathtt{welsh(X)} \rightarrow \mathtt{british(X)}\ ,$$
$$\mathtt{english(elizabeth)},\ \mathtt{scottish(mary)},\ \mathtt{scottish(james)}$$

is the theory:

$$\forall \mathtt{X}.\,(\mathtt{british(X)} \leftrightarrow (\mathtt{english(X)} \vee \mathtt{scottish(X)} \vee \mathtt{welsh(X)}))$$
$$\forall \mathtt{X}.\,(\mathtt{english(X)} \leftrightarrow \mathtt{X} = \mathtt{elizabeth})$$
$$\forall \mathtt{X}.\,(\mathtt{scottish(X)} \leftrightarrow (\mathtt{X} = \mathtt{mary} \vee \mathtt{X} = \mathtt{james}))$$
$$\forall \mathtt{X}.\,(\neg \mathtt{welsh(X)})$$
$$\mathtt{elizabeth} \neq \mathtt{james} \quad \mathtt{james} \neq \mathtt{mary} \quad \mathtt{mary} \neq \mathtt{elizabeth}$$

Note that the Clark completion is not itself a definite clause theory.

General completion procedure: Step 1

We now consider the general procedure for constructing the Clark completion $Comp(T)$ of a definite clause theory T.

First we rewrite each individual definite clause in the theory. The general form of a definite clause is

$$\forall \vec{\mathtt{X}}.\ (A_1 \wedge \cdots \wedge A_k \ \rightarrow\ p(\vec{t}))$$

where $\vec{\mathtt{X}}$ is a tuple of variables, and $\vec{t}$ is a tuple of n-terms, where n is the arity (= number of arguments) of the predicate p.

We rewrite the clause to the equivalent formula

$$\forall \vec{\mathtt{Y}}.\ (\exists \vec{\mathtt{X}}.\ A_1 \wedge \cdots \wedge A_k \wedge \vec{\mathtt{Y}} = \vec{t}) \ \rightarrow\ p(\vec{\mathtt{Y}})$$

where $\vec{\mathtt{Y}}$ is a tuple of n new variables.

Justifying this equivalence

The formula

$$\forall \vec{\mathtt{X}}.\ (A_1 \wedge \cdots \wedge A_k \ \rightarrow\ p(\vec{t}))$$

is equivalent to

$$\forall \vec{\mathtt{Y}}, \vec{\mathtt{X}}.\ (A_1 \wedge \cdots \wedge A_k \wedge \vec{\mathtt{Y}} = \vec{t} \ \rightarrow\ p(\vec{\mathtt{Y}}))$$

which is, in turn, equivalent to the desired formula

$$\forall \vec{\mathtt{Y}}.\ (\exists \vec{\mathtt{X}}.\ A_1 \wedge \cdots \wedge A_k \wedge \vec{\mathtt{Y}} = \vec{t}) \ \rightarrow\ p(\vec{\mathtt{Y}})$$

because of the general logical equivalence

$$(\forall \mathtt{X}.\ (F \rightarrow G)) \quad \leftrightarrow \quad ((\exists \mathtt{X}.\ F) \rightarrow G)\ ,$$

which holds whenever the variable $\mathtt{X}$ does not appear in the formula G.

Some special cases

$$\forall \vec{X}.\ (A_1 \wedge \cdots \wedge A_k \ \rightarrow\ p(\vec{t}))$$

For special cases of this formula, one can simplify the equivalent formulas.

When $k = 0$ (i.e., the axiom is a Prolog fact rather than rule) the equivalent formula is

$$\forall \vec{Y}.\ (\exists \vec{X}.\ \vec{Y} = \vec{t}) \ \rightarrow\ p(\vec{Y})$$

When the formula is ground, the equivalent formula simplifies to

$$\forall \vec{Y}.\ A_1 \wedge \cdots \wedge A_k \wedge \vec{Y} = \vec{t} \ \rightarrow\ p(\vec{Y})$$

When $\vec{t}$ is the vector of variables $\vec{X}$, there is no need to further rewrite the original clause

$$\forall \vec{X}.\ (A_1 \wedge \cdots \wedge A_k \ \rightarrow\ p(\vec{X}))$$

General completion procedure: Step 2

We have now rewritten each clause with head predicate p to an equivalent formula

$$\forall \vec{Y}.\ E \ \rightarrow\ p(\vec{Y})$$

Suppose there are m such clauses for p, giving

$$\begin{array}{c}
\forall \vec{Y}.\ E_1 \ \rightarrow\ p(\vec{Y}) \\
\forall \vec{Y}.\ E_2 \ \rightarrow\ p(\vec{Y}) \\
\vdots \\
\forall \vec{Y}.\ E_m \ \rightarrow\ p(\vec{Y})
\end{array}$$

Taken together, these formulas are equivalent to the single formula

$$\forall \vec{Y}.\ (E_1 \vee E_2 \vee \cdots \vee E_m) \ \rightarrow\ p(\vec{Y})$$

The *Clark formula* for the predicate p is then the formula

$$\forall \vec{Y}.\ p(\vec{Y}) \ \leftrightarrow\ (E_1 \vee E_2 \vee \cdots \vee E_m)$$

A special case

In the case that $m = 0$ (i.e., when there are no clauses with head predicate p) the Clark formula is simply

$$\forall \vec{\mathrm{Y}}.\, \neg p(\vec{\mathrm{Y}})$$

This can be understood as a genuine special case of the previous definition, since the correct definition of an empty disjunction is the truth value **false**.

General completion procedure: Step 3

The *Clark completion*, $Comp(T)$ of the definite clause theory T is the theory consisting of:

- Clark formulas for every predicate p appearing in the theory T.
- $\neg(t_1 = t_2)$ for every pair t_1, t_2 of non-unifiable terms.

Clark completion of Example 1

As another illustrative example, here is the Clark completion of Example 1.

$$\begin{gathered}
\forall \mathtt{X}.\, \mathtt{nasty(X)} \;\leftrightarrow\; \mathtt{cheap(X)} \\
\forall \mathtt{X}.\, \mathtt{cool(X)} \;\leftrightarrow\; (\mathtt{free(X)} \lor \mathtt{X} = \mathtt{mac}) \\
\forall \mathtt{X}.\, \mathtt{cheap(X)} \;\leftrightarrow\; \mathtt{X} = \mathtt{windows} \\
\forall \mathtt{X}.\, \mathtt{free(X)} \;\leftrightarrow\; \mathtt{X} = \mathtt{linux} \\
\mathtt{linux} \neq \mathtt{mac} \quad \mathtt{mac} \neq \mathtt{windows} \quad \mathtt{windows} \neq \mathtt{linux}
\end{gathered}$$

Properties of Clark completion

1. The theory $Comp(T)$ extends T.
   That is, $T \models F$ implies $Comp(T) \models F$, for all formulas F.
2. The theory $Comp(T)$ is consistent.
   Indeed, the minimal Herbrand model of T is a model of $Comp(T)$.
3. If $Comp(T) \models A$, where A is an atomic formula, then $T \models A$.
   (That is, the Clark completion adds no new *positive* information.)
4. If the prolog query `\+ A` succeeds, where A is a ground atomic formula. Then $Comp(T) \models \neg A$.
   (That is, negation by failure for ground queries is sound relative to the Clark completion.)

Two issues with the CWA revisited

1. In contrast to the CWA, one can compute $\mathit{Comp}(T)$ from T. Indeed, the description we have given for this theory essentially gives an algorithm for constructing it.
2. The Clark completion more precisely captures the behaviour of Prolog's negation by failure. Consider again the propositional theory `T` consisting of a single axiom

   $$\mathtt{p} \rightarrow \mathtt{p}$$

   As before, the Prolog query `\+ p` goes into a loop. The Clark completion of this theory is the theory

   $$\mathtt{p} \leftrightarrow \mathtt{p}$$

   Thus $\mathit{Comp}^{+}(T) \not\models \neg\mathtt{p}$, which more closely models Prolog's behaviour.

Clark completion — summary

- The Clark completion of T can be effectively computed from T and soundly models negation by failure.
- It is more faithful to Prolog behaviour on negated queries than the CWA.
- Although it has been described in this lecture for definite clause theories only, the Clark completion can be more generally defined for a more general class of “negation as failure” programs and goals.

Main points today

Procedural nature of negation by failure

Closed World Assumption

Non-monotonicity of CWA

Clark completion