Introduction to AI

Non-monotonic Reasoning

Francesca Toni

(thanks to Marek Sergot and Kostas Stathis)

Section 10.7, Russell and Norvig book

Outline

- Classical logic: the qualification problem
- Closed World Assumption
- (non-)Monotonicity
- Non-monotonic – defeasible – reasoning
- Negation-as-failure in logic programming (and Prolog)

We will use the Prolog convention on variables/constant/function symbols. Also clauses/rules will be implicitly universally quantified.

The Qualification Problem (1)

"All birds can fly..."

$\text{flies}(X) \leftarrow \text{bird}(X)$

"...unless they are penguins..."

$\text{flies}(X) \leftarrow \text{bird}(X), \neg\, \text{penguin}(X)$

"...or ostriches..."

$\text{flies}(X) \leftarrow \text{bird}(X), \neg\, \text{penguin}(X), \neg\, \text{ostrich}(X)$

"...or wounded..."

$\text{flies}(X) \leftarrow \text{bird}(X), \neg\, \text{penguin}(X), \neg\, \text{ostrich}(X), \neg\, \text{wounded}(X)$

"...or dead, or sick, or glued to the ground, or..."

Note: the frame problem (planning) is a special case of the qualification problem

The Qualification Problem (2)

Let BIRDS be the set of sentences about flying birds.

Even if we could list all these exceptions, classical logic would still not allow:

$$\text{BIRDS} \cup \text{bird(tweety)} \models \text{flies(tweety)}$$

Note: we would also have to affirm all the *qualifications*, viz.:

¬ penguin(tweety) ¬ ostrich(tweety)

¬ wounded(tweety) ¬ dead(tweety)

¬ sick(tweety) ¬ glued_to_the_ground(tweety)

…

Classical logic is inadequate

What we want is a new kind of ‘entailment’:

$\text{BIRDS} \cup \text{bird(tweety)} \models^* \text{flies(tweety)}$

Namely, from BIRDS and bird(tweety) it follows *by default* – in the absence of information to the contrary – that flies(tweety).

This kind of reasoning will be *defeasible*.

Non-monotonic logics

Classical logic is *monotonic*. For a set of sentences S:

$$\text{if } S \models \alpha \text{ then } S \cup X \models \alpha$$

Namely, new information X always preserves old conclusions α.

Reasoning by default is typically *non-monotonic*. We may have that:

$$S \models^{*} \alpha \text{ but } S \cup X \not\models^{*} \alpha$$

Examples:

- DRINKS ∪ coffee $\models^{*}$ tastes nice
- DRINKS ∪ coffee ∪ diesel oil $\not\models^{*}$ tastes nice
- BIRDS ∪ bird(tweety) $\models^{*}$ flies(tweety)
- BIRDS ∪ bird(tweety) ∪ penguin(tweety) $\not\models^{*}$ flies(tweety)

There have been huge developments in AI over the last 30 years in non-monotonic logic for default reasoning and other applications.

The 'Closed World Assumption'

Consider the set of sentences that could be describing a database DB:

has-office-in(IBM, Winchester)	city(Winchester)
has-office-in(IBM, London)	city(London)
has-office-in(IBM, Paris)	city(Paris)
has-office-in(MBI, London)	city(NewYork)
capital-city(London)	company(IBM)
capital-city(Paris)	company(MBI)

Does MBI have an office in Paris?
Does IBM have an office in New York?

We do not know. But it is usual in databases (and many other contexts) to make a *Closed World Assumption* (CWA):

if α is not in the DB, assume $\neg\alpha$

Credulous vs Sceptical reasoning: "The Nixon diamond"

- Quakers are typically pacifists.
- Republicans are typically not pacifists.
- Richard Nixon is a Quaker
- Richard Nixon is a Republican

Do we conclude that Nixon is a pacifist or not?

Sceptical (or cautious) reasoning:

No, since we cannot choose between two possible, conflicting default conclusions.

Credulous (or brave) reasoning:

Yes, to both, since we have reason to believe both.

Which form of reasoning to choose? It depends on the application.

Another example

- Alcoholics are typically adults
- Adults are typically healthy

Do we conclude:

- Alcoholics are typically healthy?

Normal logic programs

A normal logic program is a set of clauses of the form:

$$A \leftarrow L_1, \ldots, L_n \quad (n \geq 0)$$

where A is an atom and each L_i is a literal.
A literal is either an atom (a 'positive literal') or of the form

not B

where B is an atom. (not B is a 'negative literal').

The atom A is the head of the clause; the literals $L_1, \ldots, L_n$ are the body of the clause. When the body is empty ($n = 0$ above) the arrow $\leftarrow$ is usually omitted.

not is negation as failure (NAF):
not B succeeds when all attempts to prove B fail. (There are more precise ways of saying this, but this will do here).

Negation-as-failure: Operational Semantics (1)

We have already seen the computation of a goal (query) $G = L_1, \ldots, L_m$ as a series of derivation steps:

$$
\begin{array}{ll}
 & \leftarrow G \\
\theta_1 & \Big| \\
 & \leftarrow G_1 \\
\theta_2 & \Big| \\
 & \vdots \\
 & \leftarrow G_{n-1} \\
\theta_n & \Big| \\
 & \square
\end{array}
$$

- θ_i is the mgu at the i-th derivation step
- The answer computed is (the restriction to the vars of G of) the composition of all these mgus:

  $\theta = \theta_1 \circ \ldots \circ \theta_n$

Negation-as-failure: Operational Semantics (2)

Now there are two kinds of derivation steps:

(a) select a positive literal $L_j = B$ from the current goal:

$$\leftarrow L_1,\ldots,L_{j-1}, B, L_{j+1},\ldots,L_n$$

θ_i | match B with $B' \leftarrow M_1,\ldots,M_k$, with $B\theta_i = B'\theta_i$

$$\leftarrow (L_1,\ldots,L_{j-1}, M_1,\ldots,M_k, L_{j+1},\ldots,L_n)\theta_i$$

(b) select a negative literal L_j = not B from the current goal:

$$\leftarrow L_1,\ldots,L_{j-1}, \text{not } B, L_{j+1},\ldots,L_n$$

$\theta_i = \{\}$ | subcomputation:

all ways of computing goal B must fail (finitely)

$$\leftarrow (L_1,\ldots,L_{j-1}, L_{j+1},\ldots,L_n)$$

Note that the NAF sub-computation just checks not B: it does not generate bindings (substitutions) for variables.

Selection of sub-goals

- Sub-goals can be selected in any order. The answers are the same, whatever the selection rule is, though the efficiency of the computation can change.
- A language like Prolog selects always the leftmost sub-goal in the current goal.
- Strictly, the selection of sub-goals must be safe: it must not pick a negative literal containing a variable. (Most Prolog systems do not implement this!)

Example

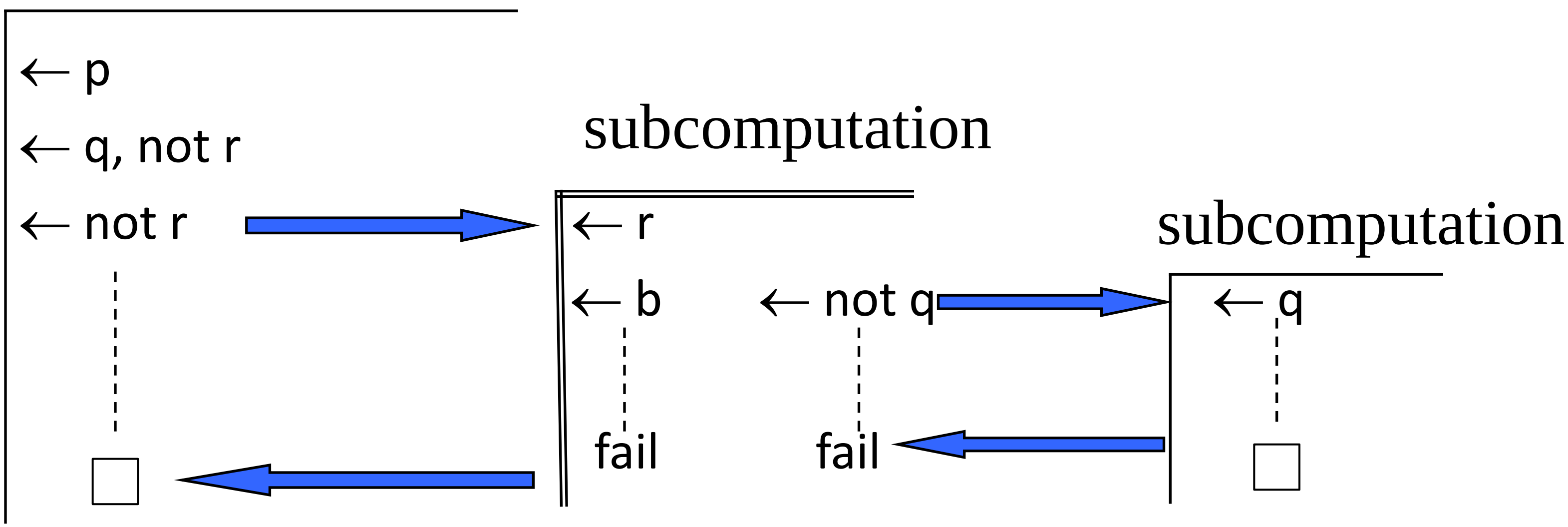
P: p ← q, **not** r

r ← b

r ← **not** q

q ←

Goal: p



computed answer: {}

Negation-as-failure is non-monotonic

Consider the set of sentences S:

$p(X) \leftarrow q(X), \text{not } r(X)$

$q(a)$

$r(b)$

Then $S \vdash_{N\,AF} p(a)$

but $S \cup \{r(a)\} \nvdash_{N\,AF} p(a)$

In fact, negation-as-failure builds in a kind of Closed World Assumption whereby

$S \cup \{r(a)\} \vdash_{N\,AF} \neg\, p(a)$

Rules & Exceptions using NAF

It is reasonably straightforward to formulate general rule and exception structures using NAF

<u>Example</u>

- Typically (by default, unless there is reason to think otherwise,...) a bird can fly.
- Except that dead birds cannot fly

can fly(X) ← bird(X), not abnormal_bird(X)
abnormal_bird(X) ← dead(X)

<u>Example</u>

People are innocent by default (unless they can be proven to be guilty)

innocent(X) ← not guilty(X)

The Nixon diamond using NAF (1)

- Quakers are typically pacifists.
- Republicans are typically not pacifists.
- Richard Nixon is a Quaker
- Richard Nixon is a Republican

Do we conclude that Nixon is a pacifist or not?

pacifist(X) ← quacker(X), not abnormal_quacker(X)
not_pacifist(X) ← republican(X), not abnormal_republican (X)

abnormal_quacker(X) ← not_pacifist(X)
abnormal_republican (X) ← pacifist(X)

quacker(nixon)
republican(nixon)

The Nixon diamond using NAF (2)

By shortening predicates (pacifist becomes p etc)

p(X) ← q(X), not ab_q(X)

ab_q(X) ← n_p(X)

q(nixon)

n_p(X) ← r(X), not ab_r (X)

ab_r (X) ← p(X)

r(nixon)

Goal (for example. Here nixon becomes nix): p(nix)

← p(nix)

← q(nix), not ab_q(nix)

← not ab_q(nix) →

← ab_q(nix)

← n_p(nix)

← r(nix), not ab_r (nix)

← not ab_r (nix) →

← ab_r (nix)

…

Infinite computation!

Answer set programming (1)

Given a set of definite clauses P:

- the *Herbrand universe* is the set of all ground terms obtained from constants and function symbols in P, and the *Herbrand base* (HB) is the set of all atoms from predicate symbols in P and terms in the Herbrand universe
- A *Herbrand model* is a subset of the Herbrand base that renders P true
- The meaning of P is the *least Herbrand model* (LHM) of P

Note: the LHM can be computed bottom-up (a-la-forward chaining)

Answer set programming (2)

- Given a (ground) normal logic program P and $S \subseteq HB$, the *reduct of P by S* (referred to as P^S) is obtained in two steps:
  1. Eliminate all rules with not p in the body, for every $p \in S$
  2. Eliminate all negative literals from the body of all remaining rules
- P^S is a set of definite clauses.
- $S \subseteq HB$ is an *answer set* of P iff the LHM of P^S is S
- Several efficient answer set solvers exist

Nixon diamond using Answer set programming (1)

p(X) ← q(X), not ab_q(X)

ab_q(X) ← n_p(X)

q(nix)

n_p(X) ← r(X), not ab_r (X)

ab_r (X) ← p(X)

r(nix)

Here HB={q(nix), r(nix), p(nix),not_p(nix), ab_q(nix), ab_r(nix)}. Let P be the set of all ground instances of these rules over HB.

Consider S={p(nix), ab_r (nix), q(nix), r(nix),}.

Is S an answer set? YES. To see this, let us construct P^S

1. Eliminates n_p(nix) ← r(nix), not ab_r(nix)
2. Gives p(nix) ← q(nix),

   ab_q(nix) ← n_p(nix), ab_r (nix) ← p(nix),

   q(nix), r(nix)

The LHM of P^S is {q(nix), r(nix), p(nix), ab_r (nix) }=S, as required.

Nixon diamond using Answer set programming (2)

S={q(nix), r(nix), n_p(nix), ab_q(nix)} is also an answer set

- Multiple "extensions" = answer sets
- Credulous (whatever holds in some answer set) vs sceptical (whatever holds in all answer sets)

Summary

- We have discussed:
- Classical logic: the qualification problem
- Closed World Assumption
- (non-)Monotonicity
- Non-monotonic – defeasible – reasoning
- Negation-as-failure for defeasible reasoning
- Answer set programming
- Examples