Introduction to AI

Non-monotonic Reasoning

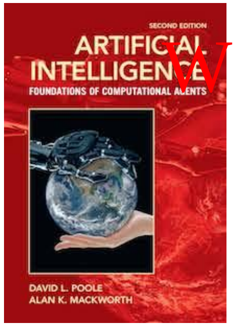
Francesca Toni

(thanks to Marek Sergot)

Outline

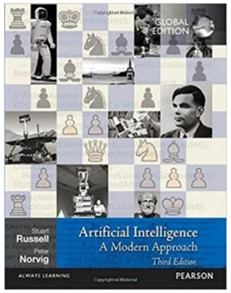
- Classical logic: the qualification problem
- Closed World Assumption
- (non-)Monotonicity
- Non-monotonic – defeasible – reasoning
- Negation-as-failure in logic programming (and Prolog)

Recommended:
Section 5.6



+ sections 1-3 of paper
on stable models (on cate)

Additional :
Section 9.4.5
Section 12.6



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

The Qualification Problem (1)

“All birds can fly...”

flies(X) ← bird(X)

“...unless they are penguins...”

flies(X) ← bird(X), ¬ penguin(X)

“...or ostriches...”

flies(X) ← bird(X), ¬ penguin(X), ¬ ostrich(X)

“...or wounded...”

flies(X) ← bird(X), ¬ penguin(X), ¬ ostrich(X), ¬wounded(X)

“...or dead, or sick, or glued to the ground, or...”

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:

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

unless we also affirm all the *qualifications*, viz.:

¬ penguin(tweety)

¬ wounded(tweety)

¬ sick(tweety)

¬ ostrich(tweety)

¬ dead(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*.

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 is a branch of AI focusing on non- monotonic logic for default reasoning – within knowledge representation and reasoning in AI.

The ‘Closed World Assumption’

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

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

city(Winchester)
city(London)
city(Paris)
city(NewYork)
company(IBM)
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 ¬α

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 ← is usually omitted.

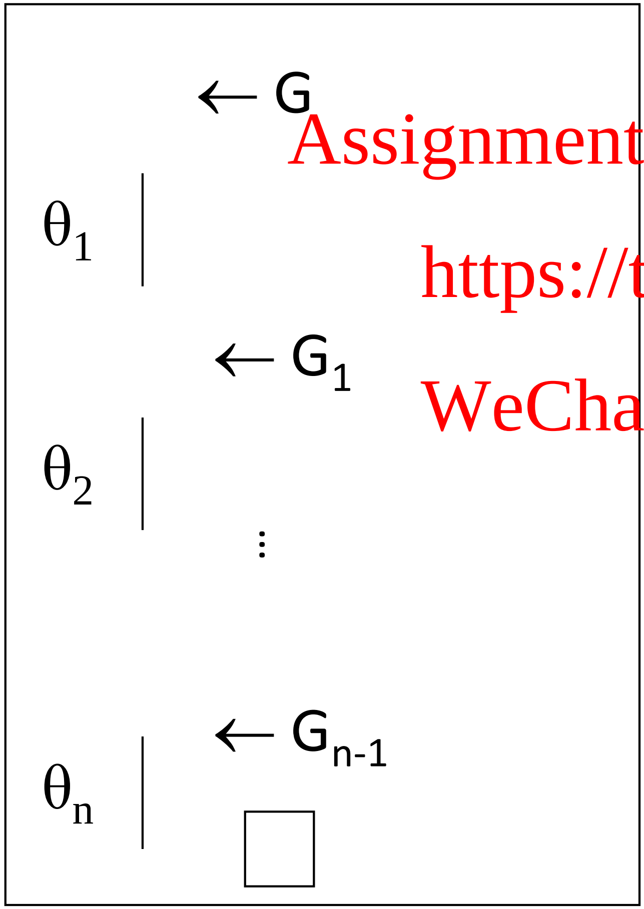
not is negation as failure (NAF): informally (for now)

not B succeeds when all attempts to prove B fail

SLD+Negation-as-failure: SLDNF (1)

We have already seen the computation of a goal (query)

$G = L_1, \ldots, L_m$ as a series of derivation steps:



$\leftarrow G$

θ_1

$\leftarrow G_1$

θ_2

$\vdots$

$\leftarrow G_{n-1}$

θ_n

$\square$

- 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:

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

SLD+Negation-as-failure: SLDNF (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:**

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

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

all ways of computing goal B must fail (finitely)

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

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

SLDNF Example (S is ground)

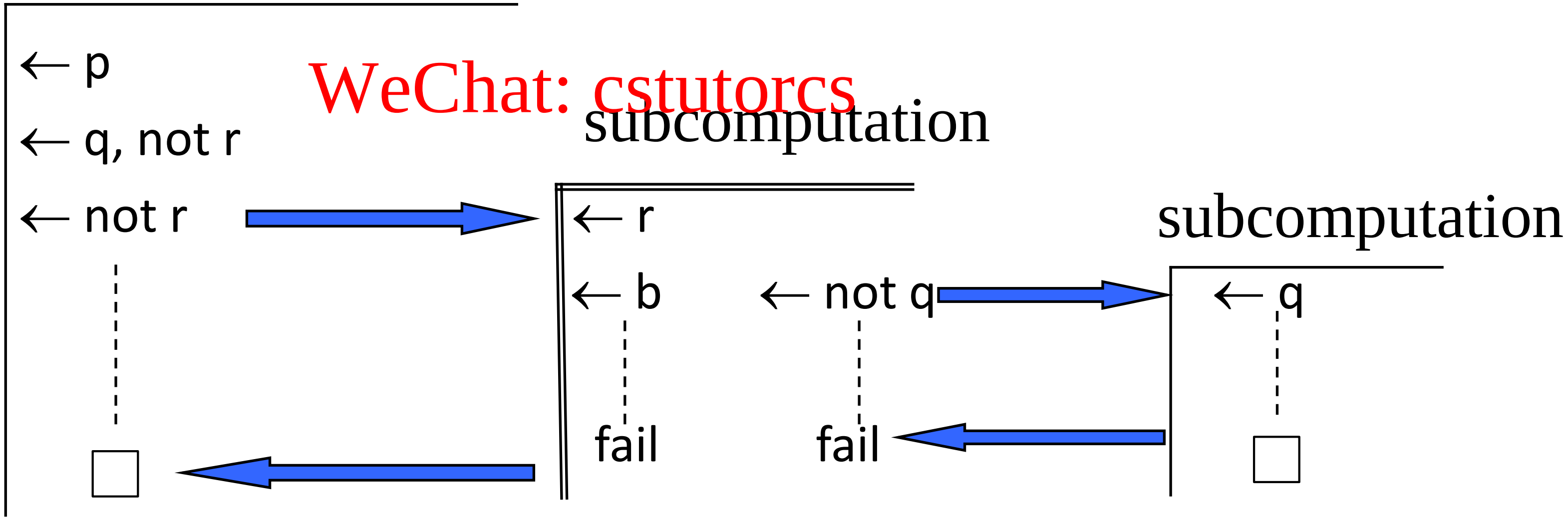
S: p ← q, **not** r

r ← b

r ← **not** q

q ←

Goal: p

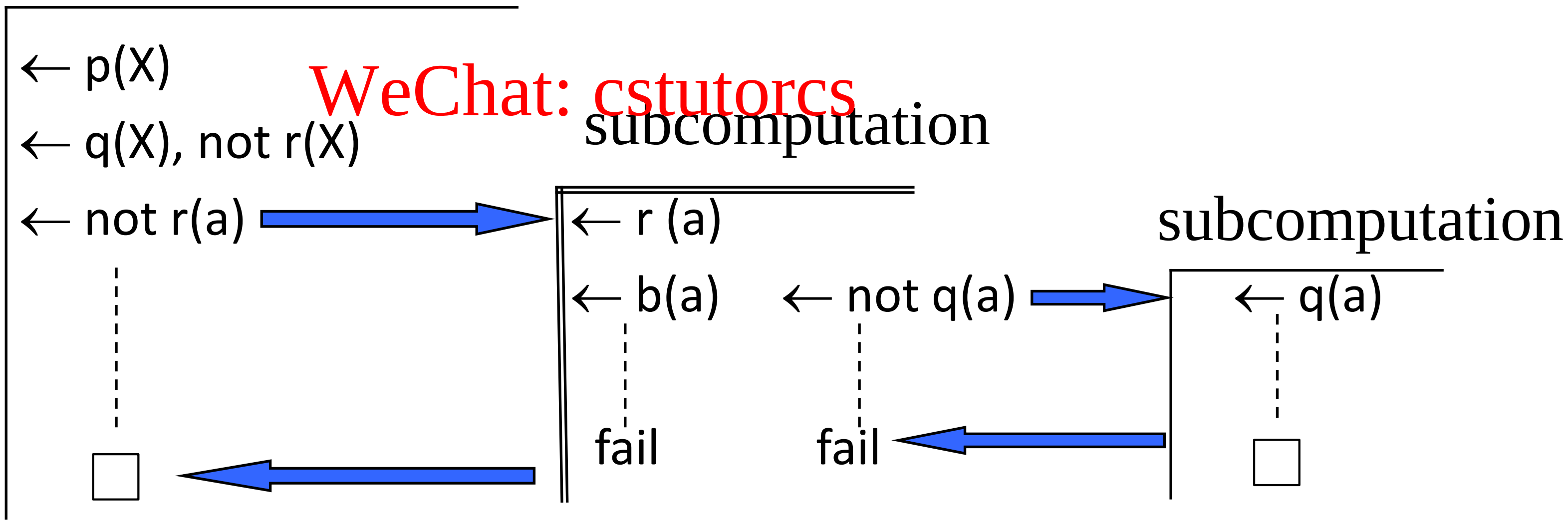


computed answer: {}

SLDNF Example (S is not ground)

S: p(X) ← q(X), **not** r(X)
r(X) ← b(X)
r(X) ← **not** q(X)
q(a) ←

Goal: p



← p(X)

← q(X), not r(X)

← not r(a)

□

subcomputation

← r (a)

← b(a)

fail

← not q(a)

fail

subcomputation

← q(a)

□

computed answer: {X/a}

Selection of sub-goals in SLDNF

- Sub-goals can be selected in any order. The answers are the same, whatever the selection rule
  - Prolog selects always the leftmost sub-goal in the current goal.
- However, the selection of sub-goals must be **safe**: it must not pick a non-ground negative literal
  - Most Prolog systems do not implement this!

E.g. S={p(a)←, q(b)←}

← <u>not p(X)</u>, q(X)	← not p(X), <u>q(X)</u>
…	← not p(b)
fail	…
	□

Wrongly interprets ∃X ¬ p(X) as ¬ ∃X p(X)

SLDNF is non-monotonic

Consider the set of sentences S:

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

$q(a)$

$r(b)$

Then $S \vdash_{SLDNF} p(a)$

but $S \cup \{r(a)\} \nvdash_{SLDNF} p(a)$

Note: SLDNF builds in a kind of Closed World Assumption:

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

Solving the qualification problem using NAF

Example

- 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)

Example

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

innocent(X) ← not guilty(X)

Credulous vs Sceptical non-monotonic 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 needs of our application/problem

The Nixon diamond using NAF

- 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)

SLDNF for the Nixon diamond

By shortening predicates (pacifist becomes p etc)

p(X) ← q(X), not ab_q(X)

ab_q(X) ← n_p(X)

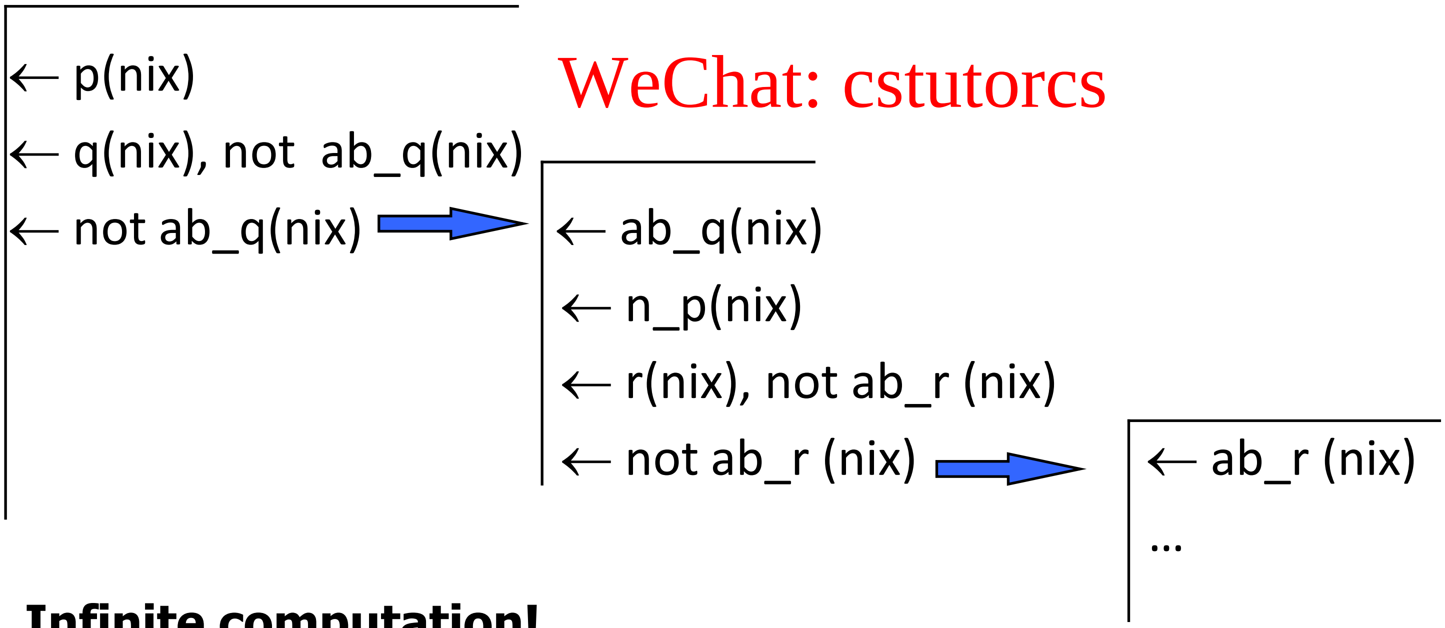
q(nixon)

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!

Semantics for normal logic programs

- Completion semantics
- Well-founded model *
- Stable models (see answer sets in Part II) *

* these semantics equate each **normal logic program** S to the set of all its ground instances over the underlying Herbrand universe

Semantics of sets of definite clauses = positive logic programs (recap)

Given a set of definite clauses S:

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

Note: the LHM is the least fixed point of the (continuous) immediate consequence operator

From positive to normal logic program semantics

Given a normal logic program S:

$\mathbf{T_S}(X)= \{a \in \mathit{HB} \mid a \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_{m+n} \in S,$

$\{b_1, \ldots, b_m\} \subseteq X,$

$\{b_{m+1}, \ldots, b_{m+n}\} \cap X=\{\}\}$

- T_S is no longer guaranteed to be continuous, as it may not be monotonic:
  - E.g. S={p ← not q}

    $\mathbf{T_S}(\{\})=\{p\}$

    $\mathbf{T_S}(\{q\})=\{\}$

Completion semantics (1)

1. Rewrite all rules in S as rules of the form (with $\land$, $\neg$)

$$\text{p}(\vec{X}) \leftarrow \vec{X} = \vec{t}, \text{body}$$

– e.g. $\text{p}(X,3) \leftarrow \text{q}(X), \text{not } \text{r}(3)$

becomes $\text{p}(X,Y) \leftarrow Y=3 \land \text{q}(X) \land \neg\, \text{r}(3)$

2. Combine all rules with the same head

$$\text{p}(\vec{X}) \leftarrow \vec{X} = \vec{t}_1 \land \text{body}_1 \ldots \ \text{p}(\vec{X}) \leftarrow \vec{X} = \vec{t}_n \land \text{body}_n$$

into $(\forall)\text{p}(\vec{X}) \leftrightarrow (\vec{X} = \vec{t}_1 \land \text{body}_1) \lor \ldots \lor (\vec{X} = \vec{t}_n \land \text{body}_n)$

– e.g. $\text{p}(X,Y) \leftarrow Y=3 \land \text{q}(X) \land \neg\, \text{r}(3)$, $\quad \text{p}(X,Y) \leftarrow X=2$

becomes $(\forall)\ \text{p}(X,Y) \leftrightarrow (Y=3 \land \text{q}(X) \land \neg\, \text{r}(3)) \lor (X=2)$

3. If some q(_) (in vocabulary of S) is not the head of any rule in S, add

$$(\forall)\ \text{q}(\vec{X}) \leftrightarrow \text{false}$$

Completion semantics (2)

4. Finally, add Clark's Equality Theory (CET):
   - $f(\vec{t}) \neq g(\vec{s})$ for all pairs of distinct terms in the Herbrand universe of S
   - $(\forall)$ X=X, X=Y→Y=X , X=Y∧Y=Z→X=Z
   - $(\forall)\ \vec{X} = \vec{Y} \rightarrow p(\vec{X}) = p(\vec{Y})$ for all predicate symbols p in S
   - $\vec{X} \neq t[\vec{X}]$ for all terms t where $\vec{X}$ occurs

The resulting logical theory is called the completion of S

Completion: example

S: $p(X) \leftarrow q(X), \textbf{not } r(X)$

$r(X) \leftarrow b(X)$

$r(X) \leftarrow \textbf{not } q(X)$

$q(a) \leftarrow$

Completion of S: CET+

$\forall X\ (p(X) \leftrightarrow q(X) \land \neg\ r(X))$

$\forall X\ \ (r(X) \leftrightarrow b(X) \lor \neg\ q(X))$

$\forall X\ \ (q(X) \leftrightarrow X{=}a)$

$\forall X\ \ (b(X) \leftrightarrow \mathrm{false})$

Soundness of SLDNF (with respect to the completion semantics)

Let Comp(S) be the completion of S

- If there exists an SLDNF-refutation of P from S with answer θ then Comp(S) $\models (\forall)\ P\theta$*

- If there exists a **finitely failed** SLDNF-refutation of P from S then Comp(S) $\models \neg\, \exists P$

Negation as **finite failure**

* Where not is interpreted as ¬

Completion semantics for SLDNF: issues

What if SLDNF does not terminate? What does non-termination “mean”?

- S=$\{p \leftarrow q,\ p \leftarrow \text{not } q,\ q \leftarrow q\}$, P=p

Completion of S = $\{p \leftrightarrow q \vee \neg q,\ q \leftrightarrow q\}$+CET $\models p$

- S=$\{p \leftarrow \text{not } p\}$, P=p

Completion of S = $\{p \leftrightarrow \neg p\}$+CET $\models p, \neg p$

(inconsistent)

Well-founded model semantics

Negation as **infinite failure**

- S={p ← q, p ← not q, q ← q}, P=p

Well-founded model of S is ({p},{not q}):

S “wfm-entails” P

- S={p ← not p}, P=p

Well-founded model of S is ({},{}):

S “does not wfm-entail” P

SLDNF+tabling: XSB Prolog: http://xsb.sourceforge.net/

Well-founded model

(In, Out) such that

- In $\subseteq$ HB
- Out $\subseteq$ HB^{not}={not a | a $\in HB$}

Note:

- atoms in HB- (In ∪ { a | not a ∈ Out}) are "undecided"

- e.g. for S={p $\leftarrow$ not p}, the well-founded model is ({},{}) and p is "undecided"

(definition of well-founded model omitted and thus non examinable)

Nixon diamond using the completion semantics and the well-founded model

p(V) ← q(V), not ab_q(V)
ab_q(V) ← n_p(V)
q(nix)

n_p(V) ← r(V), not ab_r(V)
ab_r (V) ← p(V)
r(nix)

- Completion= CET+

p(V) ↔ q(V) ∧ ¬ ab_q(V)
ab_q(V) ↔ n_p(V)
q(V) ↔ V=nix

n_p(V) ↔ r(V) ∧ ¬ ab_r(V)
ab_r (V) ↔ p(V)
r(V) ↔ V=nix

  - p(nix), n_p(nix) hold in different models
- Well-founded model =({q(nix), r(nix)},{})
  - p(nix), n_p(nix) are both undecided

Stable models

- Given a (ground) normal logic program S and $X \subseteq HB$, the *reduct of S by X* (referred to as S^X) is obtained in two steps:
    1. Eliminate all rules with not p in the body, for every $p \in X$
    2. Eliminate all negative literals from the body of all remaining rules

  (S^X is a set of definite clauses)

- $X \subseteq HB$ is a *stable model* of S iff the LHM of S^X is X

Several efficient answer set solvers exist – see Part II

Nixon diamond using stable models (1)

p(V) ← q(V), not ab_q(V)

ab_q(V) ← n_p(V)

q(nix)

n_p(V) ← r(V), not ab_r (V)

ab_r (V) ← p(V)

r(nix)

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

Consider X={p(nix), ab_r(nix), q(nix), r(nix)}. **Is X a stable model? YES**.

To see this, let us construct S^X

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 S^X is {q(nix), r(nix), p(nix), ab_r (nix) }=X, as required.

Nixon diamond using stable models (2)

X={q(nix), r(nix), n_p(nix), ab_q(nix)} is also a stable model (Exercise: verify this)

- Multiple “extensions” = stable models
- Credulous (whatever holds in some stable model) vs sceptical (whatever holds in all stable models)

Summary

- Classical logic: the qualification problem
- Closed World Assumption
- (non-)Monotonicity
- Non-monotonic/defeasible reasoning
- Negation-as-failure/SLDNF for non-monotonic reasoning
- Completion semantics, well-founded semantics (just a mention) and stable model semantics