Knowledge Representation
Chapter 2. Propositional Representation and Reasoning

Pedro Cabalar

Dept. Computer Science
University of Corunna, SPAIN

February 8, 2018

1. Propositional Logic: Syntax and Semantics
2. Propositional Reasoning
3. Default Negation: Stable Models

Propositional Logic: Syntax

- Def. Propositional Signature Σ: set of propositions or atoms. E.g. $\Sigma = \{happy, rain, weekend\}$.
- Def. Propositional language $\mathcal{L}_\Sigma$, set of well formed formulae (wff). Any of the expressions:

i)	$\top$	vi)	$\alpha \vee \beta$
ii)	$\bot$	vii)	$\alpha \wedge \beta$
iii)	p	viii)	$\alpha \rightarrow \beta$
iv)	$\neg\alpha$	ix)	$\alpha \leftrightarrow \beta$
v)	(α)		

  with $p \in \Sigma$ and $\alpha, \beta \in \mathcal{L}_\Sigma$. Precedence: $\equiv, \rightarrow, \vee, \wedge, \neg$ (left associative). Alternative notations:
  implication $\rightarrow$, $\supset$, $\Rightarrow$; equivalence $\equiv$, $=$, $\leftrightarrow$, $\Leftrightarrow$
- Def. literal: is an atom p or its negation $\neg p$.
- Def. theory: is a set of formulae $\Gamma \subseteq \mathcal{L}_\Sigma$.

Propositional Logic: Semantics

- Def. interpretation is a function $\mathcal{I} : \Sigma \longrightarrow \{1, 0\}$
  We can also use $\mathcal{I} \subseteq \Sigma$ so that, for instance,
  $I = \{happy, weekend\}$
  means $\mathcal{I}(happy) = 1, \ \ \mathcal{I}(rain) = 0, \ \ \mathcal{I}(weekend) = 0,$ etc.
- We extend its use $\mathcal{I} : \mathcal{L}_\Sigma \longrightarrow \{1, 0\}$.
  $\mathcal{I}(\alpha) =$ replace each $p \in \Sigma$ in α by $\mathcal{I}(p)$ and apply:

$$\begin{array}{rcl} \mathcal{I}(\top) & = & 1 \\ \mathcal{I}(\bot) & = & 0 \end{array}$$

	$\neg$
1	0
0	1

		$\wedge$	$\vee$	$\rightarrow$	$\leftrightarrow$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Propositional Logic: Semantics

- Def. $\mathcal{I}$ satisfies α, written $\mathcal{I} \models \alpha$, iff $\mathcal{I}(\alpha) = 1$.
  - i) $\mathcal{I} \models T$ and $\mathcal{I} \not\models F$.
  - ii) $\mathcal{I} \models p$ iff $\mathcal{I}(p) = 1$.
  - iii) $\mathcal{I} \models \neg\alpha$ iff $\mathcal{I} \not\models \alpha$.
  - iv) $\mathcal{I} \models \alpha \wedge \beta$ iff $\mathcal{I} \models \alpha$ and $\mathcal{I} \models \beta$.
  - v) $\mathcal{I} \models \alpha \vee \beta$ iff $\mathcal{I} \models \alpha$ or $\mathcal{I} \models \beta$ (or both).
  - vi) $\mathcal{I} \models \alpha \equiv \beta$ iff ($\mathcal{I} \models \alpha$ iff $\mathcal{I} \models \beta$).
- $\mathcal{I}$ is a *model* of Γ, written $\mathcal{I} \models \Gamma$, iff it satisfies all its formulae.
- Def. inconsistency or unsatisfiable formula: a formula that has no models.
  Def. α is a tautology or is valid iff any interpretation is a model of α. Examples: $\top$, $p \vee \neg p$, $b \wedge c \wedge d \rightarrow (d \rightarrow b)$
- α is a logical consequence of or is entailed by Γ, written $\Gamma \models \alpha$, iff any model of Γ satisfies α. Therefore, when $\Gamma = \emptyset$, what does $\models \alpha$ mean?

Propositional Logic: Semantics

- Alternative viewpoint: think about the set of models of a given formula α, call it $M(\alpha)$.

  $M(\bot) = \emptyset$
  $M(\top) = S$ (all the possible ones)
  $M(a \vee b) = \{\{a, b\}, \{a\}, \{b\}\}$

  We will usually identify α with $M(\alpha)$ and vice versa.

- From a set S of interpretations: How can we get a formula α s.t. $M(\alpha) = S$? Does this formula α always exist?
- How many theories (modulo equivalence) can we build with n atoms?

Propositional Logic: Semantics

Definition (Weaker/stronger formula)

If $\models \alpha \rightarrow \beta$, that is,
if $M(\alpha) \subseteq M(\beta)$, then
α is stronger than β (or β is weaker α).

- Which are the strongest and weakest possible formulae?
- Examples: for each pair, which is the strongest?

$$\begin{array}{ccc}
p & \leftarrow & p \wedge q \\
p & \rightarrow & p \vee \neg q \\
p \vee q & \leftarrow & p \wedge q \\
p & \rightarrow & (q \rightarrow p) \\
p \wedge \neg q & & \neg p \wedge q
\end{array}$$

From human to formal language . . .

$A \rightarrow B$	A *implies* B
	A is a *sufficient condition* for B
	B is a *necessary condition* for A
	if A then B
	B if A
	A only if B
	no A unless B
	B given that A
	B provided that A
$A \leftrightarrow B$	A is *equivalent* to B
	A if and only if (iff) B
$A \vee B$	A or B (inclusive or)
$\neg(A \leftrightarrow B)$	A or B (exclusive or)

Del lenguaje humano al formal ...

$A \rightarrow B$	A *implica* B
	A es *suficiente* para B
	B es *necesario* para A
	si A entonces B
	B si A
	A sólo si B
	no A a no ser que B
	no A a menos que B
	B siempre que A
$A \leftrightarrow B$	A *equivale* a B
	A si y sólo si B
$A \vee B$	A ó B (inclusivo)
$\neg(A \leftrightarrow B)$	A ó B (exclusivo)

Using propositional logic: an example

- From Lady or the Tiger? And Other Logic Puzzles Including a Mathematical Novel That Features Godel's Great Discovery (Raymond M. Smullyan)
- Behind each door we may have either a lady or a tiger. One sign tells the truth but the other one no. Which door shall we open?

1. A lady is in this room and a tiger is in the other	2. There is a lady in one room and a tiger in the other

Another variation

- There are one lady and two tigers. At most, one sign tells the truth. Which door shall we open?

1. There is a tiger in this room	2. There is a lady in this room	3. There is a tiger in room 2
		

1 Propositional Logic: Syntax and Semantics

2 Propositional Reasoning

3 Default Negation: Stable Models

Propositional Reasoning

Reasoning: $\{P_1, \ldots, P_n\} \models C$

does conclusion C follow from premises $\{P_1, \ldots, P_n\} = KB$ (the Knowledge Base)?

Example: $KB =$

P_1: On *w*eekends, I don't watch *tv* $(w \rightarrow \neg tv)$

P_2: I'm *h*appy when it *r*ains, excepting in the *w*eekend $(r \wedge \neg w \rightarrow h)$

P_3: I'm watching *tv* but I'm not *h*appy $(tv \wedge \neg h)$

Can I conclude this?

C: it is not *r*aining $(\neg r)$

Propositional Reasoning

Definition (Entailment)

A theory KB entails conclusion C, written $KB \models C$, when all models of KB are models of C. If so, C is called a semantic consequence of KB.

- In propositional logic, $\{P_1, P_2, P_3\} \models C$ is the same as checking that the formula $P_1 \wedge P_2 \wedge P_3 \rightarrow C$ is a tautology or, equivalently, that its negation $P_1 \wedge P_2 \wedge P_3 \wedge \neg C$ is inconsistent

Definition (SAT decision problem)

Decision problem $SAT(\alpha) \in \{yes, no\}$ checks whether a formula α has some model. (Time) complexity: **NP**-complete problem.

- In other words:
  $\{P_1, P_2, P_3\} \models C$ iff $SAT(P1 \wedge P2 \wedge P3 \wedge \neg C) = no$.

Propositional Reasoning

- First naive method: check all interpretations ($2^4 = 16$) one by one (truth table) to obtain a 0 in all cases.
- $\mathcal{I}(P_1 \wedge P_2 \wedge P_3 \wedge \neg C) = 0$ when some conjunct is 0.

h	tv	w	r	P_1 $(w \rightarrow \neg tv)$	P_2 $(r \wedge \neg w \rightarrow h)$	P_3 $tv \wedge \neg h$	$\neg C$ r
0	0	0	0	1	1	0	0
		$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	1	0	0	1	1	1	0
0	1	0	1	1	0	1	1
0	1	1	0	0	1	1	0
0	1	1	1	0	1	1	1
		$\vdots$					

Exercise

- The satisfaction relation `I` $\models$ `F` as `sat(I,F)` in Prolog:

```
:- op(210, yfx, &).
:- op(220, yfx, v).
:- op(1060, yfx, <->).
sat(_I, false) :- !, fail.
sat(_I, true) :- !.
sat(I, P) :- atom(P),!,member(P,I),!.
sat(I, -A) :- \+ sat(I, A).
sat(I, A & B) :- sat(I, A), sat(I, B).
sat(I, A v B) :- sat(I,A),! ; sat(I, B).
sat(I, A -> B) :- sat(I,-A v B).
sat(I, A <-> B) :- sat(I,(A -> B)&(B -> A)).
```

Exercise

- Testing whether a formula `F` for signature `S` is inconsistent:

```
inconsistent(S, F) :- \+ (subset(S,I), sat(I,F)).

subset([],[]) :- !.
subset([X | Xs],S) :- subset(Xs,S).
subset([X | Xs],[X | S]) :- subset(Xs,S).
```

- Generation of interpretations (`subset`) = exponential blow up

Propositional Reasoning

- Computational cost is exponential = 2^n with $n = |\Sigma|$ number of atoms. Can we perform better?
- Not much hope for the worst case: **NP**-complete!
- However, enumeration of interpretations always forces worst case. We can do better in particular cases.
- In our example: formulas $tv \wedge \neg h$ and r fix the truth of 3 atoms: $\mathcal{I}(h) = 0$, $\mathcal{I}(tv) = 1$ and $\mathcal{I}(r) = 1$.

$$\begin{array}{rcl}
(w \rightarrow \neg tv) & \wedge & (r \wedge \neg w \rightarrow h) \\
(w \rightarrow \neg \top) & \wedge & (\top \wedge \neg w \rightarrow \bot) \\
(w \rightarrow \bot) & \wedge & (\neg w \rightarrow \bot) \\
\neg w & \wedge & w \quad \text{inconsistent!}
\end{array}$$

Resolution

- Note that $P_1 : (w \rightarrow \neg tv)$ is equivalent to $\neg w \vee \neg tv$ that, together with tv from P_3 allows us to conclude $\neg w$.
- Trying to generalize: represent the KB as disjunctions of literals (clauses). This is called Conjunctive Normal Form (CNF) = conjunction of (disjunctive) clauses.
- Example:

$$
\begin{array}{ccccc}
P_1 & \wedge & P_2 & \wedge & P \\
(w \rightarrow \neg tv) & \wedge & (r \wedge \neg w \rightarrow h) & \wedge & tv \wedge \\
(\neg w \vee \neg tv)\,\underbrace{(\neg w \vee \neg tv)}_{C_1} & \wedge & (\neg r \vee w \vee h)\,\underbrace{(\neg r \vee w \vee h)}_{C_2} & \wedge & tv\ \underbrace{tv}_{C_3} \wedge
\end{array}
$$

we get five clauses: C_3, C_4, C_5 are unit clauses.

Resolution

- Any formula can be transformed into CNF:
  1. replace $\alpha \rightarrow \beta$ by $\neg\alpha \vee \beta$
     replace $\alpha \leftrightarrow \beta$ by $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$
  2. move negation until only applied to atoms using De Morgan laws (Negation Normal Form, NNF)
  3. apply distributivity $\wedge, \vee$

- Warning: transformation into CNF may have an exponential cost.. Example $(a \wedge b) \vee (c \wedge d) \vee (e \wedge f) \vee (h \wedge i)$

- Some techniques [Tseitin68] allow generating a CNF in polynomial time but introducing new auxiliary atoms.

- If KB is a set of facts and implications involving literals, it is already in CNF!

Exercise: NNF in Prolog

- Example: we first describe how to get the NNF

```
:- op(210, yfx, &).
:- op(220, yfx, v).

nnf(-(A & B),NA v NB):- !,nnf(-A,NA),nnf(-B,NB).
nnf(-(A v B),NA & NB):- !,nnf(-A,NA),nnf(-B,NB).
nnf(-(A -> B), A1 & NB):- !,nnf(A,A1),nnf(-B,NB).
nnf(- - A,A1):-!,nnf(A,A1).
nnf(A,A).
```

Exercise: CNF in Prolog

. . . and then the CNF

```
cnf(A -> B,Zs):- !, cnf(-A v B,Zs).
cnf(A & B,Zs):- !, cnf(A,Xs),cnf(B,Ys),
                merge_set(Xs,Ys,Zs).
cnf(A v B,Zs):- !, cnf(A,Xs),cnf(B,Ys),
               findall(Z,
                  (member(X,Xs),member(Y,Ys),
                   merge_set(X,Y,Z)
                  ),
               Zs).
cnf(A,[ [A] ]).
```

Resolution

- The resolution rule [Davis & Putnam 1960] is defined as:

$$\frac{(\alpha \vee p) \qquad\qquad (\neg p \vee \beta)}{(\alpha \vee \beta)}$$

  The resulting clause $\alpha \vee \beta$ is called the resolvent. An empty disjunction (or empty clause) corresponds to $\bot$.

- Main result [Robinson65]: α is unsatisfiable iff there exists a derivation of $\bot$ from $CNF[\alpha]$ applying resolution.

In our example

$$C_1 : \neg w \vee \neg tv \qquad C_2 : \neg r \vee w \vee h$$

$$C_6 : \neg tv \vee \neg r \vee h \qquad C_3 : tv$$

$$C_7 : \neg r \vee h \qquad C_4 : \neg h$$

$$C_8 : \neg r \qquad C_5 : r$$

$$C_9 : \bot$$

Resolution

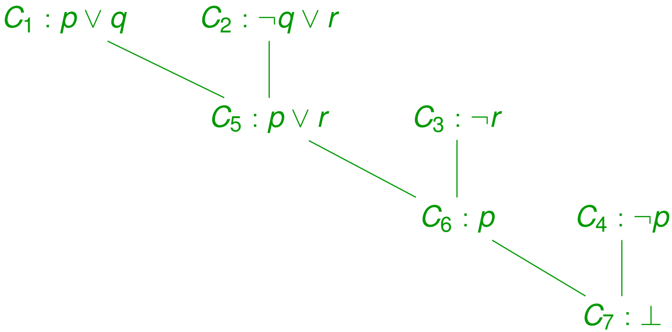
- Another example:
  prove that $\alpha : (\neg p \rightarrow q) \wedge (q \rightarrow r) \rightarrow (\neg r \rightarrow p)$ is valid.

- We first negate the formula and get its CNF

$$
\begin{aligned}
& \neg((\neg p \rightarrow q) \wedge (q \rightarrow r) \rightarrow (\neg r \rightarrow p)) \\
\equiv \quad & \neg(\neg((\neg\neg p \vee q) \wedge (\neg q \vee r)) \vee (\neg\neg r \vee p)) \\
\equiv \quad & ((p \vee q) \wedge (\neg q \vee r)) \wedge \neg(r \vee p) \\
\equiv \quad & \underbrace{(p \vee q)}_{C_1} \wedge \underbrace{(\neg q \vee r)}_{C_2} \wedge \underbrace{\neg r}_{C_3} \wedge \underbrace{\neg p}_{C_4}
\end{aligned}
$$

Resolution

- A possible application of resolution ...



SAT solvers

- Problem: generating all possible resolvents is unfeasible.
  Important: heuristics to explore only a part of the search tree.
- SAT solvers: nowadays, SAT is an outstanding state-of-the-art research area for search algorithms. There exist many efficient tools and commercial applications. See `www.satlive.com`
- SAT keypoint: instead of designing an *ad hoc* search algorithm, encode the problem into propositional logic and use SAT as a backend.

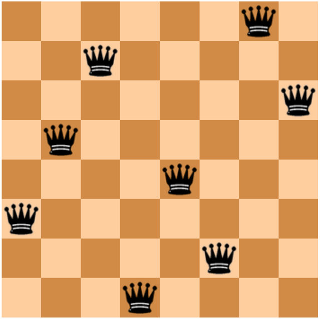
SAT solvers

- DIMACS: standard input for SAT solvers
- Line starting with `p` counts number of atoms (4) and clauses (5)
- 0 denotes end of clause. Minus=negation.
- Example: atoms $1 = h$, $2 = r$, $3 = tv$, $4 = w$

```
c happy.cnf
c This is a comment line
p cnf 4 5
-4 -3 0
-2 4 1 0
3 0
-1 0
4 0
```

- Download `clasp` solver and execute `clasp happy.cnf`
  `https://potassco.org/clasp/`

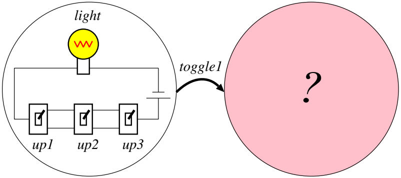
An exercise



Example (8-queens problem)

- Arrange 8 queens in a 8×8 chessboard so they do not attack one each other.
- Encode the problem as a propositional theory and use `clasp` to find a solution.

Lamp example revisited

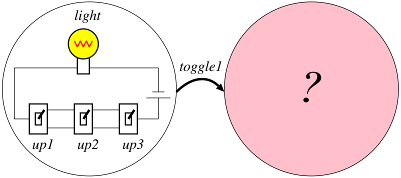


Exercise: use `clasp` to check that the theory

$$toggle(x) \wedge up(x)_0 \rightarrow \neg up(x)_1 \qquad toggle(x) \wedge light_0 \rightarrow \neg light_1$$
$$toggle(x) \wedge \neg up(x) \rightarrow up(x)_1 \qquad toggle(x) \wedge \neg light_0 \rightarrow light_1$$
$$up(1)_0 \wedge up(2)_0 \wedge up(3)_0 \wedge light_0 \wedge toggle(1)$$

for $x \in \{1, 2, 3\}$, allows concluding $\neg light_1 \wedge \neg up(1)_1$ but not concluding $up(2)_1 \wedge up(3)_1$. Frame problem!

Lamp example revisited



We require adding frame axioms:

$$toggle(x) \wedge up(y)_0 \rightarrow up(y)_1$$
$$toggle(x) \wedge \neg up(y)_0 \rightarrow \neg up(y)_1$$

for any pair $x \in \{1, 2, 3\}$ and $y \in \{1, 2, 3\}$ with $x \neq y$.

Resolution: Horn clauses

- Horn clause: it contains at most one positive literal.
- It's the basis for Prolog and Logic Programming. We can see each Horn clause as a rule:

$$
\begin{aligned}
& p \vee \neg q_1 \vee \neg q_2 \vee \neg q_3 \\
\equiv \quad & p \leftarrow q_1 \wedge q_2 \wedge q_3 \\
\equiv \quad & p \,\texttt{:-}\, q_1, q_2, q_3. \quad \text{(a rule)}
\end{aligned}
$$

$$
p \quad \text{(a fact)}
$$

$$
\begin{aligned}
& \neg q_1 \vee \neg q_2 \vee \neg q_3 \\
\equiv \quad & \bot \leftarrow q_1 \wedge q_2 \wedge q_3 \quad \text{(a constraint)} \\
\equiv \quad & \texttt{?-}\, q_1, q_2, q_3. \quad \text{(or the goal in Prolog)}
\end{aligned}
$$

Resolution: Horn clauses

- If $\mathit{CNF}(\alpha)$ exclusively contains Horn clauses, deciding its satisfiability (HORNSAT) is **P**-complete.

Positive Logic Programs

- A positive logic program is a set of implications of the form

$$\underbrace{p}_{\text{head}} \leftarrow \underbrace{q_1, \ldots, q_n}_{\text{body}}$$

  or, written in text format

```
p :- q1, ..., qn.
```

  with $n \geq 0$, where $p, q_1, \ldots, q_n$ are atoms.
  Commas in the body represent conjunctions.

- Note: these are non-negative Horn clauses. Unlike Prolog, ordering among rules or in the body is irrelevant.

- When $n = 0$, the rule is called a fact, and we usually omit the $\leftarrow$.

Positive Logic Programs

- Positive programs can be easily computed by "rule application" (deductive closure).
- Given a program P, and a propositional interpretation $\mathcal{I}$ we define the direct consequences [van Endem & Kowalski 76] operator $T_P(\mathcal{I})$ as:

$$T_P(\mathcal{I}) := \{H \mid (H \leftarrow B) \in P \text{ and } \mathcal{I} \models B\}$$

  That is, pick those rule heads H with body B satisfied by $\mathcal{I}$

- Example: given P below, $T_P(\{b, p, s\}) = \{p, q, r, a\}$

$$\begin{array}{lll} p & & \\ q & & \\ r & \leftarrow & p, s \end{array} \qquad \begin{array}{lll} s & \leftarrow & q \\ a & \leftarrow & b, p \end{array} \qquad \begin{array}{lll} b & \leftarrow & s, a \\ a & \leftarrow & c \end{array}$$

Positive Logic Programs

- Exercise: prove that T_P is $\subseteq$-monotonic, i.e., if $\mathcal{I} \subseteq J$, then $T_P(\mathcal{I}) \subseteq T_P(J)$.
- By Knaster & Tarski's theorem, T_P has a $\subseteq$-least fix point $\mathcal{I} = T_P(\mathcal{I})$.
- Moreover, T_P is continuous and the l.f.p. can be computed by iteration of T_P on $\mathcal{I}_0 = \emptyset$ until reaching a point $\mathcal{I}_{i+1} = T_P(\mathcal{I}_i) = \mathcal{I}_i$.
- Back to the example

$$\begin{array}{rcl} p & & \\ q & & \\ r & \leftarrow & p, s \end{array} \qquad \begin{array}{rcl} s & \leftarrow & q \\ a & \leftarrow & b, p \end{array} \qquad \begin{array}{rcl} b & \leftarrow & s, a \\ a & \leftarrow & c \end{array}$$

$T_P(\emptyset) = \{p, q\}$, $T_P(\{p, q\}) = \{p, q, s\}$, $T_P(\{p, q, s\}) = \{p, q, s, r\}$, $T_P(\{p, q, s, r\}) = \{p, q, s, r\}$ fixpoint.

Positive Logic Programs

- Main result by [van Endem & Kowalski 76]: P has a least propositional model $LM(P)$ that coincides with T_P least fixpoint.
- In our example:

$$
\begin{array}{lcl}
p & & \\
q & & \\
r & \leftarrow & p, s
\end{array}
\qquad
\begin{array}{lcl}
s & \leftarrow & q \\
a & \leftarrow & b, p
\end{array}
\qquad
\begin{array}{lcl}
b & \leftarrow & s, a \\
a & \leftarrow & c
\end{array}
$$

  the models of P are $\{p, q, r, s\}$, $\{p, q, r, s, a, b\}$, $\{p, q, r, s, a, b, c\}$.
- Exercise: prove it.

1 Propositional Logic: Syntax and Semantics

2 Propositional Reasoning

3 Default Negation: Stable Models

Default Negation

- A normal logic program is a set of rules of the form:

$$\underbrace{p}_{\text{head}} \leftarrow \underbrace{q_1, \ldots, q_m, not\ q_{m+1}, \ldots, not\ q_n}_{\text{body}}. \tag{1}$$

  with $n \geq m \geq 0$.

- Again, the ordering among rules or body literals is irrelevant.
- Note that when $m = n$ (no negations) we have a positive rule.
- Default negation: $not\ q_i$ = "there is no way to derive atom q_i."
- Ex.: we serve meat, unless we know that the client is vegetarian.

$$meat \leftarrow not\ veg$$

Default Negation

- Minimal truth doesn't work! we may have several minimal models. Some of them are undesired.
- The formula $\neg \mathit{veg} \rightarrow \mathit{meat}$ is equivalent to the formulas:

$$\mathit{meat} \vee \mathit{veg}$$
$$\neg \mathit{meat} \rightarrow \mathit{veg}$$

  and has 2 minimal models $\mathcal{I}_1 = \{\mathit{meat}\}$ and $\{\mathit{veg}\}$.
- However, the rule should be directional: we can derive *meat* from *not veg* but not vice versa. The only expected model is $\{\mathit{meat}\}$:
  1. No rule can derive *veg* (it is not in any head)
  2. So, *veg* is false by default, and the rule yields *meat*

Default Negation

- When the program has no cyclic dependences (stratified) through negation, it is easy to proceed.

$$\text{Layer 1} \left\{ \begin{array}{lcl} a & & \\ b & \leftarrow & a \end{array} \right. \qquad \{a, b\}$$

$$\text{Layer 2} \left\{ \begin{array}{lcl} c & \leftarrow & not\ a\, \underbrace{not\ a}_{\perp} \\ & & \end{array} \right. \qquad \{a, b\}$$

$$\text{Layer 3} \left\{ \begin{array}{lcl} d & \leftarrow & b, not\ c\, \underbrace{not\ c}_{\top} \\ & & \end{array} \right. \qquad \{a, b, d\}$$

- But how can we deal with negative cycles?

$$\begin{array}{rcl} meat & \leftarrow & \neg veg \\ veg & \leftarrow & \neg meat \end{array}$$

Adding negation: stable models

- Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In ICLP'88, 1070-1080.

Definition (program reduct)

We define the reduct of a program P with respect to an interpretation (set of atoms) $\mathcal{I}$, written $P^{\mathcal{I}}$, as the set of rules:

$$P^{\mathcal{I}} \stackrel{def}{=} \{ \ (p \leftarrow q_1, \ldots, q_m) \ | \ (p \leftarrow q_1, \ldots, q_m, not\ q_{m+1}, \ldots, not\ q_n) \in P \text{ and } q_j \notin \mathcal{I}, \text{ for all } j = m+1, \ldots, n \ \}$$

Stable models

- Observation: $P^{\mathcal{I}}$ is a positive program (it contains no negations), so it has a least model, call it $\Gamma_P(\mathcal{I}) \stackrel{def}{=} LM(P^{\mathcal{I}})$.

Definition (stable model)

An interpretation $\mathcal{I}$ is a stable model of a program P iff
$\boxed{\mathcal{I} = \Gamma_P(\mathcal{I}) = LM(P^{\mathcal{I}})}$. □

Stable models: some properties

Proposition (Stable models are models)

If $\mathcal{I}$ is a stable model of P then $\mathcal{I} \models P$.

Proposition (Stable models are minimal models)

If $\mathcal{I}$ is a stable model of P then there is no $J \subset \mathcal{I}$ such that $J \models P$.

Exercise: prove the above theorems.

Stable models

- Example 1: “Birds normally fly”

$$flies \leftarrow bird, not\ ab \qquad\qquad bird$$

- This program has these three models:

$\mathcal{I}$	$P^{\mathcal{I}}$	$LM(P^{\mathcal{I}})$
$\{bird, ab\}$	$bird$	$\{bird\} \neq \mathcal{I}$ not stable
$\{bird, ab, flies\}$	$bird$	$\{bird\} \neq \mathcal{I}$ not stable
$\{bird, flies\}$	$flies \leftarrow bird$ $bird$	$\{bird, flies\}$ stable!

Stable models

- Example 2: we add "Penguins are exceptions"

$$\begin{array}{rcl} flies & \leftarrow & bird, not\ ab \\ ab & \leftarrow & bird, penguin \end{array} \qquad \begin{array}{r} bird \\ penguin \end{array}$$

- Just two (classical) models now:

$\mathcal{I}$	$P^{\mathcal{I}}$	$LM(P^{\mathcal{I}})$
$\{bird,$ $penguin,$ $ab\}$	$bird$ $ab \leftarrow bird, penguin$ $penguin$	$\{bird,$ $penguin,$ $ab\}$ *stable*!
$\{bird,$ $penguin,$ $ab, flies\}$	$bird$ $ab \leftarrow bird, penguin$ $penguin$	$\{bird,$ $penguin,$ $ab\} \neq I$ *notstable*

Observation: the example shows non-monotonic reasoning!

- Example 1: “Birds normally fly”, stable model $\{bird, flies\}$ allowed us to conclude *flies*
- Example 2: adding new formulas “Penguins are exceptions” stable model $\{bird, penguin, ab\}$ retracts previous conclusion (*flies* is not true any more)

Stable models: some properties

- A program may have several stable models. For instance, P_1:

$$meat \leftarrow not\ veg \qquad veg \leftarrow not\ meat \tag{2}$$

has two $\{meat\}$ and $\{veg\}$.

Definition ($SM(P)$)

We denote the set of stable models of program P as $SM(P)$.

- A program may have no stable model at all, $SM(P) = \emptyset$. Example:

$$p \leftarrow not\ p \tag{3}$$

- Typical use: even cycles (2) generate multiple solutions; odd cycles (3) prune undesired models (odd cycles like P_2).
- Constraints: to avoid a model where p holds but q doesn't:

$$aux \leftarrow p, not\ q, not\ aux$$

Stable models: some properties

Proposition

Deciding whether a program P has a stable model, $SM(P) \stackrel{?}{=} \emptyset$, is an **NP***-complete problem.*

- That is, same complexity class as SAT.

Using clingo

- Download `clingo` from: http://potassco.org/
- Create the following program in a text file `bird.txt`:

```
flies :- bird, not ab.
ab :- bird, penguin.
{bird}.
{penguin}.
```

- `{...}` is a non-deterministic choice: you can choose to include the fact, or not.
- To compute all the stable models, type

  `clingo -n 0 bird.txt`
- The guide can be downloaded from: https://sourceforge.net/projects/potassco/files/guide/

Beyond normal programs

- General programs: a rule has the form

$$H_1, \ldots, H_n \leftarrow B_1, \ldots, B_m$$

  where $n \geq 0$, $m \geq 0$ and H_i and B_j are literals.
  Commas in the head correspond to disjunctions $\vee$.
  When $m = 0$ (empty body) we omit the arrow $\leftarrow$.

- Example: when not *b*usy, I go to the *c*inema or watch *tv*

$$c, tv \leftarrow not\ b$$

  Sometimes I'm *b*usy, sometimes no

$$b, not\ b$$

Beyond normal programs

Definition (program reduct)

We define the reduct of a program P with respect to an interpretation (set of atoms) $\mathcal{I}$, written $P^{\mathcal{I}}$, as the set of rules:

$$
\begin{aligned}
P^{\mathcal{I}} \stackrel{def}{=} \ & \{(p_1, \ldots, p_n \leftarrow q_1, \ldots, q_m) \mid \\
& (p_1, \ldots, p_n, not\ a_1, \ldots, not\ a_k \leftarrow q_1, \ldots, q_m, not\ b_1, \ldots, not\ b_h) \\
& \in P \text{ and all } a_i \in \mathcal{I} \text{ and all } b_j \notin \mathcal{I}\ \}
\end{aligned}
$$

Definition (stable model)

$\mathcal{I}$ is a stable model of P iff it is a minimal model of $P^{\mathcal{I}}$.

Beyond normal programs

Proposition

Stable models of general programs are classical models. For any general program P: $SM(P) \subseteq M(P)$.

- Example

$$c, tv \leftarrow not\ b \qquad\qquad b, not\ b$$

Classical model $\mathcal{I}$	$T_P(\mathcal{I})$	minimal models
$\{b\}$	b	$\{b\}$ stable
$\{c\}$	(c, tv)	$\{c\}\{tv\}$ stable
$\{tv\}$	(c, tv)	$\{c\}\{tv\}$ stable
$\{b, c\}$	b	$\{b\}$
$\{b, tv\}$	b	$\{b\}$
$\{c, tv\}$	(c, tv)	$\{c\}\{tv\}$
$\{b, c, tv\}$	b	$\{b\}$

Beyond normal programs

- Stable models of general programs may not be minimal any more

$$b, \textit{not } b$$

  has two stable models $\emptyset$ and $\{b\}$.

- This is can be abbreviated as a cardinality constraint:

$$0\{b\}1 \qquad \text{or simply} \quad \{b\}$$

Proposition

Deciding $SM(P) \stackrel{?}{=} \emptyset$ for a general program P is a $\mathbf{NP}^{\mathbf{NP}}$*-complete (a.k.a.* $\Sigma_2^{\mathbf{P}}$*-complete) problem.*

$\mathbf{NP}^{\mathbf{NP}}$: means **NP** on a Turing machine with an **NP** oracle. This is (conjectured) harder than **NP**.

Beyond normal programs

- Cardinalitity constraints: try these examples in clingo

```
% pick from 2 to 4 extra ingredients
2 {tuna; pepper; cheese; olives; ham} 4.
```

```
% I can choose group or not
{group}.
% If group, I pick at least 2 seats
2 {s1;s2;s3;s4} :- group.
% If not group, I pick at most 3 seats
{s1;s2;s3;s4} 3 :- not group.
% if I picked 4 then full
full :- 4 {s1;s2;s3;s4} 4.
```