LOGIC PROGRAMMING

Well-Founded Semantics

Properties of SMs

- Stable models are minimal models
- Stable models are supported

Importance of Stable Models

- Stable Models were an important contribution:
  - Introduced the notion of *default negation* (versus negation *as failure*)
  - Allowed important connections to NMR. Started the area of LP&NMR
  - Allowed for a better understanding of the use of LPs in Knowledge Representation
- It is considered as THE semantics of LPs by a significant part of the community.
- However…

Relevance

- A directly depends on B if B occurs in the body of some rule with head A. A depends on B if A directly depends on B or there is a C such that A directly depends on C and C depends on B.
- A semantics Sem is relevant iff for every program P, $A \in Sem(P)$ iff $A \in Sem(Rel_A(P))$
  - where Rel_A (P) contains all rules of P whose head is A or some B on which A depends.
- This property is required to allow for the usual top-down execution of logic programs.

Cumulativity

- A semantics Sem is cumulative iff for every program P, if $A \in Sem(P)$ and $B \in Sem(P)$ then $B \in Sem(P \cup \{A\})$
  - i.e. all derived atoms can be added as facts without changing the program's meaning.
- This property is very important for implementations.
  - Without it, tabling methods cannot be used.

Problems with Stable Models

- The stable models semantics does not assign meaning to every program
  - E.g. program $\{a \leftarrow \text{not } a\}$ has no stable models.
- The stable models semantics is not cumulative nor relevant. Let P be

  $a \leftarrow \text{not } b.$ $b \leftarrow \text{not } a.$ $c \leftarrow \text{not } a.$ $c \leftarrow \text{not } c.$

  whose unique stable model is {b,c}.
  - Non-cumulative: b is not true in $P \cup \{c\}$.
    - $P \cup \{c\}$ has 2 stable models: {b,c} and {a,c}, so only c is true.
  - Non-relevant: b is not true in $Rel_b(P)$.
    - the rules in $Rel_b(P)$ are $a \leftarrow \text{not } b.$ and $b \leftarrow \text{not } a.$
    - $Rel_b(P)$ has 2 stable models: {b} and {a}, so b and a are not true.

Problems with Stable Models

- The computation of Stable Models is NP-Complete (for normal logic programs)
- The stable models semantics (taken as the intersection of all stable modes) is non-supported.
  - Let P be a←not b   b←not a.   c←a. c←b.

  P has two stable models: {a,c} and {b,c}, so c is true in P, even though there is no rule whose body is true in P (neither a nor b are true in P).

ASP vs. Prolog-like programming

- ASP is adequate for:
  - NP-complete problems
  - situations where the whole program is relevant for the problem at hand
- But if the problem is polynomial, why use such a complex system?
- If only part of the program is relevant for the desired query, why compute the entire model?

ASP vs. Prolog like programming

- For such problems, top-down, goal-driven mechanisms seem more adequate
- This type of mechanisms is used by Prolog
  - Solutions come in variable substitutions rather than in complete models
  - The system is activated by queries
  - No global analysis is made
    - only the relevant part of the program is visited

Problems with Prolog

- Declarative semantics of Prolog is the completion
  - All the problems of completion are inherited by Prolog
- According to SLDNF, termination is not guaranteed
  - even for Datalog programs (i.e. programs with finite ground version)
- A proper semantics is still needed

Well-Founded Semantics

- Defined in [GRS90], generalizes SMs to 3-valued models (true/undefined/false).
- Note that
  - there are programs with no fixpoints of Γ_P
  - but all programs have fixpoints of $\Gamma_P{}^2$
    - recall that $\Gamma_P(I) = least(P/I)$
  - $P=\{a \leftarrow not\ a\}$
    - $\Gamma_P(\{a\})=\{\}$ and $\Gamma_P(\{\})=\{a\}$ so there are no Stable Models
    - But $\Gamma_P{}^2(\{a\})=\{a\}$ and $\Gamma_P{}^2(\{\})=\{\}$

Partial Stable Models

- A three-valued interpretation T ∪ not F is a Partial Stable Model (PSM) if:
  - $T=\Gamma_P^2(T)$
  - $T \subseteq \Gamma_P(T)$
  - $F=H_P-\Gamma_P(T)$

  The 2nd condition guarantees that no atom is both true and false: $T \cap F = \emptyset$
- P={a← not a}
  - has a unique PSM: {}
- P={a←not b. b←not a. c←not a. c←not c.}
  - Has three PSMs: {}, {a, not b} and {c, b, not a}
  - The last one ({c, b, not a}) corresponds to the unique SM.

Well-Founded Model

- Let P be a program. The Well-Founded Model (WFM) of P is the least Partial Stable Model (w.r.t. knowledge ordering i.e. $\subseteq$).
- Given a program P, consider the following transfinite sequence:
  - $T_0 = \{\}$
  - $T_{i+1} = \Gamma_P^2(T_i)$
  - $T_\delta = \cup_{\alpha<\delta} T_\alpha$
  - …and let T be its least fixpoint.
- $I = T \cup not\ (H_P - \Gamma_P(T))$ is the Well-Founded Model of P.

Well-Founded Semantics

- Let $I = T \cup \text{not } F$ be the Well-Founded Model of P. Then, according to the well-founded semantics:
  - A is true in P iff $A \in I$
  - A is false in P iff $\text{not } A \in I$ (i.e. if $A \in F$)
  - A is undefined in P otherwise (i.e. $A \notin I$ and $\text{not } A \notin I$),

Properties of the Well-Founded Semantics

- Every program is assigned a meaning
- For each SM, there is a PSM extending it
  - If WFM is total, it coincides with the single SM
- It is sound w.r.t. the SM semantics
  - If P has stable models and A is true (resp. false) in the WFM, it is also true (resp. false) in all SMs
- WFM coincides with the least model in definite programs

Properties of the Well-Founded Semantics

- The WFM is supported
- WFS is cumulative and relevant
- Its computation is polynomial
  - on the number of instantiated rules of P
- There are top-down proof-procedures, and sound implementations

Stable Models Problems Revisited

- The previously mentioned problems of the Stable Models are not necessarily problematic
  - Relevance is not desired when analyzing global problems
  - If the SMs correspond to the solutions of a problem, programs without SMs simply correspond to problems without solutions.
  - Some problems are in NP. So using an NP language is not a problem.
  - In case of NP problems, the efficiency gains from cumulativity are not really an issue.

Stable Models vs. Well-Founded Model

- Yield different forms of programming and of representing knowledge, for usage with different purposes
- Well-Founded Model:
  - Closer to that of Prolog
  - Local reasoning (and relevance) are important
  - When efficiency is an issue even at the cost of expressivity
- Stable Models
  - For dealing with NP-complete problems
  - Global reasoning
  - Different form of programming, not close to that of Prolog
    - Solutions are models, rather than answer/substitutions

Adding Strong Negation

- In Normal LPs all the negative information is implicit.
- Though that is desired in some cases (e.g. the database with flight connections), sometimes an explicit form of negation, is needed for Knowledge Representation.
- For example, we may want to say that penguins do not fly using the rule:

$$no\_fly(X) \leftarrow penguin(X)$$

- But if we also have a rule:

$$fly(X) \leftarrow bird(X)$$

- We do not have any logical relation between no_fly(X) and fly(X).
- We would like to have ¬ (strong negation) to be able to write:

$$\neg fly(X) \leftarrow penguin(X)$$

- …and deal with it in a way that fly(X) and ¬fly(X) are related (and inconsistent).

Adding Strong Negation

- Also, in rule bodies one form of negation does not seem to be enough…
- For example, it is fine to define innocence in terms of guilt as follows:

$$\text{innocent}(X) \leftarrow \text{not guilty}(X)$$

- But what if we want to define guilt in terms of innocence? The following rule does not seem appropriate:

$$\text{guilty}(X) \leftarrow \text{not innocent}(X)$$

- We should require that someone is (really) not innocent, instead of not innocent by default. The rule should be something like:

$$\text{guilty}(X) \leftarrow \neg\text{innocent}(X)$$

Adding Strong Negation

- The difference between not p and ¬p is essential whenever information about p cannot be assumed.
  - Open vs. Closed World Assumption

Adding Strong Negation to Stable Models

- Historically, the addition of Strong Negation to the Stable Model Semantics coincided with the change in name from Stable Models to Answer Sets.
- The simpler way to extend the Stable Models semantics is to:
  - Extend the Herbrand base H_P with the set $\{\neg A \mid A \in H_P\}$
  - Extend every program with the ICs, for every $A \in H_P$

$$\leftarrow \neg A, A.$$

  - Treat $\neg A$ and A as if they are both unrelated atoms.

Adding Strong Negation to the Well-Founded Semantics

- Generalizing the WFS the same way is not appropriate. Consider for example the program:

  pacifist(X)$\leftarrow$ not hawk(X).

  hawk(X) $\leftarrow$ not pacifist(X).

  $\neg$pacifist(kissinger)

- Using the same method, the WFS would be {$\neg$pacifist(kissinger)}. Despite the fact that we are explicitely stating that kissinger is not a pacifist, we cannot conclude that he is a hawk!
- Coherence needs to be imposed, i.e., $\neg L \in T \Rightarrow L \in F$
  - For L = A or L = $\neg$A and $\neg\neg A = A$

WFSX

- The semi-normal version of P, P_S, is obtained by adding not ¬L to every rule of P with head L.
  - So, pacifist(X)← not hawk(X). becomes
    pacifist(X) ← not hawk(X), not ¬pacifist(X).
- A three-valued interpretation T ∪ not F is a Partial Stable Model of P:
  - $T = \Gamma_P \Gamma_{P_S}(T)$
  - $T \subseteq \Gamma_{P_S}(T)$
  - $F = H_P - \Gamma_{P_S}(T)$
- Let P be a program. The WFSX model of P is the least Partial Stable Model (w.r.t. knowledge ordering i.e. ⊆).

WFSX Example

P:

pacifist(X)←not hawk(X).

hawk(X)←not pacifist(X).

¬pacifist(k).

P_S:

pacifist(X)←not hawk(X), not ¬pacifist(X).

hawk(X)←not pacifist(X), not ¬ hawk(X).

¬pacifist(k)← not pacifist(k).

Assume we have another person b.

$T_0 = \{\}$

$\Gamma_{P_S}(T_0) = \{\neg p(k), p(k), h(k), p(b), h(b)\}$

$T_1 = \Gamma_P \Gamma_{P_S}(T_0) = \{\neg p(k)\}$

$\Gamma_{P_S}(T_1) = \{\neg p(k), h(k), p(b), h(b)\}$

$T_2 = \Gamma_P \Gamma_{P_S}(T_1) = \{\neg p(k), h(k)\}$

$\Gamma_{P_S}(T_2) = \{\neg p(k), h(k), p(b), h(b)\}$

$T_3 = \Gamma_P \Gamma_{P_S}(T_2) = \{\neg p(k), h(k)\}$

$T_3 = T_2$

The well-founded model is:

{¬pacifist(k), hawk(k), not pacifist(k), not ¬hawk(k), not ¬pacifist(b), not ¬hawk(b)}

Properties of WFSX

- Complies with the coherence principle
- Coincides with WFS for normal programs
- If WFSX is total, it coincides with the unique answer set
- It is sound w.r.t. answer sets
- It is supported, cumulative, and relevant
- Its computation is polynomial
- It has sound implementations

Inconsistent Programs

- Some programs have no WFSX model.

  $a \leftarrow \qquad \neg a \leftarrow$

- Three alternatives:
- Explosive approach: everything follows from contradiction
  - like in First-Order Logic
  - provides no information in the presence of contradiction
- Belief revision approach: remove contradiction by revising P
  - computationally expensive
- Paraconsistent approach: isolate contradiction
  - efficient
  - allows to reason about the non-contradictory part

WFSXp

- A three-valued interpretation $T \cup not\ F$ is a Paraconsistent Partial Stable Model of P (the condition $T \subseteq \Gamma_{P_S}(T)$ is dropped):
  - $T = \Gamma_P \Gamma_{P_S}(T)$
  - $F = H_P - \Gamma_{P_S}(T)$
- Let P be a program. The WFSXp model of P is the least Paraconsistent Partial Stable Model (w.r.t. knowledge ordering i.e. $\subseteq$).

WFSXp Example

P:

c←not b.

b←a.

d←not e.

a←.

¬a←.

P_S:

c←not b, not ¬c.

b←a, not ¬b.

d←not e, not ¬d.

a←not ¬a.

¬a←not a.

$T_0 = \{\}$

$\Gamma_{P_S}(T_0) = \{\neg a, a, b, c, d\}$

$T_1 = \Gamma_P \Gamma_{P_S}(T_0) = \{\neg a, a, b, d\}$

$\Gamma_{P_S}(T_1) = \{d\}$

$T_2 = \Gamma_P \Gamma_{P_S}(T_1) = \{\neg a, a, b, c, d\}$

$\Gamma_{P_S}(T_2) = \{d\}$

$T_3 = \Gamma_P \Gamma_{P_S}(T_2) = \{\neg a, a, b, c, d\}$

$T_3 = T_2$

The well-founded model is

{¬a, a, b, c, d, not a, not ¬a, not b, not ¬b, not c, not ¬c, not ¬d, not e}

House M.D.

- A patient arrives with: sudden epigastric pain; abdominal tenderness; signs of peritoneal irritation
- The rules for diagnosing are:
- if he has sudden epigastric pain, abdominal tenderness, and signs of peritoneal irritation, then he has perforation of a peptic ulcer or an acute pancreatitis
- the former requires major surgery, the latter therapeutic treatment
- if he has high amylase levels, then a perforation of a peptic ulcer can be exonerated
- if he has Jobert's manifestation, then pancreatitis can be exonerated
- In both situations, the patient should not be nourished, but should take H2 antagonists

House M.D.

perforation ← pain, abd-tender, per-irrit, not high-amylase

pancreat ← pain, abd-tender, per-irrit, not jobert

¬nourish ← perforation

h2-ant ← perforation

¬nourish ← pancreat

h2-ant ← pancreat

surgery ← perforation

anesthesia ← surgery

¬surgery ← pancreat

pain.

per-irrit.

¬high-amylase.

abd-tender.

¬jobert.

- The WFSXp model is:

  {pain, not ¬pain, abd-tender, not ¬abd-tender, per-irrit, not ¬per-irrit, ¬high-am, not high-am, ¬jobert, not jobert, perforation, not ¬perforation, pancreat, not ¬pancreat, ¬nourish, not nourish, h2-ant, not ¬h2-ant, surgery, ¬surgery, not surgery, not ¬surgery, anesthesia, not anesthesia, not ¬anesthesia}

House M.D.

The WFSXp model is:

{pain, not ¬pain, abd-tender, not ¬abd-tender, per-irrit, not ¬per-irrit, ¬high-am, not high-am, ¬jobert, not jobert, perforation, not ¬perforation, pancreat, not ¬pancreat, ¬nourish, not nourish, h2-ant, not ¬h2-ant, surgery, ¬surgery, not surgery, not ¬surgery, anesthesia, not anesthesia, not ¬anesthesia}

- The symptoms are derived and non-contradictory
- Both perforation and pancreatitis are concluded
- He should not be fed (¬nourish), but should take H2 antagonists
- The information about surgery is contradictory
- Anesthesia, though not explicitly contradictory (¬anesthesia does not belong to WFM) relies on contradiction (both anesthesia and not anesthesia belong to WFM)

Representing Knowledge with WFSX

A methodology for KR

- WFSXp provides mechanisms for representing usual KR problems:
  - logic language
  - non-monotonic mechanisms for defaults
  - forms of explicitly representing negation
  - paraconsistency handling
  - ways of dealing with undefinedness
- In what follows, we propose a methodology for KR using WFSXp

Representation method (1)

Definite rules *If A, then B:*

- B ← A
  - penguins are birds: $bird(X) \leftarrow penguin(X)$

Default rules *Normally, if A, then B:*

- B ← A, rule_name, not ¬B

  rule_name ← not ¬rule_name
  - birds normally fly: $fly(X) \leftarrow bird(X), bf(X), not\ \neg fly(X)$

    $bf(X) \leftarrow not\ \neg bf(X)$

Representation method (2)

Exception to default rules *Under conditions COND, do not apply the rule named rule_name:*

- ¬rule_name ← COND
  - Penguins are an exception to the birds-fly rule $\neg bf(X) \leftarrow penguin(X)$

Preference rules *Under conditions COND, prefer rule RULE$^+$ (named rule_pref) to RULE$^-$: named rule_unpref)*

- ¬rule_unpref ← COND, rule_pref
  - for penguins, prefer the penguins-do-not-fly to the birds-fly rule: $\neg bf(X) \leftarrow penguin(X), pdf(X)$

Representation method (3)

Hypothetical rules *"If A, then B" may or not apply:*

- B ← A, rule_name, not ¬B

  rule_name ← not ¬rule_name

  ¬rule_name ← not rule_name

  - quakers might be pacifists:

    $pacifist(X) \leftarrow quaker(X), qp(X), not\ \neg pacifist(X)$

    $qp(X) \leftarrow not\ \neg qp(X)$

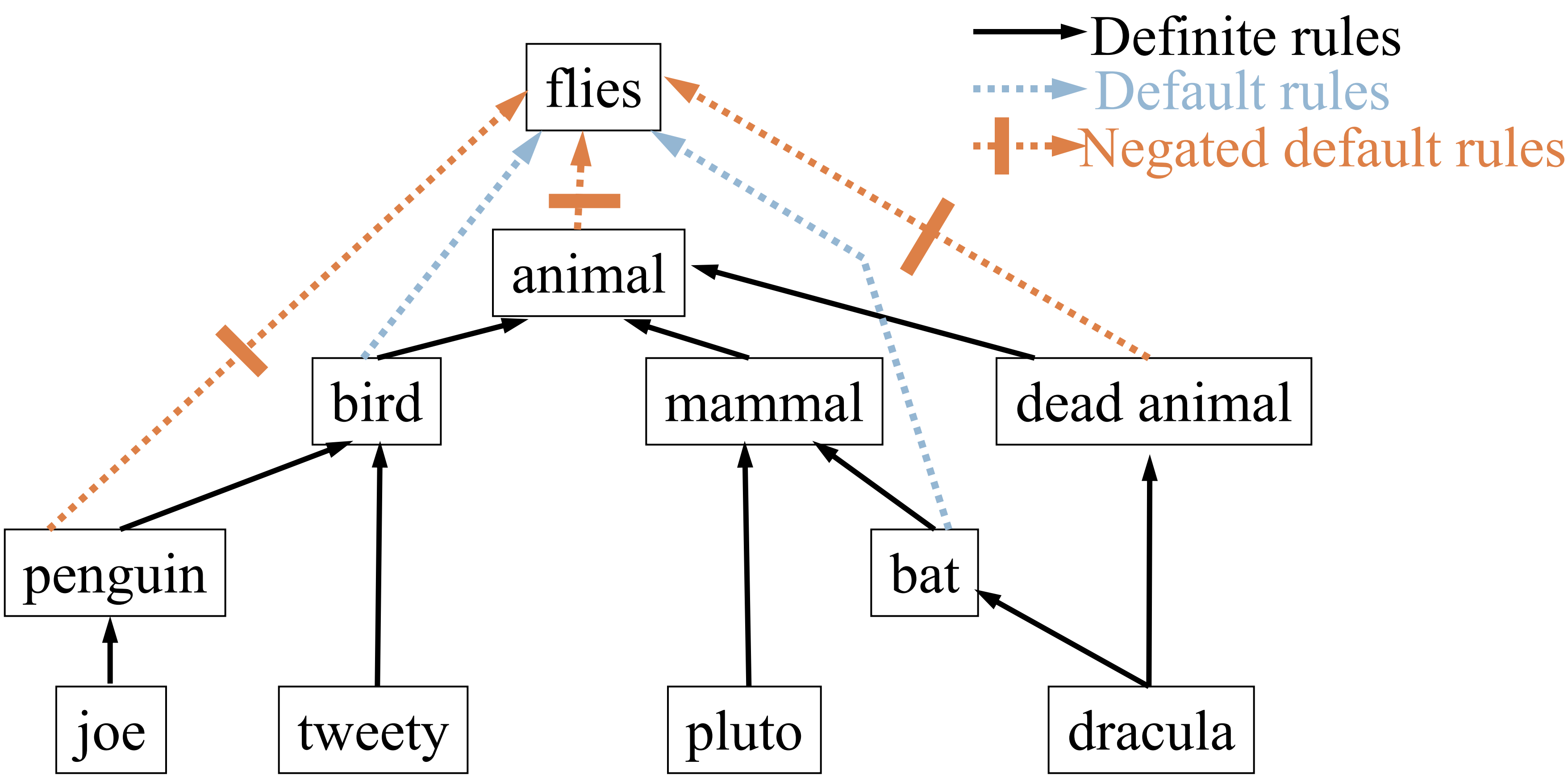
    $\neg qp(X) \leftarrow not\ qp(X)$

For a quaker, there is a PSM with *pacifist*, another with *not pacifist*. In the WFM *pacifist* is undefined

Taxonomy example

- The taxonomy
  - Mammals are animals
  - Bats are mammals
  - Birds are animals
  - Penguins are birds
  - Dead animals are animals
  - Normally animals don't fly
  - Normally bats fly
  - Normally birds fly
  - Normally penguins don't fly
  - Normally dead animals don't fly
- The preferences
  - Dead bats don't fly though bats do
  - Dead birds don't fly though birds do
  - Dracula is an exception to the above
  - In general, more specific information is preferred
- The elements
  - Pluto is a mammal
  - Joe is a penguin
  - Tweety is a bird
  - Dracula is a dead bat

The taxonomy



Taxonomy representation

Taxonomy

animal(X) ← mammal(X)
mammal(X) ← bat(X)
animal(X) ← bird(X)
bird(X) ← penguin(X)
deadAn(X) ← dead(X)

Default rules

¬flies(X) ← animal(X), adf(X), not flies(X)
adf(X) ← not ¬adf(X)
flies(X) ← bat(X), btf(X), not ¬flies(X)
btf(X) ← not ¬btf(X)
flies(X) ← bird(X), bf(X), not ¬flies(X)
bf(X) ← not ¬bf(X)
¬flies(X) ← penguin(X), pdf(X), not flies(X)
pdf(X) ← not ¬pdf(X)
¬flies(X) ← deadAn(X), ddf(X), not flies(X)
ddf(X) ← not ¬ddf(X)

Explicit preferences

¬btf(X) ← deadAn(X), bat(X), r1(X)
r1(X) ← not ¬r1(X)
¬btf(X) ← deadAn(X), bird(X), r2(X)
r2(X) ← not ¬r2(X)
¬r2(dracula)
¬r1(dracula)

Implicit preferences

¬adf(X) ← bat(X), btf(X)
¬adf(X) ← bird(X), bf(X)
¬bf(X) ← penguin(X), pdf(X)

Facts

mammal(pluto).
bird(tweety). deadAn(dracula).
penguin(joe). bat(dracula).

Taxonomy semantics

	joe	dracula	pluto	tweety
deadAn	not	✓	not	not
bat	not	✓	not	not
penguin	✓	not	not	not
mammal	not	✓	✓	not
bird	✓	not	not	✓
animal	✓	✓	✓	✓
adf	✓	¬	✓	¬
btf	✓	¬	✓	✓
bf	¬	✓	✓	✓
pdf	✓	✓	✓	✓
ddf	✓	¬	✓	✓
r1	✓	¬	✓	✓
r2	✓	¬	✓	✓
flies	¬	✓	¬	✓