Knowledge Representation
Chapter 2. Propositional Representation and Reasoning (II)

Pedro Cabalar

Dept. Computer Science
University of Corunna, SPAIN

February 27, 2021

Rule-based reasoning

- Rules are a substantial ingredient of commonsense reasoning
- Example:

"fire causes smoke"

`smoke if fire` `smoke :- fire`

logic programming notation

We sometimes write:

$$smoke \leftarrow fire \qquad \underbrace{smoke}_{head} \leftarrow \underbrace{fire}_{body}$$

Rule-based reasoning

Two possible readings

- Rule firing (bottom-up):
  "I make a *fire*, so I get *smoke* as a byproduct"

  $$\frac{smoke \leftarrow fire \qquad fire}{smoke} \quad = \text{Modus Ponens}$$

  Better for causal inference (used in Answer Set Programming)

- Goal achievement (top-down):
  "How can I get *smoke*? one way is making a *fire*"

  $$\begin{array}{rlll} \text{goal} = & smoke? & & \\ & \underline{smoke} & \leftarrow fire & \text{rule head found} \\ \text{new goal} = & fire? & & \\ & \underline{fire} & & \text{fact found = success!} \end{array}$$

  Goal-oriented backtracking (used in Prolog)

Rules as Logical Formulas

- First choice: translate as material implication in classical logic

$$smoke \leftarrow fire \qquad \equiv \qquad \neg fire \vee smoke$$

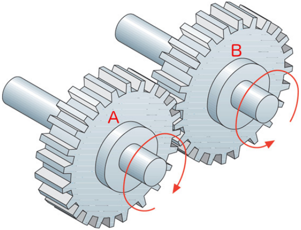
✔ Modus Ponens is granted

✘ But semantics is not aligned with rule-based reasoning
Suppose we only knew $KB = \{smoke \leftarrow fire\}$

Rule reasoning	Classical models	
fire=false:	$\{fire, smoke\}$	derivability?
no way to be derived	$\{smoke\}$	derivability?
smoke=false:	$\emptyset$	both false ✔
only derivable if *fire*		minimal model

Minimal models and recursion

- Minimal models cover (positive) recursion nicely
- Example: two gear wheels



$$\begin{array}{rcl} spinA & \leftarrow & spinB \\ spinB & \leftarrow & spinA \end{array}$$

Two classical models

$\{spinA, spinB\}$	unjustified movement!
$\emptyset$	nothing moves ✔
☝	minimal model

Positive Logic Programs (syntax)

- A positive logic program is a set of rules like

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

  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.

- 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 (semantics)

- Read rule $(p \leftarrow q_1, \ldots, q_n)$ as $(q_1 \wedge \cdots \wedge q_n \rightarrow p)$
- Close World Assumption (CWA) (minimize truth):
  get the model(s) with $\subseteq$-less true atoms
- In general, we may get several $\subseteq$-minimal models.
  Ex. $M(p \vee q) = \{\{p\}, \{q\}, \{p, q\}\}$, two minimal models $\{p\}, \{q\}$
- Positive programs have exactly one: the $\subseteq$-least model $LM(P)$.
  Example:

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

the models are $\{p, q, r, s\}$, $\{p, q, r, s, a, b\}$, $\{p, q, r, s, a, b, c\}$.

Positive Logic Programs (computation)

- The least model can be easily computed by “rule application” (deductive closure).
- Direct consequences operator [van Endem & Kowalski 76]
  $T_P(\mathcal{I})$ = collect all heads in program P whose bodies are true in $\mathcal{I}$

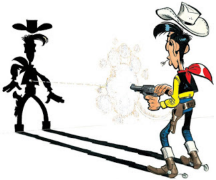
$$T_P(\mathcal{I}) := \{H \mid (H \leftarrow B) \in P \text{ and } \mathcal{I} \models B\}$$

- Compute sequence of interpretations $\mathcal{I}_0, \mathcal{I}_1, \mathcal{I}_2, \ldots$

  Start with $\mathcal{I}_0 := \emptyset$ (all atoms false)

  Repeat $\mathcal{I}_{k+1} := T_P(\mathcal{I}_k)$ until we reach a fixpoint $\mathcal{I}_{k+1} = \mathcal{I}_k$

Positive Logic Programs (computation)



Go "firing rules" (Modus Ponens) until nothing new is derived

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

$T_P(\emptyset) = \{p, q\}$, $T_P(\{p, q\}) = \{p, q, s\}$, $T_P(\{p, q, s\}) = \{p, q, s, r\}$, $T_P(\{p, q, s, r\}) = \{p, q, s, r\}$ fixpoint = least model $LM(P)$! proved by [van Endem & Kowalski 76]

- Each true atom is justified by a proof by Modus Ponens

$$
\cfrac{p \qquad \cfrac{q \qquad s \leftarrow q}{s} \qquad r \leftarrow p, s}{r}
$$

Default Negation

- Goal: incorporating default reasoning in rules
- CWA means false by default.
  But we cannot check falsity in rules

☞ Idea: allow negative literals in rule bodies
"*not p*" = "no evidence/proof for p" = "$\neg p$ can be assumed"

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

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

with $n \geq m \geq 0$. If $m = n$ (no negations) we get a positive rule.
Again, ordering is irrelevant.

Default Negation

Example: "fill the tank if empty and no evidence on fire"

$$fill \leftarrow empty, not\ fire$$

Suppose that the tank is empty indeed:

$$empty$$

Expected behaviour:

- No rule to derive *fire*, so we derive *not fire* then we get *fill* by Modus Ponens: final model $\{empty, fill\}$

Default Negation

⚠ Classical logic reading $empty \wedge (empty \wedge \neg fire \rightarrow fill)$ with minimal models (CWA) does not suffice!

- Classically equivalent to $empty \wedge (fill \vee fire)$. Minimal models: $\{empty, fill\}$ but also $\{empty, fire\}$.
- Assuming there might be a *fire* is ok but there is no proof for *fire* ☞ any assumption must be eventually ...

- We expect non-monotonicity. Example: adding the fact *fire* should now derive $\{empty, fire\}$ and retract *fill*

Default Negation

- Problem: material implication is not directional
- These formulas are classically equivalent:

$$\begin{aligned} \mathit{empty} \wedge \neg \mathit{fire} \rightarrow \mathit{fill} & \equiv \mathit{empty} \rightarrow \mathit{fire} \vee \mathit{fill} \\ & \equiv \mathit{empty} \wedge \neg \mathit{fill} \rightarrow \mathit{fire} \end{aligned}$$

but writing the latter as a rule

$$\mathit{fire} \leftarrow \mathit{empty}, \mathit{not\ fill}$$

"If empty and no evidence on filling then start a fire"
has a quite different meaning!

Default Negation

Sometimes defaults are conflicting.
A classical example: Nixon's diamond

- "*q*uakers are normally *p*acifist" (unless *b*ellicose)
- "*r*epublicans are normally *b*ellicose" (unless *p*acifist)
- "Richard Nixon is a both a Quaker and a Republican"

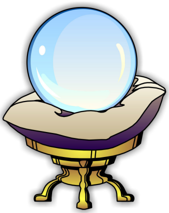
$$\begin{array}{lcl} p & \leftarrow & q, \mathit{not}\ b \\ b & \leftarrow & r, \mathit{not}\ p \\ q & & \\ r & & \end{array}$$

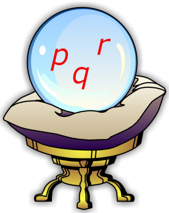
- There is no constructive way to apply the rules

Adding negation: stable models

- Gelfond, M., and Lifschitz, V. (ICLP 1988)
  The stable model semantics for logic programming.

Step 1	Step 2	
Guess an assumption	Reduce program *not*'s accordingly	pr





Default negation:
not b

$$
\begin{array}{lcl}
p & \leftarrow & q, \mathit{not}\ b \top \\
b & \leftarrow & r, \mathit{not}\ p \bot \\
q & & \\
r & &
\end{array}
$$

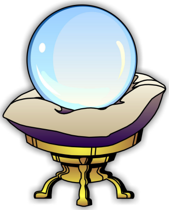
Min
= as
stab

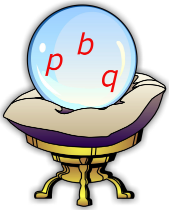
Sym
stab

Adding negation: stable models

- Gelfond, M., and Lifschitz, V. (ICLP 1988)
  The stable model semantics for logic programming.

Step 1	Step 2	
Guess an assumption	Reduce program *not*'s accordingly	pr
		Min assu





Default negation:
not r

$$\begin{array}{lll} p & \leftarrow & q, \textit{not } b \perp \\ b & \leftarrow & r, \textit{not } p \perp \\ q & & \\ r & & \end{array}$$

uns
p, b
not

Stable models: formal definition

Definition (program reduct)

$P^{\mathcal{I}}$ = *reduct of program* P *with respect to interpretation* $\mathcal{I}$

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

☞ Observation: $P^{\mathcal{I}}$ is positive, it has a least model $LM(P^{\mathcal{I}})$!

Definition (stable model)

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

Stable models: some properties

$M(P)$="classical models of P"; $SM(P)$="stable models of P"

Proposition (Stable models are models)

$SM(P) \subseteq M(P)$. Any stable model of P is also a classical model.

When the program is normal (things will change with disjunction):

Proposition (Stable models are minimal classical models)

If $\mathcal{I} \in SM(P)$ then there is no $J \in M(P)$, $J \subset \mathcal{I}$.

Proposition (Complexity)

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

Stable models

Back to the example. P has 2 rules:

$$\boxed{\begin{array}{rcl} fill & \leftarrow & empty, \text{not } fire \\ empty & & \end{array}}$$

Three atoms: possible assumptions $\mathcal{I} = 2^3$

💡 $SM(P) \subseteq M(P)$, just check the 3 classical models!

$\mathcal{I}$	$P^{\mathcal{I}}$	$LM(P^{\mathcal{I}})$
$\{empty, fire\}$	$empty$	$\{empty\} \neq \mathcal{I}$ not stable
$\{empty, fire, fill\}$	$empty$	$\{empty\} \neq \mathcal{I}$ not stable
$\{empty, fill\}$	$fill \leftarrow empty$ $empty$	$\{empty, fill\}$ stable!

Stable models

Suppose a spark starts a fire now. P has 4 rules:

$$\begin{array}{rcl} fill & \leftarrow & empty, \textit{not } fire \\ empty & & \\ fire & \leftarrow & spark \\ spark & & \end{array}$$

- Only two (classical) models now:

$\mathcal{I}$	$P^{\mathcal{I}}$	$LM(P^{\mathcal{I}})$
$\{empty, spark, fire\}$	$empty$; $fire \leftarrow spark$; $spark$	$\{empty, spark, fire\}$ stable!
$\{empty, spark, fire, fill\}$	$empty$; $fire \leftarrow spark$; $fire$	$\{empty, spark, fire\} \neq I$ not stable

Stable models: non-monotonicity

Observation: the example shows non-monotonic reasoning!

- Example 1: stable model $\{empty, fill\}$ allowed us to conclude *fill*
- Example 2: adding new formulas "a spark started a fire" stable model $\{empty, spark, fire\}$ retracts previous conclusion (*fill* is not true any more)

Stratified programs

- A normal program is stratified if it has no cycles through negation

☞ Rules can be organized in layers: negation means a layer jump.

$$
\begin{array}{lllll}
\text{Layer 1} & \left\{ \begin{array}{l} a \\ b \end{array} \right. & \begin{array}{c} \\ \leftarrow \end{array} & \begin{array}{l} \\ a \end{array} & \{a, b\} \\[2ex]
\text{Layer 2} & \left\{ \; c \right. & \leftarrow & \mathit{not}\ a\, \underbrace{\mathit{not}\ a}_{\bot} & \{a, b\} \\[2ex]
\text{Layer 3} & \left\{ \; d \right. & \leftarrow & b, \mathit{not}\ c\, \underbrace{\mathit{not}\ c}_{\top} & \{a, b, d\}
\end{array}
$$

Proposition

A stratified program has a unique stable model $|SM(P)| = 1$.

Incoherent programs

- If P unstratified we may have $|SM(P)| > 1$
  but also $|SM(P)| = 0$! P is called incoherent if $SM(P) = \emptyset$
  This may happen even if $M(P) \neq \emptyset$ (classically consistent).

- Example (Russell's paradox):
  "make a *C*atalogue citing of all books without self-citations"

$citeC$ $citeC$ $\leftarrow$ $\leftarrow$ $not\ selfC$ $not\ selfC$ Assume $\mathcal{I} \models selfC$ As
$selfC \leftarrow citeC$ proved = $\emptyset$ *selfC* unju

- An even simpler example: $problem \leftarrow not\ \ problem$

Choices and constraints

We can use auxiliary atoms to exploit negative cycles as follows:

- Choice rule: nondeterministic generation of an atom.
  Ex: when *spark*, sometimes *fire* and sometimes no

  $$\boxed{fire \leftarrow spark, not\ aux \qquad aux \leftarrow spark, not\ fire}$$

  Adding fact *spark* yields $\{spark, fire\}$ and $\{spark, aux\} = \{spark\}$ if we remove *aux*. Common abbreviation = choice rule:

  $$\boxed{\{fire\} \leftarrow spark}$$

- Constraint: dismiss stable models when a condition holds.
  If *wet* holds, choosing *fire* is disregarded.

  $$\boxed{aux \leftarrow wet, fire, not\ aux}$$

  Common abbreviation = constraint:

  $$\boxed{\bot \leftarrow wet, fire \qquad \text{or simply} \qquad \leftarrow wet, fire}$$

Splitting

Atom p is defined in P when some $(p \leftarrow B) \in P$ (possibly $B = \top$)

Some programs P can be splitted in two parts P_B, P_T

- the bottom P_B contains no atom defined in P_T
- the top P_T does not define atoms occurring in P_B

$$
\begin{array}{ll}
P_B \left\{ \begin{array}{l} \{spark\} \\ \{fire\} \leftarrow spark \\ \leftarrow wet, fire \end{array} \right. & \begin{array}{l} \emptyset \\ \{spark\} \\ \{spark, fire\} \end{array} \\
P_T \left\{ \begin{array}{l} empty \\ fill \leftarrow empty, not\ fire \end{array} \right. & \begin{array}{l} \{\underline{empty}, \underline{fill}\} \\ \{spark, \underline{empty}, \underline{fill}\} \\ \{spark, fire, \underline{empty}\} \end{array}
\end{array}
$$

- First compute the stable models of the bottom then use each of them for the top

Beyond normal programs

- Disjunctive programs: we allow multiple atoms H_i in the head

$$p_1, \ldots, p_k \leftarrow q_1, \ldots, q_n, not\ q_{n+1}, \ldots, not\ q_m$$

  Commas in the head correspond to disjunctions $\vee$.

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

$$\boxed{c, tv \leftarrow not\ b}$$

- The reduct P^I defined as before, but is not a positive program now! There is no least model $LM(P^I)$ any more.

- Example: for $I = \{c\}$, P^I is the program $\boxed{c, tv}$
  $c \vee tv$ has 3 classical models $\{c\}$, $\{tv\}$ and $\{c, tv\}$.

Definition (stable model)

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

In our example:

$$\boxed{c, tv \leftarrow not\ b}$$

I	P^I	minimal models	*Stable*?
$\{c\}$	c, tv	$\{c\}$ $\{tv\}$	*yes*
$\{tv\}$	c, tv	$\{c\}$ $\{tv\}$	*yes*
$\{c, tv\}$	c, tv	$\{c\}$ $\{tv\}$	*no*
$I \models b$		$\emptyset$	*no*

Property still preserved when P is disjunctive:

Proposition

Stable models are minimal classical models:
$I \in SM(P)$ *implies* $I \in M(P)$ *and no* $J \in M(P)$ *is smaller* $J \subset I$.

But complexity increases:

Proposition

Deciding $SM(P) \neq \emptyset$ *for disjunctive programs 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**.