Reasoning and Planning
Unit 3. Rule-based Reasoning

Pedro Cabalar

Dept. Computer Science
University of Corunna, SPAIN

October 11, 2022

1. Rule-based reasoning

2. Default negation

3. Equilibrium Logic

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 ← *fire*

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

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*"

goal $=$	*smoke*?		
	<u>*smoke*</u>	$\leftarrow$ *fire*	rule head found
new goal $=$	*fire*?		
	<u>*fire*</u>		fact found = success!

  Goal-oriented backtracking (used in Prolog)

Rules as Logical Formulas

- First choice: translate as material implication in classical logic

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

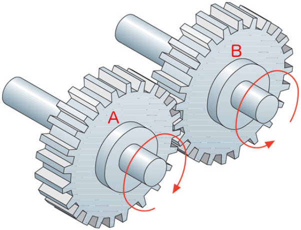
✔ Modus Ponens is granted

✖ 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:

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

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 $\mathcal{I}$

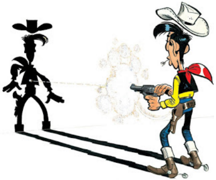
$$T_P(\mathcal{I}) := \{H \mid (H \leftarrow B) \in P \text{ and } \mathcal{I} \models B\}$$

- 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

$$
\begin{array}{llllllllll}
pp & & & & ss & \leftarrow & qq & b & \leftarrow & ss, a \\
qq & & & & a & \leftarrow & b, pp & a & \leftarrow & c \\
rr & \leftarrow & pp, ss & & & & & & &
\end{array}
$$

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

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

$$
\frac{p \qquad \dfrac{q \qquad s \leftarrow q}{s} \qquad r \leftarrow p, s}{r}
$$

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:

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

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

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

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

"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

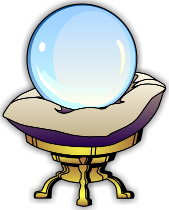
- "*q*uakers are normally *p*acifist" (unless *b*ellicose)
- "*r*epublicans are normally *b*ellicose" (unless *p*acifist)
- "Richard Nixon is a both a Quaker and a Republican"

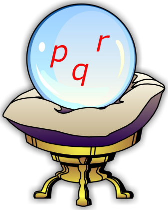
$$
\begin{array}{lcl}
p & \leftarrow & q, \mathit{not}\ b \\
b & \leftarrow & r, \mathit{not}\ p \\
q & & \\
r & &
\end{array}
$$

- There is no constructive way to apply the rules

- 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

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

Mir
= a
sta

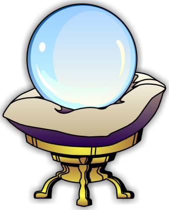
Syr
sta

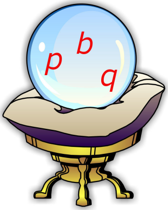
Adding negation: stable models

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

Step 1

Guess an
assumption





Default negation:
not r

Step 2

Reduce program
not's accordingly

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

p

Mir
ass
uns
p, *b*
not

Stable models: formal definition

Definition (program reduct)

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

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

☞ 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:

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

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:

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

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

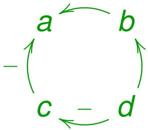
Stratified programs

- Dependence graph of a program:
  - nodes = atoms
  - edge $p \rightarrow q$ if $\exists$ rule with p in the head and q in the positive body
  - edge $p \xrightarrow{-} q$ if $\exists$ rule with p in the head and q in the negative body
- A normal program is stratified if it has no cycles through negation

Program 1:

$$
\begin{array}{lll}
a & & \\
b & \leftarrow & a \\
c & \leftarrow & not\ a \\
d & \leftarrow & b, not\ c
\end{array}
$$

Graph 1:



stratified

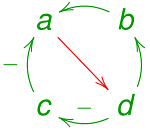
Stratified programs

- Dependence graph of a program:
  - nodes = atoms
  - edge $p \rightarrow q$ if $\exists$ rule with p in the head and q in the positive body
  - edge $p \xrightarrow{-} q$ if $\exists$ rule with p in the head and q in the negative body
- A normal program is stratified if it has no cycles through negation

Program 2:

$$
\begin{array}{lll}
a & & \\
b & \leftarrow & a \\
c & \leftarrow & \textit{not } a \\
d & \leftarrow & b, \textit{not } c \\
a & \leftarrow & d
\end{array}
$$

Graph 2:



non-stratified!

Stratified programs

- When a program is stratified
  ☞ Rules can be organized in layers: negation means a layer jump.

Layer 1	$\left\{ \begin{array}{lll} a & & \\ b & \leftarrow & a \end{array} \right.$	$\{a, b\}$
Layer 2	$\left\{ \begin{array}{lll} c & \leftarrow & not\ a\ \underbrace{not\ a}_{\bot} \end{array} \right.$	$\{a, b\}$
Layer 3	$\left\{ \begin{array}{lll} d & \leftarrow & b, not\ c\ \underbrace{not\ c}_{\top} \end{array} \right.$	$\{a, b, d\}$

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"

$$\begin{array}{rcl} \mathit{citeC}\,\mathit{citeC} & \leftarrow\leftarrow & \mathit{not\ selfC}\ \mathit{not\ selfC} \\ \mathit{selfC} & \leftarrow & \mathit{citeC} \end{array}$$

Assume $\mathcal{I} \models \mathit{selfC}$ Assu
proved = $\emptyset$ *selfC* unjust

- An even simpler example: $\mathit{problem} \leftarrow \mathit{not\ \ problem}$

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

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

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

  $$\{\ \mathit{fire}\ \} \leftarrow \mathit{spark}$$

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

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

  Common abbreviation = constraint:

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

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

$$P_B \left\{ \begin{array}{l} \{\, spark \,\} \\ \{\, fire \,\} \leftarrow spark \\ \leftarrow wet, fire \end{array} \right. \qquad \begin{array}{l} \emptyset \\ \{spark\} \\ \{spark, fire\} \end{array}$$

✄ . . . .

$$P_T \left\{ \begin{array}{l} empty \\ fill \leftarrow empty, not\ fire \end{array} \right. \qquad \begin{array}{l} \{\underline{empty}, \underline{fill}\} \\ \{spark, \underline{empty}, \underline{fill}\} \\ \{spark, fire, \underline{empty}\} \end{array}$$

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

A logical characterisation

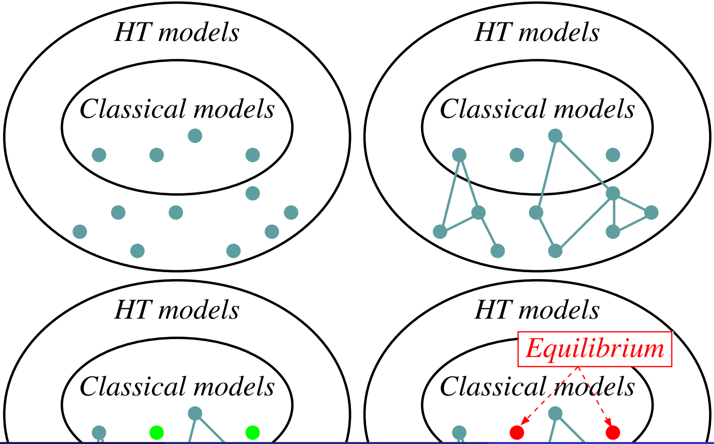
- [GL88] definition of stable models is syntactically limited
  - It relies on a syntactic transformation (reduct)
  - Connectives cannot be freely combined, e.g. $not\ (p \leftarrow q)$
- Later definitions extended the reduct to incorporate $\vee$ in the head and, further, nesting it with commas ($\wedge$) and *not*

⚠ Nesting '$\leftarrow$' was not allowed: this connective had no semantics!

A logical characterisation

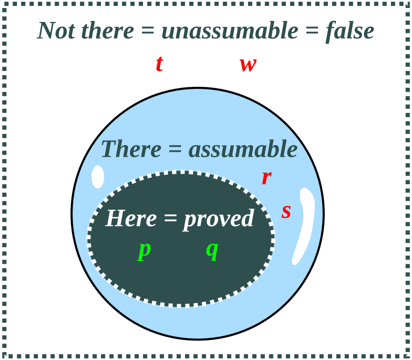
Equilibrium Logic [Pearce96]: generalises stable models for arbitrary propositional theories. Consists of:

1. A non-classical monotonic (intermediate) logic called Here-and-There (HT) [Heyting 30]



Here-and-There

- Interpretation = pairs $\langle H, T \rangle$ of sets of atoms $H \subseteq T$
  Example: $H = \{p, q\}, T = \{p, q, r, s\}$. Intuition:



- Note that we start from: proved $\subseteq$ assumable
- Atoms in $T \setminus H$ are just assumed (assumable but not proved)
- When $H = T$ (assumable = proved) we have a classical model.

Here-and-There

Satisfaction of formulas

$\langle H, T\rangle \models \alpha \quad \Leftrightarrow \quad$ "α is proved"

$\langle T, T\rangle \models \alpha \quad \Leftrightarrow \quad$ "α assumable" $\quad \Leftrightarrow \quad T \models \alpha$ classically

- $\langle H, T\rangle \models p$ if $p \in H$ (for any atom p)
- $\wedge, \vee$ as always
- $\langle H, T\rangle \models \alpha \rightarrow \beta$ if both
  - "If α is proved, then β must be proved":
    $\langle H, T\rangle \models \alpha$ implies $\langle H, T\rangle \models \beta$
  - "$\alpha \rightarrow \beta$ is assumable":
    $T \models \alpha \rightarrow \beta$ classically
- Negation $\neg F$ is defined as $F \rightarrow \bot$

HT properties

Theorem (Persistence)

$\langle H, T\rangle \models \alpha$ *(proved)* *implies* $T \models \alpha$ *(assumable)*

Theorem

$\langle H, T\rangle \models \neg\alpha$	*iff*	$T \models \neg\alpha$
Proving $\neg\alpha$	*amounts to*	*assuming* $\neg\alpha$

Definition (Equilibrium/stable model)

A model $\langle T, T\rangle$ of Γ is an equilibrium model iff

there is no $H \subset T$ such that $\langle H, T\rangle \models \Gamma$.

When this holds, T is called a stable model.

In other words, we cannot leave some assumptions $T \setminus H$ not proved

Here-and-There

- An example: $\alpha = \{\neg b \rightarrow p\}$ (not bellicous implies pacifist)
- The classical models $M(\neg b \rightarrow p) = M(b \vee p)$ are the next 3:
  1. $T = \{p\}$ The only possible subset is $H = \emptyset$
     $\langle H, T\rangle \not\models \neg b \rightarrow p$ because $\langle H, T\rangle \models \neg b$ but $\langle H, T\rangle \not\models p$
     That is $\langle \emptyset, \{p\}\rangle$ is not an HT model
     ☞ Then $\{p\}$ is an equilibrium model! (no smaller H forms a model)
  2. $T = \{b\}$ The only possible subset is $H = \emptyset$
     $\langle H, T\rangle \models \neg b \rightarrow p$ because $\langle H, T\rangle \not\models \neg b$
     Therefore $\langle \emptyset, \{b\}\rangle$ is an HT model (assumption b is not proved)
     ☞ Then $\{b\}$ is not in equilibrium
  3. $T = \{b, p\}$ The possible subsets are $H = \emptyset$, $H = \{b\}$ or $H = \{p\}$
     All of them HT models because $\langle H, T\rangle \not\models \neg b$
     ☞ Then $\{b, p\}$ is not in equilibrium

Expressiveness

- HT is weaker than classical logic
- For instance, $p \vee \neg p$ is not a tautology

  = p is either proved or not assumable
  = it rules out $H = \emptyset$ and $T = \{p\}$ (countermodel)

  - $\langle \emptyset, \{p\} \rangle \not\models p$ because $p \notin H$
  - $\langle \emptyset, \{p\} \rangle \not\models \neg p$ because $p \in T$

- In fact $p \vee \neg p \equiv \neg\neg p \rightarrow p$ which is not valid either ...
  ☞ we cannot remove double negation in $\neg\neg p$

Theorem

$\langle H, T \rangle \models \neg\neg\alpha$ *iff* $T \models \alpha$

Expressiveness

- Captures all syntactic extensions of stable models with propositional connectives (also first-order [Pearce & Valverde 04]).
- Natural representation for:

Logic	Program	Meaning
$\bot \leftarrow Body$	`:- Body.`	constraint forbidding *Body*
$p \vee \neg p \leftarrow Body$	`{p} :- Body.`	choice rule "If *Body* then we are free to derive *p* or not"

- Moreover, covers arbitrary formulas, in a very reasonable way:

$$\text{intuitionistic} \subset \text{HT} \subset \text{classical}$$

Complexity

Theorem

Deciding whether a theory Γ has some equilibrium model is $\Sigma_2^{\mathbf{P}}$-complete.

$\Sigma_2^{\mathbf{P}} = \mathbf{NP}^{\mathbf{NP}}$: means **NP** on a Turing machine with an **NP** oracle. This is (conjectured) harder than **NP**.

Same complexity arises even by just adding disjunction in rule heads:

$$p_1 \vee \ldots \vee p_n \leftarrow q_1, \ldots, q_m, \mathit{not}\ q_{m+1} \ldots \mathit{not}\ q_k$$

Strong equivalence

- Under non-monotonicity, equivalence becomes tricky

Program P_1	Program P_2
empty *fill* $\leftarrow$ *empty*, *not fire* *fire*	*empty* *fill* *fire*
stable model $\{empty, fill\}$ $\{empty, fire\}$	stable model $\{empty, fire\}$ $\{empty, fill, fire\}$

Definition (Strong Equivalence)

Two theories P_1, P_2 are strongly equivalent if $P_1 \cup Q$ and $P_2 \cup Q$ have the same equilibrium models for any theory Q.

Theorem ([Lifschitz, Pearce, Valverde 01])

Strong equivalence of equilibrium theories = HT equivalence .

Deciding HT equivalence is **NP**.

Strong equivalence

Example

Check whether these two programs are strongly equivalent or not

Program P_1	Program P_2
$p \leftarrow \neg b$	$\bot \leftarrow \neg b \land \neg p$
$p \lor \neg p$	$p \lor \neg p$