Answer Set Programming

Pedro Cabalar

Depto. Computación
University of Corunna, SPAIN

July 1, 2010

Outline

Answer Set Programming

- Answer set programming (ASP) [Gelfond & Lifschitz 88]: similar to Prolog, but more declarative.

Answer Set Programming

- Answer set programming (ASP) [Gelfond & Lifschitz 88]: similar to Prolog, but more declarative.
- (Propositional) rules with negation in the body.

$$\underbrace{p}_{head} \leftarrow \underbrace{L_1, \ldots, L_n}_{body}$$

$n \geq 0$, p is an atom and L_i are literals, that is, an atom q or its default negation *not q*.

Answer Set Programming

- The ordering is irrelevant. We can generally write the rule as:

$$p \leftarrow q_1, \ldots, q_m, \mathit{not}\ q_{m+1}, \ldots, \mathit{not}\ q_n. \tag{1}$$

with $n \geq m \geq 0$. A logic program P is a set of rules like (1)

Answer Set Programming

- The ordering is irrelevant. We can generally write the rule as:

$$p \leftarrow q_1, \ldots, q_m, \textit{not } q_{m+1}, \ldots, \textit{not } q_n. \tag{1}$$

with $n \geq m \geq 0$. A logic program P is a set of rules like (1)

- The rule is positive when $m = n$ (no negations).

Answer Set Programming

- The ordering is irrelevant. We can generally write the rule as:

$$p \leftarrow q_1, \ldots, q_m, not\ q_{m+1}, \ldots, not\ q_n. \tag{1}$$

with $n \geq m \geq 0$. A logic program P is a set of rules like (1)

- The rule is positive when $m = n$ (no negations).
- When $n = 0$, the rule is called a fact, and we usually omit the $\leftarrow$.

Positive programs

- Positive programs can be easily computed by "rule application" (deductive closure).

Positive programs

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

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

Positive programs

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

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

That is, pick those rule heads H whose body B holds in I
(a fact H can just be seen as $H \leftarrow \top$). Commas can be seen as $\wedge$.

Positive programs

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

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

  That is, pick those rule heads H whose body B holds in I (a fact H can just be seen as $H \leftarrow \top$). Commas can be seen as $\wedge$.

- 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 programs

- Exercise: prove that T_P is $\subseteq$-monotonic, i.e., if $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.

Positive programs

- Exercise: prove that T_P is $\subseteq$-monotonic, i.e., if $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.
- By Knaster & Tarski's theorem, T_P has a $\subseteq$-least fix point $I = T_P(I)$.

Positive programs

- Exercise: prove that T_P is $\subseteq$-monotonic, i.e., if $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.
- By Knaster & Tarski's theorem, T_P has a $\subseteq$-least fix point $I = T_P(I)$.
- Moreover, T_P is continuous and the l.f.p. can be computed by iteration of T_P on $I_0 = \emptyset$ until reaching a point $I_{i+1} = T_P(I_i) = I_i$.

Positive programs

- Exercise: prove that T_P is $\subseteq$-monotonic, i.e., if $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.
- By Knaster & Tarski's theorem, T_P has a $\subseteq$-least fix point $I = T_P(I)$.
- Moreover, T_P is continuous and the l.f.p. can be computed by iteration of T_P on $I_0 = \emptyset$ until reaching a point $I_{i+1} = T_P(I_i) = I_i$.
- Back to the example

$$p \qquad s \leftarrow q \qquad b \leftarrow s, a$$
$$q \qquad a \leftarrow b, p \qquad a \leftarrow c$$
$$r \leftarrow p, s$$

Positive programs

- Exercise: prove that T_P is $\subseteq$-monotonic, i.e., if $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.
- By Knaster & Tarski's theorem, T_P has a $\subseteq$-least fix point $I = T_P(I)$.
- Moreover, T_P is continuous and the l.f.p. can be computed by iteration of T_P on $I_0 = \emptyset$ until reaching a point $I_{i+1} = T_P(I_i) = I_i$.
- Back to the example

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

$T_P(\emptyset) = \{p, q\}$

Positive programs

- Exercise: prove that T_P is $\subseteq$-monotonic, i.e., if $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.
- By Knaster & Tarski's theorem, T_P has a $\subseteq$-least fix point $I = T_P(I)$.
- Moreover, T_P is continuous and the l.f.p. can be computed by iteration of T_P on $I_0 = \emptyset$ until reaching a point $I_{i+1} = T_P(I_i) = I_i$.
- Back to the example

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

$T_P(\emptyset) = \{p, q\}$, $T_P(\{p, q\}) =$

Positive programs

- Exercise: prove that T_P is $\subseteq$-monotonic, i.e., if $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.
- By Knaster & Tarski's theorem, T_P has a $\subseteq$-least fix point $I = T_P(I)$.
- Moreover, T_P is continuous and the l.f.p. can be computed by iteration of T_P on $I_0 = \emptyset$ until reaching a point $I_{i+1} = T_P(I_i) = I_i$.
- Back to the example

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

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

Positive programs

- Exercise: prove that T_P is $\subseteq$-monotonic, i.e., if $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.
- By Knaster & Tarski's theorem, T_P has a $\subseteq$-least fix point $I = T_P(I)$.
- Moreover, T_P is continuous and the l.f.p. can be computed by iteration of T_P on $I_0 = \emptyset$ until reaching a point $I_{i+1} = T_P(I_i) = I_i$.
- Back to the example

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

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

Positive programs

- Exercise: prove that T_P is $\subseteq$-monotonic, i.e., if $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.
- By Knaster & Tarski's theorem, T_P has a $\subseteq$-least fix point $I = T_P(I)$.
- Moreover, T_P is continuous and the l.f.p. can be computed by iteration of T_P on $I_0 = \emptyset$ until reaching a point $I_{i+1} = T_P(I_i) = I_i$.
- Back to the example

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

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

Positive programs

- Exercise: prove that T_P is $\subseteq$-monotonic, i.e., if $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.
- By Knaster & Tarski's theorem, T_P has a $\subseteq$-least fix point $I = T_P(I)$.
- Moreover, T_P is continuous and the l.f.p. can be computed by iteration of T_P on $I_0 = \emptyset$ until reaching a point $I_{i+1} = T_P(I_i) = I_i$.
- Back to the example

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

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

Positive programs

- Exercise: prove that T_P is $\subseteq$-monotonic, i.e., if $I \subseteq J$, then $T_P(I) \subseteq T_P(J)$.
- By Knaster & Tarski's theorem, T_P has a $\subseteq$-least fix point $I = T_P(I)$.
- Moreover, T_P is continuous and the l.f.p. can be computed by iteration of T_P on $I_0 = \emptyset$ until reaching a point $I_{i+1} = T_P(I_i) = I_i$.
- Back to the example

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

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

Positive programs

- A set of atoms I is a model of a program P, $I \models P$, when $I \models q_1 \wedge \ldots q_m \wedge \neg q_{m+1} \wedge \cdots \wedge \neg q_n \rightarrow p$ for any rule (1) in P.

Positive programs

- A set of atoms I is a model of a program P, $I \models P$, when $I \models q_1 \wedge \dots q_m \wedge \neg q_{m+1} \wedge \cdots \wedge \neg q_n \rightarrow p$ for any rule (1) in P.
- Main result by [van Endem & Kowalski 76]: a positive program P has a least propositional model $LM(P)$ that coincides with T_P least fixpoint.

Positive programs

- A set of atoms I is a model of a program P, $I \models P$, when $I \models q_1 \wedge \ldots q_m \wedge \neg q_{m+1} \wedge \cdots \wedge \neg q_n \rightarrow p$ for any rule (1) in P.
- Main result by [van Endem & Kowalski 76]: a positive program P has a least propositional model $LM(P)$ that coincides with T_P least fixpoint.
- In our example:

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

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

Positive programs

- A set of atoms I is a model of a program P, $I \models P$, when $I \models q_1 \wedge \ldots q_m \wedge \neg q_{m+1} \wedge \cdots \wedge \neg q_n \rightarrow p$ for any rule (1) in P.
- Main result by [van Endem & Kowalski 76]: a positive program P has a least propositional model $LM(P)$ that coincides with T_P least fixpoint.
- In our example:

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

  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.

A semantics for default negation

- Once negation is introduced, we don't have a least Herbrand model any more. We may have different minimal models.

A semantics for default negation

- Once negation is introduced, we don't have a least Herbrand model any more. We may have different minimal models.
- Take the simple program

$$p \leftarrow not\ q$$

A semantics for default negation

- Once negation is introduced, we don't have a least Herbrand model any more. We may have different minimal models.
- Take the simple program

$$p \leftarrow not\ q$$

Intuitively, as no information for q is available, we should conclude model $\{p\}$, that is, q false and p true.

A semantics for default negation

- Once negation is introduced, we don't have a least Herbrand model any more. We may have different minimal models.
- Take the simple program

$$p \leftarrow not\ q$$

  Intuitively, as no information for q is available, we should conclude model $\{p\}$, that is, q false and p true.
- However, this rule is classically equivalent to $q \vee p$ and has three models: $\{p, q\}, \{p\}, \{q\}$, being the last two minimal.

A semantics for default negation

- Once negation is introduced, we don't have a least Herbrand model any more. We may have different minimal models.
- Take the simple program

$$p \leftarrow not\ q$$

  Intuitively, as no information for q is available, we should conclude model $\{p\}$, that is, q false and p true.
- However, this rule is classically equivalent to $q \vee p$ and has three models: $\{p, q\}, \{p\}, \{q\}$, being the last two minimal.
- Furthermore, $q \vee p$ is also equivalent (in classical logic) to:

$$q \leftarrow not\ p$$

  whose "expected" behavior should be obviously different.

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.
- Idea: we could go assuming and concluding until all atoms have a truth value.

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.

- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.

- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.

- Example:

$$\begin{array}{rcl} p & \leftarrow & not\ q \\ q & \leftarrow & p \\ q & \leftarrow & not\ p \end{array}$$

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.
- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.
- Example:

$$\begin{array}{lcl} p & \leftarrow & not\ q \\ q & \leftarrow & p \\ q & \leftarrow & not\ p \end{array}$$

Assume, say, *not* q

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.
- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.
- Example:

$$\begin{array}{rcl} p & \leftarrow & \mathit{not}\ q \\ q & \leftarrow & p \\ q & \leftarrow & \mathit{not}\ p \end{array}$$

Assume, say, *not* q . . . .

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.
- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.
- Example:

$$\begin{array}{rcl} p & \leftarrow & \mathit{not}\ q \\ q & \leftarrow & p \\ q & \leftarrow & \mathit{not}\ p \end{array}$$

Assume, say, *not* q . . . .

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.
- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.
- Example:

$$\begin{array}{rcl} p & \leftarrow & \textit{not } q \\ q & \leftarrow & p \\ q & \leftarrow & \textit{not } p \end{array}$$

Assume, say, *not* q . . . .

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.
- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.
- Example:

$$\begin{array}{lcl} p & \leftarrow & not\ q \\ q & \leftarrow & p \\ q & \leftarrow & not\ p \end{array}$$

Assume, say, *not* q . . . .

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.

- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.

- Example:

$$\begin{array}{rcl} p & \leftarrow & \mathit{not}\ q \\ q & \leftarrow & p \\ q & \leftarrow & \mathit{not}\ p \end{array}$$

Assume, say, *not* q … q, our assumption was inconsistent.

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.

- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.

- Example:

$$\begin{array}{lcl} p & \leftarrow & not\ q \\ q & \leftarrow & p \\ q & \leftarrow & not\ p \end{array}$$

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.

- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.

- Example:

$$\begin{array}{lcl} p & \leftarrow & not\ q \\ q & \leftarrow & p \\ q & \leftarrow & not\ p \end{array}$$

Assume now *not* p . . . .

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.

- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.

- Example:

$$
\begin{array}{lcl}
p & \leftarrow & not\ q \\
q & \leftarrow & p \\
q & \leftarrow & not\ p
\end{array}
$$

Assume now *not* p ....

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.
- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.
- Example:

$$\begin{array}{lcl} p & \leftarrow & \textit{not } q \\ q & \leftarrow & p \\ q & \leftarrow & \textit{not } p \end{array}$$

Assume now *not* p ....

A semantics for default negation

- The problem seems related to a kind of directionality in the implication:
  1. First: assume that q is false;
  2. Second: conclude that p follows from your assumption, *not* q.
- Idea: we could go assuming and concluding until all atoms have a truth value. Problem: we can sometimes get that an assumption must be retracted.
- Example:

$$\begin{array}{rcl} p & \leftarrow & not\ q \\ q & \leftarrow & p \\ q & \leftarrow & not\ p \end{array}$$

Assume now *not* p . . . q, and the first two rules become redundant.

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) I, written P^I, as the set of rules:

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

Stable models

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

Stable models

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

Definition (stable model)

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

Stable models: some properties

Proposition (Stable models are models)

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

Stable models: some properties

Proposition (Stable models are models)

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

Proposition (Stable models are minimal models)

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

Stable models: some properties

Proposition (Stable models are models)

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

Proposition (Stable models are minimal models)

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

Exercise: prove the above theorems.

Stable models

- An example of default. Try this program:

$$\begin{aligned} \mathit{flies} &\leftarrow \mathit{bird}, \mathit{not}\ \mathit{ab} \\ \mathit{bird} & \end{aligned}$$

Stable models

- An example of default. Try this program:

$$\begin{array}{lcl} flies & \leftarrow & bird, not\ ab \\ bird & & \end{array}$$

- This program has these three models:

I	P^I	$LM(P^I)$
$\{bird, ab\}$		
$\{bird, ab, flies\}$		
$\{bird, flies\}$		

Stable models

- An example of default. Try this program:

$$\begin{array}{lcl} \mathit{flies} & \leftarrow & \mathit{bird}, \mathit{not}\ \mathit{ab} \\ \mathit{bird} & & \end{array}$$

- This program has these three models:

I	P^I	$LM(P^I)$
$\{bird, ab\}$	$bird$	
$\{bird, ab, flies\}$		
$\{bird, flies\}$		

Stable models

- An example of default. Try this program:

$$\begin{aligned} flies &\leftarrow bird, not\ ab \\ bird & \end{aligned}$$

- This program has these three models:

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

Stable models

- An example of default. Try this program:

$$\begin{array}{lcl} flies & \leftarrow & bird, not\ ab \\ bird & & \end{array}$$

- This program has these three models:

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

Stable models

- An example of default. Try this program:

$$
\begin{aligned}
flies &\leftarrow bird, not\ ab \\
bird &
\end{aligned}
$$

- This program has these three models:

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

Stable models

- An example of default. Try this program:

$$\begin{aligned} flies &\leftarrow bird, not\ ab \\ bird & \end{aligned}$$

- This program has these three models:

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

Stable models

- Adding new information:

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

Stable models

- Adding new information:

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

- Just two (classical) models now:

I	P^I	$LM(P^I)$
$\{bird, penguin, ab\}$		
$\{bird, penguin, ab, flies\}$		

Stable models

- Adding new information:

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

- Just two (classical) models now:

I	P^I	$LM(P^I)$
$\{bird, penguin, ab\}$	$bird$ $ab \leftarrow bird, penguin$ $penguin$	
$\{bird, penguin, ab, flies\}$		

Stable models

- Adding new information:

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

- Just two (classical) models now:

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

Stable models

- Adding new information:

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

- Just two (classical) models now:

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

Stable models: some properties

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

$$p \leftarrow not\ q \qquad q \leftarrow not\ p$$

Stable models: some properties

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

$$p \leftarrow not\ q \qquad q \leftarrow not\ p$$

- A program may have no stable model at all. Example P_2:

$$p \leftarrow not\ p$$

Stable models: some properties

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

$$p \leftarrow not\ q \qquad q \leftarrow not\ p$$

- A program may have no stable model at all. Example P_2:

$$p \leftarrow not\ p$$

- Typically use: (1) generate multiple solutions (even cycles like P_1) and (2) prune undesired models (odd cycles like P_2).

Stable models: some properties

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

$$p \leftarrow not\ q \qquad q \leftarrow not\ p$$

- A program may have no stable model at all. Example P_2:

$$p \leftarrow not\ p$$

- Typically use: (1) generate multiple solutions (even cycles like P_1) and (2) prune undesired models (odd cycles like P_2).
- Constraints. Example: to avoid a model where p holds but q doesn't:

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

where aux is a new fresh atom. Usually written: $\leftarrow p, not\ q$

Stable models vs Default Logic

- Very close to Default Logic. A rule like:

$$p \leftarrow q_1, \ldots, q_m, not\ q_{m+1}, \ldots, not\ q_n$$

just corresponds to the default:

Stable models vs Default Logic

- Very close to Default Logic. A rule like:

$$p \leftarrow q_1, \ldots, q_m, not\ q_{m+1}, \ldots, not\ q_n$$

just corresponds to the default:

$$\frac{q_1 \wedge \cdots \wedge q_m : \neg q_{m+1}, \ldots, \neg q_n}{p}$$

- So, it's like playing with defaults where we mostly deal with atoms.

Stable models vs answer sets

- We can sometimes be interested in a second negation, strong or explicit negation (originally called "classical"). Example:

  $$cross \leftarrow not\ train$$

  risky! we cross the railway tracks when no information on train approaching is available. Compare to:

  $$cross \leftarrow -train$$

  we must have the fact that the train is not approaching.

Stable models vs answer sets

- We can sometimes be interested in a second negation, strong or explicit negation (originally called "classical"). Example:

$$cross \leftarrow not\ train$$

  risky! we cross the railway tracks when no information on train approaching is available. Compare to:

$$cross \leftarrow -train$$

  we must have the fact that the train is not approaching.

- We may represent defaults like

$$-raining \leftarrow summer, not\ rainning$$

Stable models vs answer sets

- We can sometimes be interested in a second negation, strong or explicit negation (originally called "classical"). Example:

$$cross \leftarrow not\ train$$

  risky! we cross the railway tracks when no information on train approaching is available. Compare to:

$$cross \leftarrow -train$$

  we must have the fact that the train is not approaching.

- We may represent defaults like

$$-raining \leftarrow summer, not\ rainning$$

- Stable models with strong negation are called answer sets. We just compute stable models and reject those where p, $-p$ occur.

Well-founded model

- Operator Γ_P is antimonotone (on set inclusion). This implies that when applied twice, Γ_P^2, it becomes a monotonic operator.

Well-founded model

- Operator Γ_P is antimonotone (on set inclusion). This implies that when applied twice, Γ_P^2, it becomes a monotonic operator.
- It has a least fixpoint $lfp(\Gamma_P^2)$ and a greatest fixpoint $gfp(\Gamma_P^2)$ that limit the fixpoints of Γ_P (i.e., the stable models) from below and from above.

Well-founded model

- Operator Γ_P is antimonotone (on set inclusion). This implies that when applied twice, Γ_P^2, it becomes a monotonic operator.
- It has a least fixpoint $lfp(\Gamma_P^2)$ and a greatest fixpoint $gfp(\Gamma_P^2)$ that limit the fixpoints of Γ_P (i.e., the stable models) from below and from above.
- The well-founded model (WFM) of P is a three-valued interpretation such that:
  - atoms in $lfp(\Gamma_P^2)$ are called well-founded;
  - atoms not in $gfp(\Gamma_P^2)$ are called unfounded;
  - the rest of atoms would be undefined.

Well-founded model

- Operator Γ_P is antimonotone (on set inclusion). This implies that when applied twice, Γ_P^2, it becomes a monotonic operator.
- It has a least fixpoint $lfp(\Gamma_P^2)$ and a greatest fixpoint $gfp(\Gamma_P^2)$ that limit the fixpoints of Γ_P (i.e., the stable models) from below and from above.
- The well-founded model (WFM) of P is a three-valued interpretation such that:
  - atoms in $lfp(\Gamma_P^2)$ are called well-founded;
  - atoms not in $gfp(\Gamma_P^2)$ are called unfounded;
  - the rest of atoms would be undefined.
- A. Van Gelder, K.A. Ross and J.S. Schlipf. The Well-Founded Semantics for General Logic Programs. Journal of the ACM 38(3) pp. 620—650, 1991.

Well-founded model

- Why is the WFM interesting for stable models? . . .

Well-founded model

- Why is the WFM interesting for stable models? ...

Proposition

Any stable model I of P includes all well-founded atoms, and includes no atom.

Well-founded model

- Why is the WFM interesting for stable models? ...

Proposition

Any stable model I of P includes all well-founded atoms, and includes no atom.

Corollary

If there are no undefined atoms, then $lfp(\Gamma_P^2)$ is the only stable model of P.

Well-founded model

- Why is the WFM interesting for stable models? ...

Proposition

Any stable model I of P includes all well-founded atoms, and includes no atom.

Corollary

If there are no undefined atoms, then $lfp(\Gamma_P^2)$ is the only stable model of P.

- Checking whether P has a stable model is an NP-complete problem [Eiter & Gottlob 93]. Computing $WFM(P)$ takes polynomial time (quadratic).

Well-founded model

- Computing the WFM: when P is finite, we can just iterate Γ^2_P on $\emptyset$.
  Example: try with program P_3

$$\begin{array}{rcl} p & \leftarrow & \mathit{not}\ q \\ r & \leftarrow & p, \mathit{not}\ s \\ s & \leftarrow & \mathit{not}\ r \end{array}$$

Well-founded model

- Computing the WFM: when P is finite, we can just iterate Γ_P^2 on $\emptyset$. Example: try with program P_3

$$\begin{array}{rcl} p & \leftarrow & not\ q \\ r & \leftarrow & p, not\ s \\ s & \leftarrow & not\ r \end{array}$$

and with program P_4

$$\begin{array}{rcl} p & \leftarrow & not\ q \\ r & \leftarrow & p, s \\ s & \leftarrow & r \\ t & \leftarrow & r, not\ t \end{array}$$

Computing WFM: rewritting approach

- An alternative to compute the WFM is using a bottom-up rewritting technique [Brass,Dix,Freitag,Zukowski 2001].

Computing WFM: rewritting approach

- An alternative to compute the WFM is using a bottom-up rewritting technique [Brass,Dix,Freitag,Zukowski 2001].
- Rewritting rules:
  1. Facts: for any fact p in the program remove p from positive bodies, and remove rules containing *not* p in the body.

Computing WFM: rewritting approach

- An alternative to compute the WFM is using a bottom-up rewritting technique [Brass,Dix,Freitag,Zukowski 2001].
- Rewritting rules:
  1. Facts: for any fact *p* in the program remove *p* from positive bodies, and remove rules containing *not p* in the body. We can also remove the rest of rules with head *p*.

Computing WFM: rewritting approach

- An alternative to compute the WFM is using a bottom-up rewritting technique [Brass,Dix,Freitag,Zukowski 2001].
- Rewritting rules:
  1. Facts: for any fact p in the program remove p from positive bodies, and remove rules containing *not* p in the body. We can also remove the rest of rules with head p.
  2. Non-heads: for any atom p not occurring as a rule head, remove all *not* p from bodies and all rules containing p as positive body literal.

Computing WFM: rewritting approach

- An alternative to compute the WFM is using a bottom-up rewritting technique [Brass,Dix,Freitag,Zukowski 2001].
- Rewritting rules:
  1. Facts: for any fact p in the program remove p from positive bodies, and remove rules containing *not* p in the body. We can also remove the rest of rules with head p.
  2. Non-heads: for any atom p not occurring as a rule head, remove all *not* p from bodies and all rules containing p as positive body literal.
  3. "Unreachable atoms" (or positive loop detection) : for any $p \notin \Gamma_P(\emptyset)$ remove all rules containing p as positive body literal.

Computing WFM: rewritting approach

- An alternative to compute the WFM is using a bottom-up rewritting technique [Brass,Dix,Freitag,Zukowski 2001].
- Rewritting rules:
  1. Facts: for any fact p in the program remove p from positive bodies, and remove rules containing *not* p in the body. We can also remove the rest of rules with head p.
  2. Non-heads: for any atom p not occurring as a rule head, remove all *not* p from bodies and all rules containing p as positive body literal.
  3. "Unreachable atoms" (or positive loop detection) : for any $p \notin \Gamma_P(\emptyset)$ remove all rules containing p as positive body literal.
- When we exhaust these rules, we get the program remainder.

Proposition

The facts of the program remainder are the well-founded atoms; the non-head atoms are the unfounded atoms.

Computing WFM: rewritting approach

- Transformations 1 and 2 (i.e. without loop detection) obtain the so-called Fitting's model.

Computing WFM: rewritting approach

- Transformations 1 and 2 (i.e. without loop detection) obtain the so-called Fitting's model.
- Try the program P_5

$$\begin{array}{lcl} a & \leftarrow & not\ b, c \\ b & \leftarrow & not\ a \\ c & & \end{array} \qquad \begin{array}{lcl} d & \leftarrow & not\ g, e \\ e & \leftarrow & not\ g, d \\ f & \leftarrow & not\ d \end{array} \qquad \begin{array}{lcl} g & \leftarrow & not\ c \\ h & \leftarrow & g \end{array}$$

- Fitting's model=$\langle\{c\},\{g,h\}\rangle$. The final program remainder is:

$$\begin{array}{lcl} a & \leftarrow & not\ b \\ b & \leftarrow & not\ a \end{array} \qquad \begin{array}{l} c \\ f \end{array}$$

and so, $WFM = \langle\{c,f\},\{g,h,d,e\}\rangle$.

Computing WFM: rewritting approach

- Transformations 1 and 2 (i.e. without loop detection) obtain the so-called Fitting's model.
- Try the program P_5

$$\begin{array}{lcl} a & \leftarrow & not\ b, c \\ b & \leftarrow & not\ a \\ c & & \end{array} \qquad \begin{array}{lcl} d & \leftarrow & not\ g, e \\ e & \leftarrow & not\ g, d \\ f & \leftarrow & not\ d \end{array} \qquad \begin{array}{lcl} g & \leftarrow & not\ c \\ h & \leftarrow & g \end{array}$$

- Fitting's model=$\langle \{c\}, \{g, h\} \rangle$. The final program remainder is:

$$\begin{array}{lcl} a & \leftarrow & not\ b \\ b & \leftarrow & not\ a \end{array} \qquad \begin{array}{l} c \\ f \end{array}$$

and so, $WFM = \langle \{c, f\}, \{g, h, d, e\} \rangle$. Stable models $\{c, f, a\}$ and $\{c, f, b\}$.

ASP

- Most ASP solvers (DLV, smodels, clasp) alternate computation of WFM and nondeterministic choice with backtracking.
  1. Compute the WFM
  2. If no undefined atoms: stable model found.
  3. Else: select an undefined atom *p* (using some heuristics) and branch: *p*; *not p*. Simplify the program accordingly to the choice and go to 1.

Clark's completion

- Idea: the only way of making a predicate true is through its defining rules.

Clark's completion

- Idea: the only way of making a predicate true is through its defining rules.
- Example:

```
p :- q, not r.
p :- s.
```

Clark's completion

- Idea: the only way of making a predicate true is through its defining rules.
- Example:

```
p :- q, not r.
p :- s.
```

This program classically implies $p \leftarrow (q \wedge \neg r) \vee s$.

Clark's completion

- Idea: the only way of making a predicate true is through its defining rules.
- Example:

```
p :- q, not r.
p :- s.
```

This program classically implies $p \leftarrow (q \wedge \neg r) \vee s$. But atom `p` can only be true when body $q \wedge \neg r$ or body s become true. That is we complete the other direction of implication $p \rightarrow (q \wedge \neg r) \vee s$.

Clark's completion

- Idea: the only way of making a predicate true is through its defining rules.

- Example:

```
p :- q, not r.
p :- s.
```

  This program classically implies $p \leftarrow (q \wedge \neg r) \vee s$. But atom `p` can only be true when body $q \wedge \neg r$ or body s become true. That is we complete the other direction of implication $p \rightarrow (q \wedge \neg r) \vee s$.

- [K. L. Clark 1978] $COMP[P]$ is the classical theory consisting of:

$$p \leftrightarrow B_1 \vee \cdots \vee B_n$$

  for each atom p, and all rules $p \leftarrow B_i$ in P.

Clark's completion

- Idea: the only way of making a predicate true is through its defining rules.
- Example:

```
p :- q, not r.
p :- s.
```

This program classically implies $p \leftarrow (q \wedge \neg r) \vee s$. But atom `p` can only be true when body $q \wedge \neg r$ or body s become true. That is we complete the other direction of implication $p \rightarrow (q \wedge \neg r) \vee s$.

- [K. L. Clark 1978] $COMP[P]$ is the classical theory consisting of:

$$p \leftrightarrow B_1 \vee \cdots \vee B_n$$

for each atom p, and all rules $p \leftarrow B_i$ in P.
An empty disjunction is $\bot$.

Clark's completion

- Example: let *P* be the program

```
p :- q, not r.
p :- s.
t.
q :- t
```

Clark's completion

- Example: let P be the program

```
p :- q, not r.
p :- s.
t.
q :- t
```

$COMP[P]$ consists of the equivalences:

$$\begin{aligned} p &\leftrightarrow (q \wedge \neg r) \vee s \\ q &\leftrightarrow t \\ r &\leftrightarrow \bot \\ s &\leftrightarrow \bot \\ t &\leftrightarrow \top \end{aligned}$$

Clark's completion

- Example: let P be the program

```
p :- q, not r.
p :- s.
t.
q :- t
```

$COMP[P]$ consists of the equivalences:

$$
\begin{array}{lcl}
p & \leftrightarrow & (q \wedge \neg r) \vee s \\
q & \leftrightarrow & t \\
r & \leftrightarrow & \bot \\
s & \leftrightarrow & \bot \\
t & \leftrightarrow & \top
\end{array}
$$

whose only model is $\{p, q, t\}$.

Clark's completion

- Semantic counterpart: supported models.

Definition (Supported model)

I is a supported model of P iff $I = T_P(I)$.

Clark's completion

- Semantic counterpart: supported models.

Definition (Supported model)

I is a supported model of P iff $I = T_P(I)$.

- That is I is a fixpoint of T_P, i.e.
  $I = \{p \mid (p \leftarrow B) \in P, I \models B\}$

Clark's completion

- Semantic counterpart: supported models.

Definition (Supported model)

I is a supported model of P iff $I = T_P(I)$.

- That is I is a fixpoint of T_P, i.e.
  $I = \{p \mid (p \leftarrow B) \in P, I \models B\}$
- In the example:

```
p :- q, not r.
p :- s.
t.
q :- t
```

$T_P(\{p, q, t\}) = \{p, q, t\}$ (supported), whereas, for instance $T_P(\{p, s\}) = \{p, t\}$ (non-supported).

Clark's completion

Theorem

I is a supported model of P iff $I \models COMP[P]$.

- Exercise: prove it.

Clark's completion

Theorem

I is a supported model of P iff $I \models COMP[P]$.

- Exercise: prove it.
- In the previous example, $\{p, q, t\}$ happens to be the only stable model. What happens with these typical examples?

$$p \leftarrow not\ q$$
$$q \leftarrow not\ p$$

and

$$p \leftarrow not\ p$$

Clark's completion

Theorem

I is a supported model of P iff $I \models COMP[P]$.

- Exercise: prove it.
- In the previous example, $\{p, q, t\}$ happens to be the only stable model. What happens with these typical examples?

$$p \leftarrow not\ q$$
$$q \leftarrow not\ p$$

and

$$p \leftarrow not\ p$$

- Yes, supported and stable models coincide! Is this general?

Clark's completion

- No: they differ in positive cycles. Consider this extremely simple example:

$$p \leftarrow p$$

Completion would be $p \leftrightarrow p$ which has two supported models, $\emptyset$ and $\{p\}$. Only $\emptyset$ is stable.

Clark's completion

- No: they differ in positive cycles. Consider this extremely simple example:

$$p \leftarrow p$$

Completion would be $p \leftrightarrow p$ which has two supported models, $\emptyset$ and $\{p\}$. Only $\emptyset$ is stable.

Theorem

Any stable model is supported.

- Prove it.

Clark's completion

- No: they differ in positive cycles. Consider this extremely simple example:

$$p \leftarrow p$$

Completion would be $p \leftrightarrow p$ which has two supported models, $\emptyset$ and $\{p\}$. Only $\emptyset$ is stable.

Clark's completion

- No: they differ in positive cycles. Consider this extremely simple example:

$$p \leftarrow p$$

Completion would be $p \leftrightarrow p$ which has two supported models, $\emptyset$ and $\{p\}$. Only $\emptyset$ is stable.

Theorem

Any stable model is supported.

- Prove it.

Clark's completion

- If we don't have these positive cycles, we can define a quite general class of programs where supported and stable coincide.

Clark's completion

- If we don't have these positive cycles, we can define a quite general class of programs where supported and stable coincide.
- Tight programs [Fages 1994][Babovich, Erdem, Lifschitz 2000].

Definition (Tight programs)

P is tight on a set I of atoms if there is some partial ordinal mapping $\lambda : X \rightarrow N$ such that all $\lambda(B_i) < \lambda(H)$ for any rule in P like:

$$H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$$

Clark's completion

- If we don't have these positive cycles, we can define a quite general class of programs where supported and stable coincide.
- Tight programs [Fages 1994][Babovich, Erdem, Lifschitz 2000].

Definition (Tight programs)

P is tight on a set I of atoms if there is some partial ordinal mapping $\lambda : X \rightarrow N$ such that all $\lambda(B_i) < \lambda(H)$ for any rule in P like:

$$H \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$$

Theorem

If I is supported model of P and P tight on I, then I is stable model of P.

Clark's completion

- Example:

$$p \leftarrow not\ q \qquad q \leftarrow not\ p \qquad r \leftarrow r \qquad p \leftarrow r$$

the completion

$$p \leftrightarrow \neg q \vee r \qquad q \leftrightarrow \neg p \qquad r \leftrightarrow r$$

has models $\{p\}$, $\{q\}$ and $\{p, r\}$, but $\{p, r\}$ is not tight - only the first 2 ones are stable.

Loop formulas

- We can strengthen completion to obtain stable models by adding loop formulas [Lin , Zhao 2004].

Loop formulas

- We can strengthen completion to obtain stable models by adding loop formulas [Lin , Zhao 2004].
- We define a positive dependency graph G with vertices $V = \Sigma$ and edges E, one (p, q) for each rule $p \leftarrow B$ with q in the positive body.

Loop formulas

- We can strengthen completion to obtain stable models by adding loop formulas [Lin , Zhao 2004].
- We define a positive dependency graph G with vertices $V = \Sigma$ and edges E, one (p, q) for each rule $p \leftarrow B$ with q in the positive body.
- A loop L is a set of atoms forming a Strongly Connected Component in G.

Loop formulas

- We can strengthen completion to obtain stable models by adding loop formulas [Lin , Zhao 2004].
- We define a positive dependency graph G with vertices $V = \Sigma$ and edges E, one (p, q) for each rule $p \leftarrow B$ with q in the positive body.
- A loop L is a set of atoms forming a Strongly Connected Component in G.
- Given a loop $L = \{p_1, \ldots, p_n\}$ its loop formula $LF(L)$ is defined as:

$$\neg(BB_1 \vee \cdots \vee BB_n) \rightarrow \neg p_1 \wedge \cdots \wedge \neg p_n$$

where BB_i is the disjunction $B_1 \vee \cdots \vee B_{m_i}$ of all bodies for rules in P like

$$p_i \leftarrow B_j$$

such that not atom in L occurs in the positive body of B_j.

Loop formulas

- Example:

$$a \leftarrow b \qquad b \leftarrow a \qquad a \leftarrow not\ c$$
$$c \leftarrow d \qquad d \leftarrow c \qquad c \leftarrow not\ a$$

Loop formulas

- Example:

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

Completion $COMP[P]$ is:

$$a \leftrightarrow \neg c \vee b \qquad b \leftrightarrow a \qquad c \leftrightarrow \neg a \vee d \qquad d \leftrightarrow c$$

has 3 models $\{a, b\}$, $\{c, d\}$, $\{a, b, c, d\}$.

Loop formulas

- Example:

$$a \leftarrow b \qquad b \leftarrow a \qquad a \leftarrow not\ c$$
$$c \leftarrow d \qquad d \leftarrow c \qquad c \leftarrow not\ a$$

Completion $COMP[P]$ is:

$$a \leftrightarrow \neg c \vee b \qquad b \leftrightarrow a \qquad c \leftrightarrow \neg a \vee d \qquad d \leftrightarrow c$$

has 3 models $\{a, b\}$, $\{c, d\}$, $\{a, b, c, d\}$.

- Loops $L_1 = \{a, b\}$ and $L_2 = \{c, d\}$. Loop Formulas:

$$LF(L_1) : c \rightarrow \neg a \wedge \neg b$$
$$LF(L_2) : a \rightarrow \neg c \wedge \neg d$$

Adding them to $COMP[P]$ leaves $\{a, b\}$ and $\{c, d\}$ as only stable models.

Introducing variables

- Programs with variables in ASP are understood as abbreviations of their ground cases.

Introducing variables

- Programs with variables in ASP are understood as abbreviations of their ground cases.
- Keypoint: use of functions was typically forbidden. The introduction of a function f makes the Herbrand universe infinite $f(c), f(f(c)), f(f(f(c))), \ldots$.

Introducing variables

- Programs with variables in ASP are understood as abbreviations of their ground cases.
- Keypoint: use of functions was typically forbidden. The introduction of a function f makes the Herbrand universe infinite $f(c), f(f(c)), f(f(f(c))), \ldots$.
- This restriction is being overcome:
  - `lparse` allows functors, but their nesting is limited (no lists, for instance).

Introducing variables

- Programs with variables in ASP are understood as abbreviations of their ground cases.
- Keypoint: use of functions was typically forbidden. The introduction of a function f makes the Herbrand universe infinite $f(c), f(f(c)), f(f(f(c))), \dots$.
- This restriction is being overcome:
  - `lparse` allows functors, but their nesting is limited (no lists, for instance).
  - More recently, `DLV complex` allows functions (lists, sets, etc) for finitely ground programs, a class of programs with finitely many answer sets that are finite.

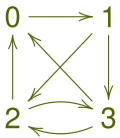
Introducing variables

- A simple example: Hamiltonian circuits. Find a cyclic path that visits once each node in a graph.

Introducing variables

- A simple example: Hamiltonian circuits. Find a cyclic path that visits once each node in a graph.
- We have the extensional database describing the graph

```
node(0).  node(1).  node(2).  node(3).
edge(0,1).  edge(1,2).  edge(1,3).
edge(2,0).  edge(2,3).  edge(3,2).  edge(3,0).
```



Introducing variables

- Predicate `in(X,Y)` points out that and edge $\texttt{X} \rightarrow \texttt{Y}$ is in the cycle.

Introducing variables

- Predicate `in(X,Y)` points out that and edge $X \longrightarrow Y$ is in the cycle. We generate arbitrary choices with an auxiliary predicate `out`.

```
in(X,Y)  :- edge(X,Y), not out(X,Y).
out(X,Y) :- edge(X,Y), not in(X,Y).
```

Introducing variables

- Predicate `in(X,Y)` points out that and edge `X` $\longrightarrow$ `Y` is in the cycle. We generate arbitrary choices with an auxiliary predicate `out`.

```
in(X,Y)  :- edge(X,Y), not out(X,Y).
out(X,Y) :- edge(X,Y), not in(X,Y).
```

- Only one outgoing node, only one incoming node:

```
:- in(X,Y), in(X,Z), Y!=Z.
:- in(X,Z), in(Y,Z), X!=Y.
```

Introducing variables

- Predicate `in(X,Y)` points out that and edge `X` $\longrightarrow$ `Y` is in the cycle. We generate arbitrary choices with an auxiliary predicate `out`.

```
in(X,Y)  :- edge(X,Y), not out(X,Y).
out(X,Y) :- edge(X,Y), not in(X,Y).
```

- Only one outgoing node, only one incoming node:

```
:- in(X,Y), in(X,Z), Y!=Z.
:- in(X,Z), in(Y,Z), X!=Y.
```

- Disregard disconnected cycles. We use a predicate `reached(X)` meaning that `X` can be reached from an arbitrary fixed node, say 0.

```
reached(X) :- in(0,X).
reached(Y) :- reached(X), in(X,Y).
```

Introducing variables

- Predicate `in(X,Y)` points out that and edge $X \longrightarrow Y$ is in the cycle. We generate arbitrary choices with an auxiliary predicate `out`.

```
in(X,Y) :- edge(X,Y), not out(X,Y).
out(X,Y) :- edge(X,Y), not in(X,Y).
```

- Only one outgoing node, only one incoming node:

```
:- in(X,Y), in(X,Z), Y!=Z.
:- in(X,Z), in(Y,Z), X!=Y.
```

- Disregard disconnected cycles. We use a predicate `reached(X)` meaning that `X` can be reached from an arbitrary fixed node, say 0.

```
reached(X) :- in(0,X).
reached(Y) :- reached(X), in(X,Y).
```

  and we forbid unreached nodes:

```
:- node(X), not reached(X)
```

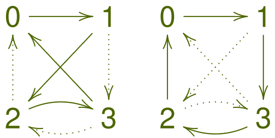
Introducing variables

- Making the call:

  ```
  lparse -n 0 hamilt.txt | smodels
  ```

  We obtain two answers:

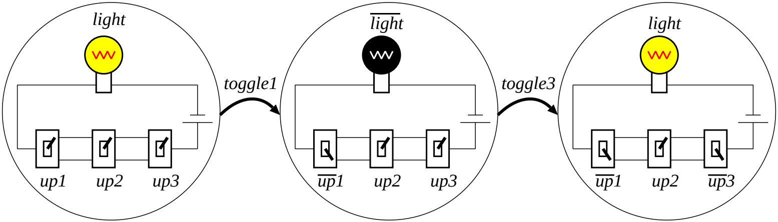
  ```
  Answer: 1
  Stable Model: in(0,1) in(3,0) in(2,3) in(1,2)
  Answer: 2
  Stable Model: in(0,1) in(3,2) in(2,0) in(1,3)
  False
  ```



Answer 1 Answer 2

Reasoning about actions with ASP

- An example of action domain.



Reasoning about actions with ASP

- We begin with some "type declarations".

```
time(0..pathlength).
previoustime(0..pathlength-1).
switch(1..3).
#domain previoustime(I).
#domain time(J).
#domain switch(X).
#domain switch(Y).
```

Reasoning about actions with ASP

```
% Effect axioms
up(X,true,I+1)    :- up(X,false,I), toggle(X,I).
up(X,false,I+1)   :- up(X,true,I), toggle(X,I).
light(true,I+1)   :- light(false,I), toggle(X,I).
light(false,I+1) :- light(true,I), toggle(X,I).

% Inertia
up(X,true,I+1)   :- up(X,true,I), not up(X,false,I+1).
up(X,false,I+1) :- up(X,false,I), not up(X,true,I+1).

light(true,I+1)   :- light(true,I),
                     not light(false,I+1).
light(false,I+1) :- light(false,I),
                     not light(true,I+1).
```

Reasoning about actions with ASP

```
% Constraints: unique value
:- up(X,true,J), up(X,false,J).
:- light(true,J), light(false,J).

% Unique action
:- toggle(X,I), toggle(Y,I), X!=Y.
```

Reasoning about actions with ASP

Prediction example

```
% switches-predict.txt
% Initial state
light(true,0).     up(X,true,0).

% Performed actions
toggle(1,0).
```

Calling lparse/smodels with

```
lparse -c pathlength=1 switches.txt
   switches-predict.txt | smodels
```

we get ...

```
Answer: 1
Stable Model: up(1,false,1) up(2,true,1) up(3,true,1)
light(false,1) ...
```

Reasoning about actions with ASP

Postdiction example:

```
% switches-postdict.txt

% Actions generation
1 { toggle(Z,I) : switch(Z) } 1.
       % generate 1 toggle among all switches Z

% Completing facts about the initial situation
1 {up(X,true,0), up(X,false,0)} 1.
1 {light(true,0), light(false,0)} 1.

% Observations
up(3,true,0). light(true,0). toggle(3,1).
light(false,1). up(1,false,1). up(3,true,1).
```

Reasoning about actions with ASP

Calling lparse with

```
lparse -c pathlength=1 -n 0 switches.txt
   switches-postdict.txt | smodels
```

we get 6 possible explanations. One of them:

```
Answer: 1
Stable Model: toggle(1,0) up(2,false,0) up(1,true,0)
up(2,false,1) ...
```

Reasoning about actions with ASP

Planning example

```
% switches-plan.txt
% Planning problem

% Actions generation
1 { toggle(Z,I) : switch(Z) } 1.

% Initial state
light(true,0). up(X,true,0).

% Goal state
goal :- light(true,pathlength),up(1,false,pathlength),
        up(2,true,pathlength), up(3,false,pathlength).

:- not goal.
```

Reasoning about actions with ASP

Calling lparse with

```
lparse -c pathlength=1 -n 0 switches.txt
   switches-plan.txt | smodels
```

We don't get models. After increasing `pathlength`

```
lparse -c pathlength=2 -n 0 switches.txt
   switches-plan.txt | smodels
```

we get 2 possible plans

```
Answer: 1
Stable Model: toggle(1,0) toggle(3,1) ...
Answer: 2
Stable Model: toggle(3,0) toggle(1,1) ...
```

Extending the syntax

- Disjunctive programs: bodies B as before, but heads allow disjunctions of atoms:

$$p_1 \vee \cdots \vee p_n \leftarrow B$$

Extending the syntax

- Disjunctive programs: bodies B as before, but heads allow disjunctions of atoms:

$$p_1 \vee \cdots \vee p_n \leftarrow B$$

- The reduct is defined as before, but note that P^I does not have now a least Herbrand model: only minimal ones.

Extending the syntax

- Disjunctive programs: bodies B as before, but heads allow disjunctions of atoms:

$$p_1 \vee \cdots \vee p_n \leftarrow B$$

- The reduct is defined as before, but note that P^I does not have now a least Herbrand model: only minimal ones. Example:

$$p \vee q \leftarrow t, not\ s \qquad\qquad t \leftarrow not\ q$$

Given $I = \{p, t\}$, P^I is the program:

$$p \vee q \leftarrow t \qquad\qquad t \leftarrow$$

whose minimal models are $\{p, t\}$ (stable) and $\{q, t\}$ (non-stable).

Extending the syntax

- The definition is adapted accordingly

Definition (stable model)

I is a stable model of a disjunctive program P if it is a minimal model of P^I.

Extending the syntax

- The definition is adapted accordingly

Definition (stable model)

I is a stable model of a disjunctive program P if it is a minimal model of P^I.

- Finding a stable model of a disjunctive program is slightly more complex: Σ_2^P-complete.

Extending the syntax

- The definition is adapted accordingly

Definition (stable model)

I is a stable model of a disjunctive program P if it is a minimal model of P^I.

- Finding a stable model of a disjunctive program is slightly more complex: Σ_2^P-complete.
- Tools for disjunctive ASP: `DLV`, `GnT`, `cmodels`.

Extending the syntax

- Adding default negation in the head [Inoue & Sakama 98]. Rules $H \leftarrow B$ where:
  1. Body B = conjunction of literals (as before).

Extending the syntax

- Adding default negation in the head [Inoue & Sakama 98]. Rules $H \leftarrow B$ where:
  1. Body B = conjunction of literals (as before). We define:

$$B = \underbrace{q_1 \wedge \cdots \wedge q_n}_{B^+} \wedge \underbrace{not\ q_{n+1} \wedge \cdots \wedge not\ q_m}_{B^-}$$

Extending the syntax

- Adding default negation in the head [Inoue & Sakama 98]. Rules $H \leftarrow B$ where:
  1. Body B = conjunction of literals (as before). We define:

$$B = \underbrace{q_1 \wedge \cdots \wedge q_n}_{B^+} \wedge \underbrace{not\ q_{n+1} \wedge \cdots \wedge not\ q_m}_{B^-}$$

  2. Head H = disjunction of literals.

Extending the syntax

- Adding default negation in the head [Inoue & Sakama 98]. Rules $H \leftarrow B$ where:
  1. Body B = conjunction of literals (as before). We define:

$$B = \underbrace{q_1 \wedge \cdots \wedge q_n}_{B^+} \wedge \underbrace{not\ q_{n+1} \wedge \cdots \wedge not\ q_m}_{B^-}$$

  2. Head H = disjunction of literals. We define:

$$H = \underbrace{p_1 \vee \cdots \vee p_k}_{H^+} \vee \underbrace{not\ p_{k+1} \vee \cdots \vee not\ p_s}_{H^-}$$

Extending the syntax

- We adapt the definition of reduct as follows:

$$P^I = \{H^+ \leftarrow B^+ \mid I \models B^- \wedge \neg H^-\}$$

Extending the syntax

- We adapt the definition of reduct as follows:

$$P^I = \{H^+ \leftarrow B^+ \mid I \models B^- \wedge \neg H^-\}$$

- Example: given $I = \{a, d, c\}$ and program

$$b \vee not\ a \vee not\ d \quad \leftarrow \quad d, not\ e, not\ h$$

$B^- = \neg e \wedge \neg h$ and $\neg H^- = \neg(\neg a \vee \neg d) = (a \wedge d)$.
As $I \models B^- \wedge \neg H^-$, its reduct would correspond to:

$$b \quad \leftarrow \quad h$$

Extending the syntax

- Stable models are defined as before: I minimal model of P^I.

Extending the syntax

- Stable models are defined as before: I minimal model of P^I.
- Example: P is the program

$$p \vee \neg p$$
$$q \leftarrow \neg p$$

I	P^I	minimal models
$\emptyset$		
$\{p\}$		
$\{q\}$		
$\{p, q\}$		

Extending the syntax

- Stable models are defined as before: I minimal model of P^I.
- Example: P is the program

$$p \vee \neg p$$
$$q \leftarrow \neg p$$

I	P^I	minimal models
$\emptyset$	q	$\{q\} \neq I$ not stable
$\{p\}$		
$\{q\}$		
$\{p, q\}$		

Extending the syntax

- Stable models are defined as before: I minimal model of P^I.
- Example: P is the program

$$p \vee \neg p$$
$$q \leftarrow \neg p$$

I	P^I	minimal models
$\emptyset$	q	$\{q\} \neq I$ not stable
$\{p\}$	p	$\{p\} \neq I$ stable!
$\{q\}$		
$\{p, q\}$		

Extending the syntax

- Stable models are defined as before: I minimal model of P^I.
- Example: P is the program

$$p \vee \neg p$$
$$q \leftarrow \neg p$$

I	P^I	minimal models
$\emptyset$	q	$\{q\} \neq I$ not stable
$\{p\}$	p	$\{p\} \neq I$ stable!
$\{q\}$	q	$\{q\}$ stable!
$\{p, q\}$		

Extending the syntax

- Stable models are defined as before: I minimal model of P^I.
- Example: P is the program

$$p \vee \neg p$$
$$q \leftarrow \neg p$$

I	P^I	minimal models
$\emptyset$	q	$\{q\} \neq I$ not stable
$\{p\}$	p	$\{p\} \neq I$ stable!
$\{q\}$	q	$\{q\}$ stable!
$\{p, q\}$	p	$\{p\}$ not stable

Extending the syntax

- Nested expressions [Lifschitz, Tang, Turner 99]:
  H and B can be any combination of atoms with $\bot, \top, \wedge, \vee,$ *not* .

Extending the syntax

- Nested expressions [Lifschitz, Tang, Turner 99]:
  H and B can be any combination of atoms with $\bot, \top, \wedge, \vee,$ *not* .
- Several transformation rules (we'll see later) allow reducing nested expressions to disjunctive programs with negation in the head.

Extending the syntax

- Nested expressions [Lifschitz, Tang, Turner 99]:
  H and B can be any combination of atoms with $\bot, \top, \wedge, \vee, \mathit{not}$.
- Several transformation rules (we'll see later) allow reducing nested expressions to disjunctive programs with negation in the head.
- An example: the nested rule

$$a \vee \mathit{not}\ (b \wedge \mathit{not}\ c) \leftarrow d \vee \mathit{not}\ e$$

  becomes the program:

$$\begin{array}{lcl} a \vee \mathit{not}\ b & \leftarrow & d \wedge \mathit{not}\ c \\ a \vee \mathit{not}\ b & \leftarrow & \mathit{not}\ e \wedge \mathit{not}\ c \end{array}$$

Extending the syntax

- Nested expressions [Lifschitz, Tang, Turner 99]:
  H and B can be any combination of atoms with $\bot, \top, \wedge, \vee, not$.
- Several transformation rules (we'll see later) allow reducing nested expressions to disjunctive programs with negation in the head.
- An example: the nested rule

  $$a \vee not\ (b \wedge not\ c) \leftarrow d \vee not\ e$$

  becomes the program:

  $$\begin{array}{rcl} a \vee not\ b & \leftarrow & d \wedge not\ c \\ a \vee not\ b & \leftarrow & not\ e \wedge not\ c \end{array}$$

- But, which is the semantics for $not\ (a \leftarrow b)$ or $a \leftarrow (b \leftarrow c)$?

Equilibrium Logic

- Let us write rules like $p \leftarrow q, \mathit{not}\ r$ in standard logical notation
  $q \wedge \neg r \rightarrow p$

Equilibrium Logic

- Let us write rules like $p \leftarrow q, not\ r$ in standard logical notation $q \wedge \neg r \rightarrow p$
- Equilibrium Logic [Pearce96]: generalises Answer Sets for arbitrary propositional theories.

Equilibrium Logic

- Let us write rules like $p \leftarrow q, not\ r$ in standard logical notation $q \wedge \neg r \rightarrow p$
- Equilibrium Logic [Pearce96]: generalises Answer Sets for arbitrary propositional theories.
- Consists of:
  1. A non-classical monotonic (intermediate) logic called Here-and-There (HT)

Equilibrium Logic

- Let us write rules like $p \leftarrow q, not\ r$ in standard logical notation $q \wedge \neg r \rightarrow p$
- Equilibrium Logic [Pearce96]: generalises Answer Sets for arbitrary propositional theories.
- Consists of:
  1. A non-classical monotonic (intermediate) logic called Here-and-There (HT)
  2. A selection of (certain) minimal models that yields nonmonotonicity

Here-and-There

- Interpretation = pairs $\langle H, T \rangle$ of sets of atoms $H \subseteq T$

Here-and-There

- Interpretation = pairs $\langle H, T \rangle$ of sets of atoms $H \subseteq T$
- Intuition: H= true atoms, T = non-false. When $H = T$ we have a classical model.

Here-and-There

- Interpretation = pairs $\langle H, T \rangle$ of sets of atoms $H \subseteq T$
- Intuition: H= true atoms, T = non-false. When $H = T$ we have a classical model.
- Satisfaction of formulas
  - $\langle H, T \rangle \models p$ if $p \in H$ (for any atom p)
  - $\wedge, \vee$ as always
  - $\langle H, T \rangle \models \varphi \rightarrow \psi$ if both
    - $\langle H, T \rangle \models \varphi$ implies $\langle H, T \rangle \models \psi$
    - $\langle T, T \rangle \models \varphi$ implies $\langle T, T \rangle \models \psi$

Here-and-There

- Interpretation = pairs $\langle H, T\rangle$ of sets of atoms $H \subseteq T$
- Intuition: H= true atoms, T = non-false. When $H = T$ we have a classical model.
- Satisfaction of formulas
  - $\langle H, T\rangle \models p$ if $p \in H$ (for any atom p)
  - $\wedge, \vee$ as always
  - $\langle H, T\rangle \models \varphi \rightarrow \psi$ if both
    - $\langle H, T\rangle \models \varphi$ implies $\langle H, T\rangle \models \psi$
    - $\langle T, T\rangle \models \varphi$ implies $\langle T, T\rangle \models \psi$

      This is the same than $T \models \varphi \rightarrow \psi$ in classical logic.
  - Negation $\neg F$ is defined as $F \rightarrow \bot$

Here-and-There

Some properties

- $\langle T, T \rangle \models \Gamma$ is the same than $T \models \Gamma$ in classical logic.

Here-and-There

Some properties

- $\langle T, T \rangle \models \Gamma$ is the same than $T \models \Gamma$ in classical logic.
- $\langle H, T \rangle \models \Gamma$ implies $T \models \Gamma$.

Here-and-There

Some properties

- $\langle T, T \rangle \models \Gamma$ is the same than $T \models \Gamma$ in classical logic.
- $\langle H, T \rangle \models \Gamma$ implies $T \models \Gamma$.
- $\langle H, T \rangle \models \neg\varphi$ iff $T \not\models \varphi$ in classical logic.

Equilibrium Logic

- Possible alternative description using 3-valued semantics (Gödel's logic G_3).

Equilibrium Logic

- Possible alternative description using 3-valued semantics (Gödel's logic G_3).
- Given $M = \langle H, T \rangle$, we can define a 3-valued mapping $M : Atoms \mapsto \{0, 1, 2\}$ reading:

  2 = $(p \in H)$ = true
  0 = $(p \notin T)$ = false
  1 = $(p \in T \setminus H)$ = undefined

Equilibrium Logic

- Possible alternative description using 3-valued semantics (Gödel's logic G_3).
- Given $M = \langle H, T \rangle$, we can define a 3-valued mapping $M : Atoms \mapsto \{0, 1, 2\}$ reading:

  2 = $(p \in H)$ = true
  0 = $(p \notin T)$ = false
  1 = $(p \in T \setminus H)$ = undefined

- $\wedge$ returns minimum value, $\vee$ returns maximum and $M(\phi \rightarrow \psi) = 2$ if $M(\phi) \leq M(\psi)$ or returns $M(\psi)$ otherwise.

Equilibrium models

Definition (Equilibrium model)

$\langle T, T \rangle$ is an equilibrium model of a theory Γ if:
$\langle T, T \rangle \models \Gamma$, and there is no $H \subset T$ such that $\langle H, T \rangle \models \Gamma$.

Equilibrium Logic

- Logical techniques available: e.g., methods from many-valued semantics (tableaux, signed logics,. . .)

Equilibrium Logic

- Logical techniques available: e.g., methods from many-valued semantics (tableaux, signed logics,. . .)
- Captures all previous syntax extensions, plus other non-propositional constructions:
  - weight constraints can be represented as nested expressions [Ferraris, Lifschitz 2005];
  - aggregates represented by rules with embedded implications [Ferraris 2004].
  - ordered disjunction from [Brewka et al 2004] (LPOD) can also be captured [Cabalar 2010].

Equilibrium logic

Other interesting features

- In nonmonotonic reasoning, we talk about strong equivalence of Γ_1, Γ_2 when, for any Π:
  $\Gamma_1 \cup \Pi$ and $\Gamma_2 \cup \Pi$ have the same (selected) models.

Equilibrium logic

Other interesting features

- In nonmonotonic reasoning, we talk about strong equivalence of Γ_1, Γ_2 when, for any Π:
  $\Gamma_1 \cup \Pi$ and $\Gamma_2 \cup \Pi$ have the same (selected) models.
- Γ_1, Γ_2 are strongly equivalent iff they are equivalent in HT [Lifschitz et al 2001].

Equilibrium logic

Other interesting features

- Disjunctive programs with negation in the head are a (conjunctive) normal form (CNF) for Equilibrium Logic. [Cabalar & Ferraris 2007].

Theorem

The number of different logic programs (modulo strong equivalence) that can be built for a finite signature of n atoms is:

$$\prod_{i=0}^{n} \left(2^{2^{i}-1}+1\right)^{\binom{n}{i}}$$

With $n = 2$ we get 162, with $n = 3$ around 5 million.

Equilibrium logic

Other interesting features

- Disjunctive programs with negation in the head are a (conjunctive) normal form (CNF) for Equilibrium Logic. [Cabalar & Ferraris 2007].

Theorem

The number of different logic programs (modulo strong equivalence) that can be built for a finite signature of n atoms is:

$$\prod_{i=0}^{n} \left(2^{2^i-1}+1\right)^{\binom{n}{i}}$$

With $n = 2$ we get 162, with $n = 3$ around 5 million.

- Transformations into this CNF [Cabalar, Pearce & Valverde 2005].

Equilibrium logic

Other interesting features

- Equilibrium Logic also covers full First Order Theories with equality [Pearce & Valverde 2004].

Equilibrium logic

Other interesting features

- Equilibrium Logic also covers full First Order Theories with equality [Pearce & Valverde 2004].
- Introduction of partial functions [Cabalar 2008].

Equilibrium logic

Other interesting features

- Equilibrium Logic also covers full First Order Theories with equality [Pearce & Valverde 2004].
- Introduction of partial functions [Cabalar 2008].
- Linear temporal equilibrium logic [Cabalar & Pérez 2007].

Equilibrium logic

Other interesting features

- Equilibrium Logic also covers full First Order Theories with equality [Pearce & Valverde 2004].
- Introduction of partial functions [Cabalar 2008].
- Linear temporal equilibrium logic [Cabalar & Pérez 2007].
- Equivalent to the extension of reduct [Ferraris 2005] for arbitrary propositional theories, and general stable model [Ferraris, Lee & Lifschitz 2007] for first order theories.

Minimal logic programs