Theoretical Foundations of Logic Programming

Mirosław Truszczyński

Department of Computer Science
University of Kentucky

July 24-27, 2008
Some logic terminology

Language

- **Constant, variable, function and predicate** symbols
- **Terms**: strings built recursively from constant, variable and function symbols
  - $c, X, f(c, X), f(f(c, X), f(X, f(X, c)))$
- **Atoms**: built of predicate symbols and terms
  - $p(X, c, f(a, Y))$
**Horn logic programming**

**Horn clause**

- \( p \leftarrow q_1, \ldots, q_k \)
  - where \( p, q_i \) are atoms
- Clauses are *universally* quantified
  - special sentences
- Intuitive reading: if \( q_1, \ldots, q_k \) then \( p \)

**Horn program**

- A collection of Horn clauses
Horn logic programming

Horn clause

- $p \leftarrow q_1, \ldots, q_k$
  - where $p, q_i$ are atoms
- Clauses are *universally* quantified
  - special sentences
- Intuitive reading: if $q_1, \ldots, q_k$ then $p$

Horn program

- A collection of Horn clauses
More terminology

Herbrand model

- **Ground terms**: no variable symbols
- **Herbrand universe**: collection of all ground terms
- **Ground atoms**: atoms built of predicate symbols and ground terms
  
  \[ p(a, c, f(a, a)) \]

- **Herbrand base**: collection of all ground atoms
- **Herbrand model**: subset of an Herbrand base
Given by Herbrand models

\[ \text{grnd}(P): \text{the set of all ground instances of clauses in } P \]

Thus, no difference between \( P \) and \( \text{grnd}(P) \)

Key question:

which ground facts hold in every Herbrand model of \( P \)?

Sufficient to restrict to Herbrand models contained in \( HB(P) \)

- Herbrand universe of \( P \), \( HU(P) \)
  (if no constant symbols in \( P \), a single constant symbol introduced)
- Herbrand base of \( P \), \( HB(P) \)
- Ground atoms not in \( HB(P) \) are not true in all Herbrand models
### Least Herbrand model

- Every Horn program $P$ has a **least** Herbrand model $LM(P)$
  - the intersection of a set of Herbrand models of a Horn program is a Herbrand model of the program
  - $HB(P)$ is an Herbrand model of $P$
  - the intersection of all models is a least Herbrand model (and it is contained in $HB(P)$)

- **Single** intended Herbrand model
  - For a *ground* $t$, $P \models p(t)$ if and only if $p(t) \in LM(P)$
  - For *ground* $t$, if $P \not\models p(t)$, **defeasibly** conclude $\neg p(t)$
  - Closed World Assumption (CWA)
Computing with Horn programs

What do they specify, what can they express?

- A Horn program $P$ specifies a subset $X$ of the Herbrand universe for $P$, $HU(P)$, if for some predicate symbol $p$ occurring in $P$ we have:

  $$X = \{ t \in HU(P) : p(t) \in LM(P) \}$$

- Finite Horn programs specify precisely the r.e. sets and relations

Reachability — an example

Program $P$

\[
\begin{align*}
&\text{arc}(a, b).
&\text{arc}(b, c).
&\text{arc}(d, c).

&\text{reach}(X, X).
&\text{reach}(X, Y) \leftarrow \text{arc}(X, Z), \text{reach}(Z, Y).
\end{align*}
\]
Reachability — an example

\( HU(P), HB(P), \text{ground}(P) \)

- \( HU(P) = \{a, b, c, d\} \)
- \( HB(P) = \{\text{arc}(a, a), \text{arc}(a, b), \ldots, \text{reach}(a, a), \ldots\} \)
- \( \text{ground}(P) : \)
  
  \[
  \begin{align*}
  \text{arc}(a, b), \quad \text{arc}(b, c), \quad \text{arc}(d, c), \\
  \text{reach}(a, a), \quad \text{reach}(b, b), \quad \text{reach}(c, c), \quad \text{reach}(d, d), \\
  \text{reach}(a, a) & \leftarrow \text{arc}(a, a), \text{reach}(a, a), \\
  \text{reach}(a, b) & \leftarrow \text{arc}(a, b), \text{reach}(b, a), \\
  \ldots & \\
  \text{reach}(c, b) & \leftarrow \text{arc}(c, d), \text{reach}(d, b), \\
  \ldots
  \end{align*}
  \]
Reachability — an example

Least model

- \( \text{arc}(a, b), \text{arc}(a, c), \text{arc}(d, c) \)
- \( \text{reach}(a, a), \text{reach}(b, b), \text{reach}(c, c), \text{reach}(d, d) \)
- \( \text{reach}(a, b), \text{reach}(a, c), \text{reach}(d, c), \text{reach}(a, c) \)

What’s computed?

- Assume \( \text{reach} \) is the distinguished “solution” predicate
- \( \{(a, a), (b, b), (c, c), (d, d), (a, b), (a, c), (d, c), (a, c)\} \)
Reachability — an example

Least model

- $arc(a, b)$, $arc(a, c)$, $arc(d, c)$
- $reach(a, a)$, $reach(b, b)$, $reach(c, c)$, $reach(d, d)$
- $reach(a, b)$, $reach(a, c)$, $reach(d, c)$, $reach(a, c)$

What’s computed?

- Assume $reach$ is the distinguished “solution” predicate
- $\{(a, a), (b, b), (c, c), (d, d), (a, b), (a, c), (d, c), (a, c)\}$
Possible issues?

- Function symbols necessary!
- List constructor \( [\cdot|\cdot] \) used to define higher-order objects
- Terms - basic data structures
- Queries (goals):
  - ?p(t) - is \( p(t) \) entailed?
  - ?p(X) - for what ground \( t \), is \( p(t) \) entailed?
- Proofs provide answers
- SLD-resolution
- Unification - basic mechanism to manipulate data structures
- Extensive use of recursion
- Leads to Prolog
Example

Manipulating lists: append and reverse

\[
\text{append}([], X, X).
\]
\[
\text{append}([X|Y], Z, [X|T]) \leftarrow \text{append}(Y, Z, T).
\]

\[
\text{reverse}([], []).
\]
\[
\text{reverse}([X|Y], Z) \leftarrow \text{append}(U, [X], Z), \text{reverse}(Y, U).
\]

- both relations defined recursively
- terms represent complex objects: lists, sets, ...
Example, cont’d

Playing with *reverse*

- Problem: reverse list \([a, b, c]\)
  - Ask query ? – *reverse*([\(a, b, c\)], \(X\)).
  - A proof of the query yields a substitution: \(X = [c, b, a]\)
  - The substitution constitutes an answer
- Query ? – *reverse*([\(a\|X\)], [\(b, c, d, a\)]) returns \(X = [d, c, b]\)
- Query ? – *reverse*([\(a\|X\)], [\(b, c, d, b\)]) returns no substitutions (there is no answer)
### Observations

- **Techniques to search for proofs — the key**
- Understanding of the resolution mechanism is important.
- It may make a difference which logically equivalent form is used:
  - $\text{reverse}([X|Y], Z) \leftarrow \text{append}(U, [X], Z), \text{reverse}(Y, U)$.
  - $\text{reverse}([X|Y], Z) \leftarrow \text{reverse}(Y, U), \text{append}(U, [X], Z)$.
- Termination vs. non-termination for query:
  - $? \leftarrow \text{reverse}([a|X], [b, c, d, b])$
- **Is it truly knowledge representation?**
  - is it truly declarative?
  - implementations are not!
- **Nonmonotonicity quite restricted**
Negation in the body

Why negation?

- Natural linguistic concept
- Facilitates knowledge representation (declarative descriptions and definitions)
- Needed for modeling convenience
- Clauses of the form:
  \[ p(\bar{X}) \leftarrow q_1(\bar{X}_1), \ldots, q_k(\bar{X}_k), \text{not } r_1(\bar{Y}_1), \ldots, \text{not } r_l(\bar{Y}_l) \]
- Things get more complex!
Observations

- Still Herbrand models
- Still restricted to $HB(P)$
- But — usually no least Herbrand model!
- Program
  
  \[
  a \leftarrow \text{not } b \\
  b \leftarrow \text{not } a
  \]

  has two \textbf{minimal} Herbrand models: $M_1 = \{a\}$ and $M_2 = \{b\}$.
- Identifying a \textbf{single} intended model a major issue
## Great Logic Programming Schism

- **Single intended model approach**
  - continue along the lines of Prolog
- **Multiple intended model approach**
  - branch into answer-set programming
Single intended model approach

“Better” Prolog

- Extensions of Horn logic programming
  - Perfect semantics of stratified programs
  - 3-val well-founded semantics for (arbitrary) programs
- Top-down computing based on unification and resolution
- XSB – David Warren at SUNY Stony Brook
- Will come back to it
Multiple intended models

Answer-set programming

- Semantics assigns to a program not one but many intended models!
  - for instance, all stable or supported models (to be introduced soon)
- How to interpret these semantics?
  - skeptical reasoning: a ground atom is cautiously entailed if it belongs to all intended models
  - intended models represent different possible states of the world, belief sets, solutions to a problem
- Nonmonotonicity shows itself in an essential way
  - as in default logic
Preliminary observations and comments

- Logic programs with negation
- Still interested only in Herbrand models
- Thus, enough to consider propositional case
- Supported model semantics
- Stable model semantics
- Connection to propositional logic (Clark’s completion, tightness, loop formulas)
- Kripke-Kleene and well-founded semantics
- Strong and uniform equivalence
Syntax

- Propositional language over a set of atoms $At$ (possibly infinite)
- Clause $r$
  
  $$a ← b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n$$

  - $a, b_i, c_j$ are atoms
  - $a$ is the head of the clause: $hd(r)$
  - literals $b_i, not\ c_j$ form the body of the rule: $bd(r)$
  - $\{b_1, \ldots, b_m\}$ - positive body $bd^+(r)$
  - $\{c_1, \ldots, c_n\}$ - negative body $bd^-(r)$
One-step provability operator

Basic tool in LP

van Emden, Kowalski 1976

- Operator on interpretations (sets of atoms)
  - \( T_P(I) = \{ \text{hd}(r) : I \models bd(r) \} \)
- If \( P \) is Horn, \( T_P \) is monotone
  - Let \( I \subseteq J \)
  - Let \( bd(r) = b_1, \ldots, b_m \) (no negation!)
  - If \( I \models bd(r) \) then \( J \models bd(r) \)
  - Thus, \( T_P(I) \subseteq T_P(J) \)
  - Least fixpoint of \( T_P \) exists and coincides with the least Herbrand model of \( P \)
- In general, not the case (due to negation)
  - \( \emptyset \models \textit{not } a \)
  - but \( \{ a \} \not\models \textit{not } a \)
**Definition and some observations**

- $M \subseteq \text{At}$ is a **supported** model of $P$ if $T_P(M) = M$
- Supported models are indeed models
  - let $M \models bd(r)$
  - then $hd(r) \in T_P(M)$
  - and so, $hd(r) \in M$
- Supported models are subsets of $\text{At}(P)$ (the Herbrand base of $P$)
- Thus, they are Herbrand models
**Supported models - example**

<table>
<thead>
<tr>
<th>Program</th>
<th>$p \leftarrow \text{not } q$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>One supported model:</td>
<td>$M_1 = {p}$</td>
</tr>
<tr>
<td>$M_2 = {q}$ - not supported (but model)</td>
<td></td>
</tr>
<tr>
<td>$p$ “follows”</td>
<td></td>
</tr>
<tr>
<td>If $q$ included in the program (more exactly: a rule $q \leftarrow$)</td>
<td></td>
</tr>
<tr>
<td>Just one supported model:</td>
<td>$M_1 = {q}$.</td>
</tr>
<tr>
<td>$p$ does not ‘follow’</td>
<td></td>
</tr>
<tr>
<td>nonmonotonicity</td>
<td></td>
</tr>
</tbody>
</table>
Two supported models: $M_1 = \emptyset$ and $M_2 = \{p\}$

The second one is self-supported (circular justification)

A problem for KR
Clark’s completion

Rules as implications

- $bd^\wedge(r)$  the conjunction of all literals in the body of $r$
  with all not $c$ replaced with $\neg c$
- $\text{cmpl}(P) = \{ bd^\wedge(r) \rightarrow hd(r) : r \in P \}$
Clark’s completion

Rules as definitions

- **Notation:** $\text{def}_P(a) = \bigvee \{ bd^\wedge(r) : \text{hd}(r) = a \}$
- **Note:** if $a$ not the head of any rule in $P$, $\text{def}_P(a) = \bot$
- $\text{cmpl} \rightarrow(P) = \{ a \rightarrow \text{def}_P(a) : a \in \text{At} \}$
- $\text{cmpl}(P) = \text{cmpl} \leftarrow(P) \cup \text{cmpl} \rightarrow(P)$
- **Note:** if $a \notin \text{At}(P)$, $\text{cmpl}(P) \models \neg a$
Clark’s completion

Example

\[ a \leftarrow b, \text{not } c \]
\[ a \leftarrow d \]
\[ b \leftarrow a \]

- \( \text{def}(a) = (b \land \neg c) \lor d \)
- \( \text{def}(b) = a \)
- \( \text{def}(c) = \bot \)
- \( \text{cmpl} \leftarrow = \{ b \land \neg c \rightarrow a; d \rightarrow a; a \rightarrow b \} = \{(b \land \neg c) \lor d \rightarrow a; a \rightarrow b \} \)
- \( \text{cmpl} \leftarrow = \{ \text{def}(a) \rightarrow a; \text{def}(b) \rightarrow b; \text{def}(c) \rightarrow c \} \)
- \( \text{cmpl} \rightarrow = \{ a \rightarrow \text{def}(a); b \rightarrow \text{def}(b); c \rightarrow \text{def}(c) \} \)
- \( \text{cmpl} = \{ a \leftrightarrow \text{def}(a); b \leftrightarrow \text{def}(b); c \leftrightarrow \text{def}(c) \} \)
- \( \text{cmpl} \) has two models: \( \emptyset \) and \( \{a, b\} \)
Clark’s completion

A set $M \subseteq At$ is a supported model of a program $P$ if and only if $M$ is a model (in a standard sense) of $cmpl(P)$

- Connection to SAT
- Allows us to use SAT solvers to compute supported models of $P$
Connection to supported models — proof

\[ M \overset{\text{— supported model of } P}{=} T_P(M) \]

- Let \( a \in M \Rightarrow \exists r \in P \text{ st: } hd(r) = a \text{ and } M \models bd(r) \)
- \( \Rightarrow M \models bd^\wedge(r), \quad M \models \text{def}(a) \text{ and } M \models a \leftrightarrow \text{def}(a) \)
- Let \( a \notin M \Rightarrow \forall r \in P \text{ st: } hd(r) = a, \quad M \not\models bd(r) \)
- \( \Rightarrow M \not\models \text{def}(a) \text{ and } M \models a \leftrightarrow \text{def}(a) \)

Conversely: let \( M \models \text{cmpl}(P) \)

- \( \Rightarrow M \models P \text{ and so, } T_P(M) \subseteq M \)
- Let \( a \in M \Rightarrow M \models \text{def}(a) \)
- \( \Rightarrow \exists r \in P \text{ st: } M \models bd^\wedge(r) \)
- \( \Rightarrow M \models bd(r) \text{ and } a \in T_P(M) \Rightarrow M \subseteq T_P(M) \)
- Thus, \( M = T_P(M) \text{ and } M \text{ supported} \)
### Connection to supported models — proof

**Proof**

Let $a \in M$ \implies \exists r \in P \text{ st: } hd(r) = a \text{ and } M \models bd(r)$

\implies M \models bd(r), \text{ and } M \models def(a)$

Let $a \notin M \implies \forall r \in P \text{ st: } hd(r) = a, \text{ and } M \not\models bd(r)$

\implies M \not\models def(a)$

Conversely: let $M \models cmpl(P)$

\implies M \models P \text{ and so, } T_P(M) \subseteq M$

Let $a \in M \implies M \models def(a)$

$\exists r \in P \text{ st: } M \models bd(r)$

\implies M \models bd(r)$

Thus, $M = T_P(M)$ and $M$ supported
Stable model semantics

<table>
<thead>
<tr>
<th>Supported models of interest, but ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Some supported models based on circular arguments</td>
</tr>
<tr>
<td>▶ $p \leftarrow p$</td>
</tr>
<tr>
<td>▶ ${p}$ is supported model (circular argument)</td>
</tr>
<tr>
<td>▶ Some more stringent bases for selecting intended models needed</td>
</tr>
</tbody>
</table>
Stable model semantics

Gelfond-Lifschitz reduct

- $P$ — logic program
- $M$ — set of atoms
- \textbf{Reduct} $P^M$
  - for each $a \in M$ remove rules with $\text{not } a$ in the body
  - remove literals $\text{not } a$ from all other rules
Stable model semantics

Definition through reduct

- Reduct $P^M$ is a Horn program
- It has the least model $LM(P^M)$
- $M$ is a stable model of $P$ if

$$M = LM(P^M)$$
Stable model semantics

And now through Gelfond-Lifschitz operator

- $GL_P(M) = LM(P^M)$
- $M$ is a stable model if and only if $M = GL_P(M)$
- $GL_P$ is antimonotone
- For $M \subseteq N$: $GL_P(N) \subseteq GL_P(M)$
Multiple stable models

\[
\begin{align*}
    p & \leftarrow q, \text{not } s \\
    r & \leftarrow p, \text{not } q, \text{not } s \\
    s & \leftarrow \text{not } q \\
    q & \leftarrow \text{not } s
\end{align*}
\]

- Two stable models: \( M_1 = \{p, q\} \) and \( M_2 = \{s\} \)

No stable models

\[
\begin{align*}
    p & \leftarrow \text{not } p
\end{align*}
\]

- No stable models!!
Stable models — examples

Multiple stable models

\[ p \leftarrow q, \text{not } s \]
\[ r \leftarrow p, \text{not } q, \text{not } s \]
\[ s \leftarrow \text{not } q \]
\[ q \leftarrow \text{not } s \]

- Two stable models: \( M_1 = \{p, q\} \) and \( M_2 = \{s\} \)

No stable models

\[ p \leftarrow \text{not } p \]

- No stable models!!
Stable models — properties

Stable models are models!

- Let $M$ be a stable model
- $M$ is a model of all rules that are removed from the program when forming the reduct
- $M$ is a model of every rule that contributes to the reduct
- Indeed, each such rule is subsumed by a rule in the reduct and $M$ satisfies all rules in the reduct
Stable models — properties

Stable models are minimal models!

- Let $N$ be a stable model and $M$ a model s.t. $M \subseteq N$
- Then
  $$N = GL_p(N) \subseteq GL_p(M) \subseteq M$$
- Thus, $N \subseteq M$ and so $N = M$
- The minimality of $N$ follows
- Stable models form an antichain!
Lemma: If $M$ model of $P$, $GL_P(M) \subseteq M$

- Since $M$ model of $P$, then $M$ is a model of $P^M$
- $a \leftarrow b_1, \ldots, b_m$ - a rule in reduct
- $a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$ - the original rule in $P$
- $M$ satisfies the latter, and it satisfies every $\text{not } c_i$ (as $c_i \not\in M$)
- Thus, $M$ satisfies the reduct rule
Stable models — properties

Connection to supported models

- If $M$ is a stable model of $P$ then it is a supported model of $P$
- Let $M$ be a stable model of $P$
- Then $M$ model of $P$ and so, $T_P(M) \subseteq M$
- $r = a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$ - a rule in $P$ such that $M \models bd(r)$
- Then $r' = a \leftarrow b_1, \ldots, b_m$ belongs to the reduct $P^M$
- And $M \models bd(r')$
- Since $M$ is a model of $P^M$, $a \in M$
- Hence, $T_P(M) \subseteq M$ and so, $M = T_P(M)$
- That is, $M$ is supported!!
But ...

- The converse not true, in general (as it should not be)

Counterexample

- $p \leftarrow p$
- $\{p\}$ is supported but not stable
- Positive dependency of $p$ on itself is a problem
But ...

- The converse not true, in general (as it should not be)

Counterexample

- $p \leftarrow p$
- $\{p\}$ is supported but not stable
- Positive dependency of $p$ on itself is a problem
Fages Lemma

Positive dependency graph $G^+(P)$

- Atoms of $P$ are vertices
- $(a, b)$ is an edge in $G^+(P)$ if for some $r \in P$: $hd(r) = a$, $b \in bd^+(r)$

Tight programs

- $P$ is tight if $G^+(P)$ is acyclic
- Alternatively, if there is a labeling of atoms with non-negative integers $\lambda(a)$ s.t.
- for every rule $r \in P$

$$\lambda(hd(r)) > \max\{\lambda(b) : b \in bd^+(r)\}$$

- Connection to topological ordering of positive dependency graphs
**Fages Lemma**

**Positive dependency graph** $G^+(P)$

- Atoms of $P$ are vertices
- $(a, b)$ is an edge in $G^+(P)$ if for some $r \in P$: $hd(r) = a$, $b \in bd^+(r)$

**Tight programs**

- $P$ is tight if $G^+(P)$ is **acyclic**
- Alternatively, if there is a labeling of atoms with non-negative integers ($a \mapsto \lambda(a)$) s.t.
  - for every rule $r \in P$
    \[
    \lambda(hd(r)) > \max\{\lambda(b) : b \in bd^+(r)\}
    \]
- Connection to topological ordering of positive dependency graphs
The statement — finally

- If $P$ is tight then every supported model is stable
- Intuitively, circular support not possible
Fages Lemma — example

Program $P$

\[
\begin{align*}
p & \leftarrow q, \text{not } s \\
r & \leftarrow p, \text{not } q, \text{not } s \\
s & \leftarrow \text{not } q \\
q & \leftarrow \text{not } s
\end{align*}
\]

Graph $G^+(P)$

$P$ is tight

- \{p, q\} and \{s\} are supported models of $P$
  - $T_P(\{p, q\}) = \{p, q\}$
  - $T_P(\{s\}) = \{s\}$
- Thus, they are stable (which we verified directly earlier)
Fages Lemma — example

Program $P$

\[
\begin{align*}
  p & \leftarrow q, \text{not } s \\
  r & \leftarrow p, \text{not } q, \text{not } s \\
  s & \leftarrow \text{not } q \\
  q & \leftarrow \text{not } s
\end{align*}
\]

Graph $G^+(P)$

$P$ is tight

- \{p, q\} and \{s\} are supported models of $P$
  - $T_P(\{p, q\}) = \{p, q\}$
  - $T_P(\{s\}) = \{s\}$

- Thus, they are stable (which we verified directly earlier)
Fages Lemma — example

Program $P$

$p \leftarrow q, \neg s$
$r \leftarrow p, \neg q, \neg s$
$s \leftarrow \neg q$
$q \leftarrow \neg s$

Graph $G^+(P)$

$p$ $q$

$P$ is tight

- $\{p, q\}$ and $\{s\}$ are supported models of $P$
  - $T_P(\{p, q\}) = \{p, q\}$
  - $T_P(\{s\}) = \{s\}$
- Thus, they are stable (which we verified directly earlier)
Proof

- Let $P$ be tight and $M$ be its supported model.
- Then $M$ is a model of $P^M$.
- Let $N$ be a model of $P^M$.
- Claim: for every $k$, if $a \in M$ and $\lambda(a) < k$, then $a \in N$.
- Holds for $k = 0$ (trivially).
- Let $a \in M$ and $\lambda(a) = k$.
- Since $M$ supported, there is $r \in P$ such that $a = hd(r)$ and $M \models bd(r)$.
- In particular, $bd^+(r) \subseteq M$ and so, $bd^+(r) \subseteq N$ (by I.H.).
- Since $M \models bd(r)$, $M$ contributes to the reduct.
- Since $N$ is a model of $P^M$, $a \in N$.
- It follows that $M = LM(P^M)$.
Relativized tightness

- Let $X \subseteq \text{At}(P)$
- $P$ is tight on $X$ if the program consisting of rules $r$ such that $\text{bd}^+(r) \subseteq X$ is tight

Generalization

- If $P$ is tight on $X$ and $M$ is a supported model of $P$ such that $M \subseteq X$, then $M$ is stable
A generalization

Erdem and Lifschitz, 2000

Relativized tightness

- Let $X \subseteq At(P)$
- $P$ is tight on $X$ if the program consisting of rules $r$ such that $bd^+(r) \subseteq X$ is tight

Generalization

- If $P$ is tight on $X$ and $M$ is a supported model of $P$ such that $M \subseteq X$, then $M$ is stable
Generalized Fages Lemma — example

Program $P$

\[
\begin{align*}
p & \leftarrow q, \text{not } s \\
r & \leftarrow p, \text{not } q, \text{not } s \\
s & \leftarrow \text{not } q \\
q & \leftarrow \text{not } s \\
p & \leftarrow r
\end{align*}
\]

Graph $G^+(P)$

$P$ is not tight

- $\{p, q\}$ and $\{s\}$ are still supported models of $P$
  - $T_P(\{p, q\}) = \{p, q\}$
  - $T_P(\{s\}) = \{s\}$
- Since $P$ is tight on each of them, they are stable
Generalized Fages Lemma — example

Program $P$

\[
\begin{align*}
p & \leftarrow q, \neg s \\
r & \leftarrow p, \neg q, \neg s \\
s & \leftarrow \neg q \\
q & \leftarrow \neg s \\
p & \leftarrow r
\end{align*}
\]

Graph $G^+(P)$

$P$ is not tight

- $\{p, q\}$ and $\{s\}$ are still supported models of $P$
  - $T_P(\{p, q\}) = \{p, q\}$
  - $T_P(\{s\}) = \{s\}$

- Since $P$ is tight on each of them, they are stable
Generalized Fages Lemma — example

Program $P$

\[
\begin{align*}
p & \leftarrow q, \text{not } s \\
r & \leftarrow p, \text{not } q, \text{not } s \\
s & \leftarrow \text{not } q \\
q & \leftarrow \text{not } s \\
p & \leftarrow r
\end{align*}
\]

Graph $G^+(P)$

$P$ is not tight

- \{p, q\} and \{s\} are still supported models of $P$
  - $T_P(\{p, q\}) = \{p, q\}$
  - $T_P(\{s\}) = \{s\}$
- Since $P$ is tight on each of them, they are stable
Loops and loop formulas

Lin and Zhao, 2002

External support formula for $Y \subseteq At(P)$

- For a rule $r$:
  - $bd^\wedge(r)$ the conjunction of all literals in the body of $r$
    
    *with all not $c$ replaced with $\neg c$*
  - $ES_P(Y)$ the disjunction of $bd^\wedge(r)$ for all $r \in P$ st:
    - $hd(r) \in Y$
    - $bd^+(r) \cap Y = \emptyset$
- For finite programs: well-formed formulas
- Hence, will assume finiteness

Observations

- $ES_P(\emptyset) = \top$
- $ES_P(\{a\}) = def_P(a)$
  
  *cf. Clark’s completion*
External support formula for $Y \subseteq At(P)$

- For a rule $r$:
  - $bd^\wedge(r)$ the conjunction of all literals in the body of $r$ with all not $c$ replaced with $\neg c$
  - $ES_P(Y)$ the disjunction of $bd^\wedge(r)$ for all $r \in P$ st:
    - $hd(r) \in Y$
    - $bd^+(r) \cap Y = \emptyset$

- For finite programs: well-formed formulas
- Hence, will assume finiteness

Observations

- $ES_P(\emptyset) = \top$
- $ES_P(\{a\}) = \text{def}_P(a)$
  - cf. Clark’s completion
A characterization of stable models

for finite programs, the following conditions are equivalent

- $X$ is a stable model of $P$
- $X$ is a model of $\text{cmpl}(P) \cup \{ Y^\land \rightarrow ES_P(Y) : Y \subseteq \text{At}(P) \}$
- $X$ is a model of $\text{cmpl}(P) \cup \{ Y^\lor \rightarrow ES_P(Y) : Y \subseteq \text{At}(P) \}$
- OK to replace $\text{cmpl}(P)$ with $\text{cmpl}(P)$
  - $\text{cmpl}(P) \subseteq \{ Y^\land \rightarrow ES_P(Y) : Y \subseteq \text{At}(P) \}$
  - $\text{cmpl}(P) \subseteq \{ Y^\lor \rightarrow ES_P(Y) : Y \subseteq \text{At}(P) \}$
Loops

Definition

- A loop is a set $Y \subseteq At(P)$ that induces in $G^+(P)$ a strongly connected subgraph.
- In particular, all singleton sets are loops.
**Program $P$**

- $p \leftarrow q, \text{not } r$
- $q \leftarrow p$
- $r \leftarrow \text{not } p$

**Graph $G^+(P)$**

- $\{p\}$, $\{q\}$, $\{r\}$, $\{p, q\}$ are loops
- $\{p, q, r\}$ is not!
Loops — example

Program $P$

\[
\begin{align*}
p & \leftarrow q, \text{not } r \\
q & \leftarrow p \\
r & \leftarrow \text{not } p
\end{align*}
\]

Graph $G^+(P)$

- $\{p\}$, $\{q\}$, $\{r\}$, $\{p, q\}$ are loops
- $\{p, q, r\}$ is not!
Loop Theorem

For finite programs, the following conditions are equivalent:

- $X$ is a stable model of $P$
- $X$ is a model of $\text{cmpl}^{-}(P) \cup \{ Y^\lor \rightarrow ES_P(Y): Y \text{ – a loop} \}$
- $X$ is a model of $\text{cmpl}^{-}(P) \cup \{ Y^\land \rightarrow ES_P(Y): Y \text{ – a loop} \}$

- OK to replace $\text{cmpl}^{-}(P)$ with $\text{cmpl}(P)$
  - Singleton sets are loops!
Loop Theorem

Let $X$ be a stable model of $P$

- $\Rightarrow X \models P \Rightarrow X \models P^X$
- $X \models P \Rightarrow X \models \text{cmpl} \leftarrow (P)$
- Let $Y$ be a loop st: $X \models Y^\lor \Rightarrow X \cap Y \neq \emptyset$
- Let $a \in Y$ be the “first” element of $Y$ in $X$
  recall that $X = LM(P^X)$
- $\Rightarrow \exists r \in P \text{ st: } a = \text{hd}(r), \ bd^+(r) \cap Y = \emptyset$
- $\Rightarrow bd^\lor(r)$ is a disjunct of $ES_P(Y)$
- Also: $bd^+(r) \subseteq X$ and $bd^-(r) \cap X = \emptyset \Rightarrow X \models bd^\lor(r)$
- $\Rightarrow X \models ES_P(Y) \Rightarrow X \models Y^\lor \rightarrow ES_P(Y)$
- No difference if $Y^\lor$ replaced with $Y^\land$
Loop Theorem

Let $X \models \text{cmp} \leftarrow (P) \cup \{ Y^\uparrow \rightarrow ES_P(Y) : Y \text{ a loop} \}$

- $X \models P \Rightarrow X \models P^X$
- Let $X' = LM(P^X) \Rightarrow X' \subseteq X$
- Let $X \setminus X' \neq \emptyset$
- Consider subgraph $H$ of $G(P)$ induced by $X \setminus X'$
- Let $Y$ be a “terminal” strongly connected component of $H$
  no edge in $H$ starts in $Y$ and ends outside of $Y$
Let $X \models \text{cmpl}^\leftarrow(P) \cup \{Y^\uparrow \rightarrow ES_P(Y) : Y \text{ -- a loop}\}$

- $\Rightarrow X \models P \Rightarrow X \models P^X$
- Let $X' = LM(P^X) \Rightarrow X' \subseteq X$
- Let $X \setminus X' \neq \emptyset$
- Consider subgraph $H$ of $G(P)$ induced by $X \setminus X'$
- Let $Y$ be a “terminal” strongly connected component of $H$
  - no edge in $H$ starts in $Y$ and ends outside of $Y$
Loop Theorem

Proof, cont’d

- \( X \models Y^\wedge \rightarrow ES_P(Y) \) (also: \( X \models Y^\vee \rightarrow ES_P(Y) \))
- Since \( Y \subseteq X \): \( \Rightarrow X \models ES_P(Y) \)
- \( \Rightarrow \exists r \in P \text{ st: } hd(r) \in Y, \ bd^+(r) \cap Y = \emptyset, \ X \models bd^+(r) \)
- \( \Rightarrow bd^+(r) \subseteq X \) and \( r^X \in P^X \)
- Since \( Y \) terminal and \( bd^+(r) \cap Y = \emptyset \): \( \Rightarrow bd^+(r) \subseteq X' \)
  - if \( a \in bd^+(r) \): \( a \in X \)
  - \((hd(r), a)\) edge in \( G^+(P) \)
  - if \( a \in X \setminus X' \): \((hd(r), a)\) edge in \( H \)
  - \( \Rightarrow a \in Y \), contradiction
  - \( \Rightarrow a \in X' \)
- Since \( X' \models P^X \): \( \Rightarrow X' \models r^X \)
- \( \Rightarrow hd(r) \in X' \)
- Since \( X' \cap Y = \emptyset \): \( \Rightarrow \) contradiction
Some programs have no stable nor supported models

- Sufficient conditions for the existence of stable models
- 4-val supported and stable models
Sufficient conditions

General dependency graph $G(P)$

- Atoms of $P$ are vertices
- $(a, b)$ is an edge in $P$ if for some $r \in P$: $\text{hd}(r) = a$, $b \in \text{bd}(r)$
- If $b \in \text{bd}^+(r)$ — edge is positive
- If $b \in \text{bd}^-(r)$ — edge is negative

A propositional program $P$ is

- Call-consistent: if $G(P)$ has no odd cycles (cycles with an odd number of negative edges)
- Stratified: if $G(P)$ has no paths with infinitely many negative edges
  - in particular, no cycles with a negative edge (for finite programs both conditions are equivalent)
Sufficient conditions

General dependency graph \( G(P) \)

- Atoms of \( P \) are vertices
- \((a, b)\) is an edge in \( P \) if for some \( r \in P: \) \( \text{hd}(r) = a, b \in \text{bd}(r) \)
- If \( b \in \text{bd}^+(r) \) — edge is **positive**
- If \( b \in \text{bd}^-(r) \) — edge is **negative**

A propositional program \( P \) is

- **Call-consistent:** if \( G(P) \) has no odd cycles (cycles with an odd number of negative edges)
- **Stratified:** if \( G(P) \) has no paths with infinitely many negative edges
  - in particular, no cycles with a negative edge (for finite programs both conditions are equivalent)
Sufficient conditions

Results

- If a finite logic program is call-consistent, it has a stable model
- If a program is stratified it has a unique stable model
Splitting

- Let $P$ and $Q$ be programs such that $P$ does not contain any of the head atoms of $Q$
  
  “$Q$ above $P$”

- A set $M$ is a stable model of $P \cup Q$ iff there is a stable model $N$ of $P$ such that $M$ is a stable model of $Q \cup N$
Splitting Theorem

Proof: ($\Rightarrow$) Let $M \in \text{StM}(P \cup Q)$

- $N := M \cap \text{At}(P)$
- $P^N = P^M$ (as $(M \setminus N) \cap \text{At}(P) = \emptyset$)
- $M \models P \Rightarrow M \models P^M \Rightarrow M \models P^N$
- $\Rightarrow N \models P^N$ (as $(M \setminus N) \cap \text{At}(P) = \emptyset$)
- Let $N' \subseteq N$ be st: $N' \models P^N$
- $\Rightarrow N' \models P^M \Rightarrow N' \cup (M \setminus N) \models P^M$
- Let $r \in Q^M$ be st: $N' \cup (M \setminus N) \models \text{bd}(r)$
- $\Rightarrow M \models \text{bd}(r) \Rightarrow M \models \text{hd}(r)$ (as $M \models Q$ and so, $M \models Q^M$)
- $\Rightarrow \text{hd}(r) \in M \Rightarrow \text{hd}(r) \in M \setminus N \Rightarrow \text{hd}(r) \in N' \cup (M \setminus N)$
- $\Rightarrow N' \cup (M \setminus N) \models Q^M \Rightarrow N' \cup (M \setminus N) \models (P \cup Q)^M$
- $\Rightarrow N' \cup (M \setminus N) = M \Rightarrow N' = N \Rightarrow N = \text{LM}(P^N)$
- $\Rightarrow N \in \text{StM}(P)$
Next, we show that $M \in StM(Q \cup N)$

- Recall: $N = M \cap At(P)$  $\Rightarrow$  $N \subseteq M$
- Also: $M \models Q$  $\Rightarrow$  $M \models Q^M \cup N = (Q \cup N)^M$
- Let $M' \subseteq M$ be st: $M' \models (Q \cup N)^M$
- $\Rightarrow$  $N \subseteq M'$  $M' \models Q^M$
- Recall: $N \models P^N$ and so $N \models P^M$ (as $P^M = P^N$)
- $\Rightarrow$  $M' \models P^M$  $\Rightarrow$  $M' \models (P \cup Q)^M$
- Recall: $M = LM((P \cup Q)^M)$  $\Rightarrow$  $M = M'$
- $\Rightarrow$  $M = LM((P \cup N)^M)$  $\Rightarrow$  $M \in StM(Q \cup N)$
Splitting Theorem

Conversely: $M \in StM(Q \cup N)$ and $N \in StM(P)$

- $\Rightarrow M \models Q$, $N \subseteq M$, $M \subseteq \text{hd}(Q) \cup N$
- $\Rightarrow M \cap \text{At}(P) = N \Rightarrow M \models P$
- $\Rightarrow M \models P \cup Q \Rightarrow M \models (P \cup Q)^M$
- Let $M' \subseteq M$ be st: $M' \models (P \cup Q)^M$
- $N' := M' \cap \text{At}(P)$
- $\Rightarrow M' \models P^M \Rightarrow N' \models P^M \Rightarrow N' \models P^N$
- $\Rightarrow N' = N \Rightarrow N' \subseteq M' \Rightarrow M' \models Q^M \cup N = (Q \cup N)^M$
- $\Rightarrow M' = M \Rightarrow M = LM((Q \cup N)^M \Rightarrow M \in StM(P \cup Q)$
Stratification

Equivalent definition in the finite case

- **P stratified** if
  - \( \text{hd}(P) \cap \text{bd}^-(P) = \emptyset \), or
  - \( P = P_1 \cup P_2 \), where \( P_2 \) stratified, \( \text{hd}(P_1) \cap (\text{bd}^-(P_1) \cup \text{At}(P_2)) = \emptyset \)

Finite stratified programs have a unique stable model

- Induction
- Basis: exident
- Inductive step: \( P_2 \) has a unique stable model, say \( N \)
- Clearly, \( P_1 \cup N \) has a unique stable model, too
- Apply splitting theorem
What do I mean?

- Logic allows us to manipulate theories
- Tautologies can be added or removed without changing the meaning
- Consequences of formulas in theories can be added or removed without changing the meaning
  - \( \{p, p \rightarrow q\} \) the same as \( \{p, p \rightarrow q, q\} \)
  - one can always be replaced with another (within any larger context)
- Equivalence for replacement — logical equivalence necessary and sufficient
- Is there a logic which captures such manipulation with theories in nonmonotonic systems?
Query optimization

- Compute answers to a query $Q$ (program) from a knowledge base $KB$ (another program)
  
  \[
  \text{reason from } Q \cup KB
  \]

- Rewrite $Q$ into an equivalent query $Q'$, which can be processed more efficiently

  \[
  \text{reasoning from } Q' \cup KB \text{ easier}
  \]

- When are two queries equivalent?
  
  - If $Q \cup KB$ and $Q' \cup KB$ have the same meaning

    \[
    \text{not quite what we want — knowledge-base dependent}
    \]

  - If $Q \cup KB$ and $Q' \cup KB$ have the same meaning for every knowledge base $KB$

    \[
    \text{better — knowledge-base independent}
    \]
Towards modular logic programming

Equivalence of programs

$P$ and $Q$ are equivalent if they have the same models

Nonmonotonic equivalence of programs

$P$ and $Q$ are stable-equivalent if they have the same stable models
Towards modular logic programming

Equivalence of programs

- \( P \) and \( Q \) are equivalent if they have the same models

Nonmonotonic equivalence of programs

- \( P \) and \( Q \) are stable-equivalent if they have the same stable models
Towards modular logic programming

Equivalence for replacement

- **Equivalence for replacement** — for every program $R$, programs $P \cup R$ and $Q \cup R$ have the same stable models

- Commonly known as **strong equivalence**
  
  Lifschitz, Pearce, Valverde 2001; Lin 2002; Turner 2003; Eiter, Fink 2003; Eiter, Fink, Tompits, Woltran, 2005; T 2006; Woltran 2008

- Different than equivalence
  
  - $\{p \leftarrow not q\}$ and $\{q \leftarrow not p\}$
  - The same models but different meaning

- Different than stable-equivalence
  
  - $P = \{p\}$ and $Q = \{p \leftarrow not q\}$
  - The same stable models; $\{p\}$ is the only stable model in each case
  - But, $P \cup \{q\}$ and $Q \cup \{q\}$ have different stable models! ($\{p, q\}$ and $\{q\}$, respectively)
When are two programs strongly equivalent?

Se-model characterization

- A pair \((X, Y)\) of sets of atoms is an se-model of a program \(P\) if
  - \(X \subseteq Y\)
  - \(Y \models P\)
  - \(X \models P^Y\)
- \(SE(P)\) set of se-models of \(P\)
- Logic programs \(P\) and \(Q\) are strongly equivalent iff they have the same se-models \((SE(P) = SE(Q))\)
  - A similar concept characterizes strong equivalence of default theories
    - *Turner 2003*
When are two programs strongly equivalent?

**Lemma 1:** \( SE(P) = SE(Q) \implies StM(P) = StM(Q) \)

- \( Y \in StM(P) \implies Y \models P \) and \( Y \models P^Y \)
- \( \implies (Y, Y) \in SE(P) \implies (Y, Y) \in SE(Q) \)
- \( \implies Y \models Q^Y \)
- If \( Z \subseteq Y \) and \( Z \models Q^Y \) \( \implies (Z, Y) \in SE(Q) \)
- \( \implies (Z, Y) \in SE(P) \)
- \( \implies Z \models P^Y \implies Z = Y \) (as \( Y = LM(P^Y) \))
- \( \implies Y = LM(Q^Y) \implies Y \in StM(Q) \)
When are two programs strongly equivalent?

Lemma 2: \( SE(P \cup R) = SE(P) \cap SE(R) \)

- \((X, Y) \in SE(P \cup R) \text{ iff}\)
- \(X \subseteq Y \text{ and } Y \models P \cup R \text{ and } X \models (P \cup R)^Y = P^Y \cup R^Y \text{ iff}\)
- \(X \subseteq Y \text{ and } (Y \models P \text{ and } Y \models R) \text{ and } (X \models P^Y \text{ and } X \models R^Y) \text{ iff}\)
- \((X \subseteq Y, Y \models P, X \models P^Y), \text{ and}\)
  \((X \subseteq Y, Y \models R, X \models R^Y) \text{ iff}\)
- \((X, Y) \in SE(P) \text{ and } (X, Y) \in SE(R) \text{ iff}\)
- \((X, Y) \in SE(P) \cap SE(R) \)
When are two programs strongly equivalent?

**SE(P) = SE(Q) ⇒ P and Q are strongly equivalent**

- By Lemma 2, for every R:
  \[ SE(P \cup R) = SE(P) \cap SE(R) = SE(Q) \cap SE(R) = SE(PQ \cup R) \]
- By Lemma 1, \( StM(P \cup R) = StM(Q \cup R) \)

**P and Q are strongly equivalent ⇒ SE(P) = SE(Q)**

- Let \((X, Y) \in SE(P) \setminus SE(Q)\):
  \((X, Y) \in SE(P)\) and \((X, Y) \notin SE(Q)\)
  \[ \Rightarrow Y \models P^Y \Rightarrow Y = LM(P^Y \cup Y) \]
- Since \( P^Y \cup Y = (P \cup Y)^Y \), \( Y = LM((P \cup Y)^Y) \) \( \Rightarrow Y \in StM(P \cup Y) \)
  \[ \Rightarrow Y \in StM(Q \cup Y) \Rightarrow Y \models Q \]
- \( \Rightarrow X \not\models Q^Y \)
When are two programs strongly equivalent?

\[ SE(P) = SE(Q) \implies P \text{ and } Q \text{ are strongly equivalent} \]

- By Lemma 2, for every \( R \):
  \[ SE(P \cup R) = SE(P) \cap SE(R) = SE(Q) \cap SE(R) = SE(PQ \cup R) \]

- By Lemma 1, \( StM(P \cup R) = StM(Q \cup R) \)

\[ P \text{ and } Q \text{ are strongly equivalent} \implies SE(P) = SE(Q) \]

- Let \((X, Y) \in SE(P) \setminus SE(Q)\): \((X, Y) \in SE(P)\) and \((X, Y) \notin SE(Q)\)
  \[ \implies Y \models P^Y \implies Y = LM(P^Y \cup Y) \]

- Since \( P^Y \cup Y = (P \cup Y)^Y \), \( Y = LM((P \cup Y)^Y) \implies Y \in StM(P \cup Y) \)
  \[ \implies Y \in StM(Q \cup Y) \implies Y \models Q \]

\[ \implies X \not\models Q^Y \]
When are two programs strongly equivalent?

\[(X, Y) \in SE(P), (X, Y) \not\in SE(Q), Y \models Q, X \not\models Q^Y\]

- Define \(R = X \cup \{y \leftarrow y' \mid y, y' \in Y \setminus X\}\)
- \(\Rightarrow Y \models Q \cup R\) and \(Y \models (Q \cup R)^Y\)
- Let \(Z \subseteq Y\) st: \(Z \models (Q \cup R)^Y\) \(\Rightarrow Z \models Q^Y \cup R\)
- \(\Rightarrow Z \models Q^Y\) \(\Rightarrow X \neq Z\)
- Since \(Z \models R\), \(X \subseteq Z\) \(\Rightarrow \exists y \in Y \setminus X\) st: \(y \in Z\)
- Since \(Z \models R\), \(Y \setminus X \subseteq Z\)
- \(\Rightarrow Y \subseteq Z\) \(\Rightarrow Z = Y\)
- \(\Rightarrow Y \in StM(Q \cup R)\) \(\Rightarrow Y \in StM(P \cup R)\)
- \(\Rightarrow Y = LM(P \cup R)^Y\)
- Since \(X \models P^Y \cup R = (P \cup R)^Y\), \(X = Y\)
- \(\Rightarrow Y \not\models Q^Y\) \(\Rightarrow Y \not\models Q\), a contradiction
An interesting variant

Uniform equivalence

- Programs $P$ and $Q$ are uniformly equivalent if for every set $D$ of facts (rules with empty body) $P \cup D$ and $Q \cup D$ have the same stable models
- Relevant for DB query optimization
- Different than other types of equivalence discussed here
When are two programs uniformly equivalent?

Se-model characterization

- Programs $P$ and $Q$ are uniformly equivalent iff
  - for every $Y \subseteq At$, $Y$ is a model of $P$ if and only if $Y$ is a model of $Q$
  - for every $(X, Y) \in SE(P)$ such that $X \subseteq Y$, there is $U \subseteq At$ such that $X \subseteq U \subseteq Y$ and $(U, Y) \in SE(Q)$
  - for every $(X, Y) \in SE(Q)$ such that $X \subseteq Y$, there is $U \subseteq At$ such that $X \subseteq U \subseteq Y$ and $(U, Y) \in SE(P)$
When are two programs uniformly equivalent?

Ue-model characterization

- A pair \((X, Y)\) of sets of atoms is a \textit{ue-model} of a program \(P\) if it is an se-model of \(P\) and
- For every se-model \((X', Y)\) such that \(X \subseteq X'\), \(X' = X\) or \(X' = Y\)
- \textbf{Finite} logic programs \(P\) and \(Q\) are uniformly equivalent \textit{iff} they have the same ue-models

\textit{Eiter and Fink, 2003}
## Formulas

- **Base**: atoms and the symbol $\bot$ ("false")
- **Connectives** $\land$, $\lor$ and $\rightarrow$
- **Shortcuts**
  - $\neg F ::= F \rightarrow \bot$
  - $\top ::= \bot \rightarrow \bot$
  - $F \leftrightarrow G ::= (F \rightarrow G) \land (G \rightarrow F)$
General logic programs

Positive and negative occurrences of atoms in formulas

- An occurrence of $a$ in $F$ is **positive**, if the # of implications with this occurrence of $a$ in antecedent is even
- Otherwise, it is **negative**
- An occurrence of $a$ in $F$ is **strictly positive** if no implication contains this occurrence of $a$ in the antecedent
  - $\neg F$ (that is, $F \rightarrow \bot$) has no strict occurrences of any atom.
- A **head** atom (of a formula) an atom with at least one strictly positive occurrence
- In $(\neg p \rightarrow q) \rightarrow (p \lor \neg q)$:
  - the first occurrence of $p$ is negative
  - the second occurrence of $p$ is strictly positive
  - both occurrences of $q$ are negative
Stable-model semantics

Reduct of a formula $F$ with respect to a set $X$ of atoms

- The formula $F^X$ obtained by replacing in $F$ each maximal subformula of $F$ that is not satisfied by $X$ with $\bot$

Example: $F = (\neg p \rightarrow q) \land (\neg q \rightarrow p)$ and $X = \{p\}$

- $\neg p = p \rightarrow \bot$, and $X \models \neg p \rightarrow q$
- Thus: $\neg p$ is a maximal subformula not satisfied by $X$
- $\neg q = q \rightarrow \bot$, $X \not\models q$, $X \models \neg q$
- Thus, $q$ is a maximal subformula not satisfied by $X$
- Thus: $F^X = (\bot \rightarrow q) \land ((\bot \rightarrow \bot) \rightarrow p)$
- Classically equivalent to $p$
Stable-model semantics

Reduct of a formula $F$ with respect to a set $X$ of atoms

- The formula $F^X$ obtained by replacing in $F$ each maximal subformula of $F$ that is not satisfied by $X$ with $\bot$

Example: $F = (\neg p \rightarrow q) \land (\neg q \rightarrow p)$ and $X = \{p\}$

- $\neg p = p \rightarrow \bot$, and $X \models \neg p \rightarrow q$
- Thus: $\neg p$ is a maximal subformula not satisfied by $X$
- $\neg q = q \rightarrow \bot$, $X \not\models q$, $X \models \neg q$
- Thus, $q$ is a maximal subformula not satisfied by $X$
- Thus: $F^X = (\bot \rightarrow q) \land ((\bot \rightarrow \bot) \rightarrow p)$
- Classically equivalent to $p$
Stable-model semantics

To facilitate computation of the reduct

- $\bot^X = \bot$
- For $a$ an atom, if $a \in X$, $a^X = a$; otherwise, $a^X = \bot$
- If $X \models F \circ G$, $(F \circ G)^X = F^X \circ G^X$; otherwise, $(F \circ G)^X = \bot$ ($\circ$ stands for any of $\land$, $\lor$, $\to$)
- If $X \models F$, $(\neg F)^X = \bot$; otherwise, $(\neg F)^X = (F \to \bot) = (\bot \to \bot) = \top$
Stable-model semantics

Definition

A set $X$ of atoms is a *stable model* of a formula $F$ if $X$ is a minimal model of $F$.

Example: $F = (\neg p \rightarrow q) \land (\neg q \rightarrow p)$, $X = \{p\}$

- $F^X = (\bot \rightarrow q) \land ((\bot \rightarrow \bot) \rightarrow p)$ (which is equivalent to $p$)
- $X$ is a minimal model of $F^X$, so a stable model.

Example: $F = (\neg p \rightarrow q) \land (\neg q \rightarrow p)$, $X = \{p, q\}$

- $F^X = (\bot \rightarrow q) \land (\bot \rightarrow p)$ (which is equivalent to $\top$)
- $X$ is not a minimal model of $F^X$, so not a stable model.
Stable-model semantics

**Definition**

- A set $X$ of atoms is a **stable model** of a formula $F$ if $X$ is a minimal model of $F$

**Example:** $F = (\neg p \rightarrow q) \land (\neg q \rightarrow p)$, $X = \{p\}$

- $F^X = (\bot \rightarrow q) \land ((\bot \rightarrow \bot) \rightarrow p)$ (which is equivalent to $p$)
- $X$ is a minimal model of $F^X$, so a stable model

**Example:** $F = (\neg p \rightarrow q) \land (\neg q \rightarrow p)$, $X = \{p, q\}$

- $F^X = (\bot \rightarrow q) \land (\bot \rightarrow p)$ (which is equivalent to $\top$)
- $X$ is not a minimal model of $F^X$, so not a stable model
## Stable-model semantics

### Definition

- A set $X$ of atoms is a **stable model** of a formula $F$ if $X$ is a minimal model of $F$.

### Example: $F = (\neg p \rightarrow q) \land (\neg q \rightarrow p)$, $X = \{p\}$

- $F^X = (\bot \rightarrow q) \land ((\bot \rightarrow \bot) \rightarrow p)$ (which is equivalent to $p$)
- $X$ is a minimal model of $F^X$, so a stable model

### Example: $F = (\neg p \rightarrow q) \land (\neg q \rightarrow p)$, $X = \{p, q\}$

- $F^X = (\bot \rightarrow q) \land (\bot \rightarrow p)$ (which is equivalent to $\top$)
- $X$ is not a minimal model of $F^X$, so not a stable model
Stable-model semantics

Properties

- If $X$ is a stable model of a formula $F$ then $X$ consists of head atoms of $F$
- A least model of a Horn formula (conjunction of definite Horn clauses given as implications) is a unique stable model of the theory
- A set $X$ is a stable model of a formula $F \land \neg G$ if and only if $X$ is a stable model of $F$ and $X \models \neg G$
Strong equivalence

- Formulas $F$ and $F'$ are strongly equivalent if for every formula $G$, $F \land G$ and $F' \land G$ have the same stable models.
- $(X, Y)$ is an se-model of $F$ if $Y \subseteq \text{At}$, $X \subseteq Y$, $Y \models F$ and $X \models F^Y$.
- The following conditions are equivalent:
  - Formulas $F$ and $G$ are strongly equivalent.
  - For every set $X$ of atoms, $F^X$ and $G^X$ are equivalent in classical logic.
  - $F$ and $G$ have the same se-models.
  - $F$ and $G$ are equivalent in the logic here-and-there (details later).
Splitting

- Let $F$ and $G$ be formulas such that $F$ does not contain any of the head atoms of $G$.
- A set $X$ is a stable model of $F \land G$ iff there is a stable model $Y$ of $F$ such that $X$ is a stable model of $G \land \land Y$. 
Multivalued semantics

2-input one-step operator $\Phi_P$

- Given two interpretations $I$ and $J$
  \[
  \Phi_P(I, J) = \{ \text{hd}(r) : r \in P, \ bd^+(r) \subseteq I, \ bd^-(r) \cap J = \emptyset \}\n  \]
- $\Phi_P(\cdot, J)$ monotone
  - $I \subseteq I' \Rightarrow \Phi_P(I, J) \subseteq \Phi_P(I', J)$
- $\Phi_P(I, \cdot)$ antimonotone
  - $J \subseteq J' \Rightarrow \Phi_P(I, J') \subseteq \Phi_P(I, J)$
- $\Phi_P(I, I) = \mathcal{T}_P(I)$
Multivalued semantics: 4-val interpretations

Pairs \((I, J)\) of 2-val interpretations

- Atoms in \(I\) are **known** and atoms in \(J\) are **possible**
- Give rise to 4 truth values
  - If \(a \in I \cap J\), \(a\) is **true**
  - If \(a \notin I \cup J\), \(a\) is **false**
  - If \(a \in J \setminus I\), \(a\) is **unknown**
  - If \(a \in I \setminus J\), \(a\) is **overdefined** (inconsistent)
- \((I, J)\) **consistent** if \(I \subseteq J\)

Alternatively

- Functions \(val\) from \(At\) to \(\{t, f, u, i\}\)
- \(I := \{a \mid val(a) = t \text{ or } val(a) = i\}\)
- \(J := \{a \mid val(a) = t \text{ or } val(a) = u\}\)
Multivalued semantics: 4-val interpretations

<table>
<thead>
<tr>
<th>Pairs $(I, J)$ of 2-val interpretations</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Atoms in $I$ are <strong>known</strong> and atoms in $J$ are <strong>possible</strong></td>
</tr>
<tr>
<td>▶ Give rise to 4 truth values</td>
</tr>
<tr>
<td>▶ If $a \in I \cap J$, $a$ is true</td>
</tr>
<tr>
<td>▶ If $a \notin I \cup J$, $a$ is false</td>
</tr>
<tr>
<td>▶ If $a \in J \setminus I$, $a$ is unknown</td>
</tr>
<tr>
<td>▶ If $a \in I \setminus J$, $a$ is overdefined (inconsistent)</td>
</tr>
<tr>
<td>▶ $(I, J)$ <strong>consistent</strong> if $I \subseteq J$</td>
</tr>
</tbody>
</table>

Alternatively

<table>
<thead>
<tr>
<th>Functions $val$ from $At$ to ${t, f, u, i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ $I := {a \mid val(a) = t \text{ or } val(a) = i}$</td>
</tr>
<tr>
<td>▶ $J := {a \mid val(a) = t \text{ or } val(a) = u}$</td>
</tr>
</tbody>
</table>
Multivalued semantics

4-val one-step provability operator

- \( \mathcal{I}_P(I, J) = (\Phi_P(I, J), \Phi_P(J, I)) \)

- Precision (information) ordering:
  \((I, J) \leq_i (I', J') \)  - if \( I \subseteq I' \) and \( J' \subseteq J \)

- \( \mathcal{I}_P \) monotone wrt \( \leq_i \)

- \((I, J) \leq_i (I', J') \)  \( \Rightarrow \)  \( \mathcal{I}_P(I, J) \leq_i \mathcal{I}_P(I', J') \)
  - We have: \( I \subseteq I' \) and \( J' \subseteq J \)
    - \( \Phi_P(I, J) \subseteq \Phi_P(I', J) \) (monotonicity of \( \Phi_P(\cdot, J) \))
    - \( \Phi_P(I, J') \subseteq \Phi_P(I, J) \) (antimonotonicity of \( \Phi_P(I, \cdot) \))

\((I, J)\) consistent  \( \Rightarrow \)  \( \mathcal{I}_P(I, J)\) consistent

- Let \( I \subseteq J \)
  - \( \Rightarrow \)  \( \Phi_P(I, J) \subseteq \Phi_P(I, I) \subseteq \Phi_P(J, I) \)
Multivalued semantics

4-val one-step provability operator

- \( \mathcal{I}_P(I, J) = (\Phi_P(I, J), \Phi_P(J, I)) \)
- Precision (information) ordering:
  \( (I, J) \leq_i (I', J') \) - if \( I \subseteq I' \) and \( J' \subseteq J \)
- \( \mathcal{I}_P \) monotone wrt \( \leq_i \)
- \( (I, J) \leq_i (I', J') \) \( \Rightarrow \) \( \mathcal{I}_P(I, J) \leq_i \mathcal{I}_P(I', J') \)
  - We have: \( I \subseteq I' \) and \( J' \subseteq J \)
  - \( \Phi_P(I, J) \subseteq \Phi_P(I', J) \) (monotonicity of \( \Phi_P(\cdot, J) \))
  - \( \Phi_P(I, J') \subseteq \Phi_P(I, J) \) (antimonotonicity of \( \Phi_P(I, \cdot) \))

\( (I, J) \) consistent \( \Rightarrow \) \( \mathcal{I}_P(I, J) \) consistent

- Let \( I \subseteq J \)
- \( \Rightarrow \) \( \Phi_P(I, J) \subseteq \Phi_P(I, I) \subseteq \Phi_P(J, I) \)
4-val supported models

Recall: \( \mathcal{I}_P(I, J) = (\Phi_P(I, J), \Phi_P(J, I)) \) and \( T_P(I) = \Phi_P(I, I) \)

- \((I, J)\) is a 4-val supported model of \( P \) if \((I, J) = \mathcal{I}_P(I, J)\)
- \((I, I)\) is a 4-val supported model \textit{iff} \( I \) is a supported model
  - \((I, I) = \mathcal{I}_P(I, I) \text{ iff } (I, I) = (\Phi_P(I, I), \Phi_P(I, I)) = (T_P(I), T_P(I))\)
- The least 4-val supported model exists!
  - \( \mathcal{I}_P \) is monotone and so has the least (wrt \( \leq_i \)) fixpoint
  - Moreover, it is consistent!
- Kripke-Kleene (Fitting) fixpoint or semantics: \((KK^t(P), KK^p(P))\)
4-val Gelfond-Lifschitz operator

\[ \mathcal{GL}_P(I, J) = (\mathcal{GL}_P(J), \mathcal{GL}(I)) \]

Also monotone wrt \( \leq_i \)

\((I, J)\) is a 4-val stable model if \( \mathcal{GL}_P(I, J) = (I, J) \)

\( M \) is a stable model of \( P \) if and only if \((M, M)\) is a 4-val stable model of \( P \)

The least fixpoint of \( \mathcal{GL} \) exists!! (by monotonicity)

And is consistent

- if \( I \subseteq J \) then: \( \mathcal{GL}_P(J) \subseteq \mathcal{GL}(I) \) (antimonotonicity)

Well-founded fixpoint (semantics): \((WF^t(P), WF^p(P))\)

For every stable model \( M \) of \( P \)

\[ WF^t(P) \subseteq M \subseteq WF^p(P) \]
### Syntax

- **Connectives:** \( \bot, \lor, \land, \rightarrow \)
- **Formulas** - standard extension of atoms by means of connectives
  - \( \neg \varphi \) - shorthand for \( \varphi \rightarrow \bot \)
  - \( \varphi \leftrightarrow \psi \) - shorthand for \( (\varphi \rightarrow \psi) \land (\psi \rightarrow \varphi) \)
- **Language** \( \mathcal{L}_{ht} \)
Why important?

- Disjunctive logic programs — special theories in $L_{ht}$
  - $a_1 | \ldots | a_k \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$
  - $b_1 \land \ldots \land b_m \land \neg c_1 \land \ldots \land \neg c_n \rightarrow c_1 \lor \ldots \lor c_n$

- General logic programs (Ferraris, Lifschitz) = theories in $L_{ht}$
  - answer-set semantics extends to general logic programs
  - equilibrium models in logic $ht$
  - the two coincide!
Entailment in logic here-and-there

Ht-interpretations

- Pairs $\langle H, T \rangle$, where $H \subseteq T$ are sets of atoms
- Kripke interpretations with two worlds “here” and “there”
  - $H$ determines the valuation for “here”
  - $T$ determines the valuation for “there”

Kripke-model satisfiability in the world “here” $\models_{ht}$

- $\langle H, T \rangle \not\models_{ht} \bot$
- $\langle H, T \rangle \models_{ht} p$ if $p \in H$ (for atoms only)
- $\langle H, T \rangle \models_{ht} \varphi \land \psi$ and $\langle H, T \rangle \models_{ht} \varphi \lor \psi$ — standard recursion
- $\langle H, T \rangle \models_{ht} \varphi \rightarrow \psi$ if
  - $\langle H, T \rangle \not\models_{ht} \varphi$ or $\langle H, T \rangle \models_{ht} \psi$
  - $T \models \varphi \rightarrow \psi$ (in standard propositional logic).
Entailment in logic here-and-there

Ht-interpretations

- Pairs \( \langle H, T \rangle \), where \( H \subseteq T \) are sets of atoms
- Kripke interpretations with two worlds “here” and “there”
  - \( H \) determines the valuation for “here”
  - \( T \) determines the valuation for “there”

Kripke-model satisfiability in the world “here” \( \models_{ht} \)

- \( \langle H, T \rangle \not \models_{ht} \bot \)
- \( \langle H, T \rangle \models_{ht} p \) if \( p \in H \) (for atoms only)
- \( \langle H, T \rangle \models_{ht} \varphi \land \psi \) and \( \langle H, T \rangle \models_{ht} \varphi \lor \psi \) — standard recursion
- \( \langle H, T \rangle \models_{ht} \varphi \rightarrow \psi \) if
  - \( \langle H, T \rangle \not \models_{ht} \varphi \) or \( \langle H, T \rangle \models_{ht} \psi \)
  - \( T \models \varphi \rightarrow \psi \) (in standard propositional logic).
Entailment in logic here-and-there

**ht-model, ht-validity, ht-equivalence**

- If $\langle H, T \rangle \models_{ht} \varphi$ - $\langle H, T \rangle$ is an **ht-model** of $\varphi$
- $\varphi$ is **ht-valid** if for every **ht-model** $\langle H, T \rangle$, $\langle H, T \rangle \models \varphi$
- $\varphi$ and $\psi$ are **ht-equivalent** if they have the same **ht**-models

- $\varphi$ and $\psi$ are ht-equivalent iff $\varphi \leftrightarrow \psi$ is **ht-valid**
Proof theory

Natural deduction — sequents and rules

- Sequents $\Gamma \Rightarrow \varphi$ — “$\varphi$ under the assumptions $\Gamma$”
- Introduction rules for $\land$, $\lor$, $\rightarrow$
  \[
  \frac{\Gamma \Rightarrow \varphi \quad \Delta \Rightarrow \psi}{\Gamma, \Delta \Rightarrow \varphi \land \psi}
  \]
- Elimination rules for $\land$, $\lor$, $\rightarrow$
  \[
  \frac{\Gamma \Rightarrow \varphi \quad \Delta \Rightarrow \varphi \rightarrow \psi}{\Gamma, \Delta \Rightarrow \psi}
  \]
- Contradiction
  \[
  \frac{\Gamma \Rightarrow \bot}{\Gamma \Rightarrow \varphi}
  \]
- Weakening
  \[
  \frac{\Gamma \Rightarrow \varphi}{\Gamma' \Rightarrow \varphi}
  \quad \text{for all $\Gamma'$, $\Gamma$ s.t. $\Gamma' \subseteq \Gamma$}
  \]
### Axiom schemas

- **(AS1)** \( \varphi \Rightarrow \varphi \)
- **(AS2)** \( \Rightarrow \varphi \lor \neg \varphi \) (Excluded Middle)
- **(AS2')** \( \Rightarrow \neg \varphi \lor \neg \neg \varphi \) (Weak EM)
- **(AS2'')** \( \Rightarrow \varphi \lor (\varphi \rightarrow \psi) \lor \neg \psi \) (in between (AS2) and (AS2'))

### Logics through natural deduction

- Propositional logic: (AS1), (AS2)
- Intuitionistic logic: (AS1)
- Logic here-and-there: (AS1), (AS2'')
## Proof theory

### Axiom schemas

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Formula</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(AS1)</td>
<td>$\varphi \Rightarrow \varphi$</td>
<td></td>
</tr>
<tr>
<td>(AS2)</td>
<td>$\Rightarrow \varphi \lor \neg \varphi$</td>
<td>(Excluded Middle)</td>
</tr>
<tr>
<td>(AS2′)</td>
<td>$\Rightarrow \neg \varphi \lor \neg \neg \varphi$</td>
<td>(Weak EM)</td>
</tr>
<tr>
<td>(AS2″)</td>
<td>$\Rightarrow \varphi \lor (\varphi \rightarrow \psi) \lor \neg \psi$</td>
<td>(in between (AS2) and (AS2′))</td>
</tr>
</tbody>
</table>

### Logics through natural deduction

- Propositional logic: (AS1), (AS2)
- Intuitionistic logic: (AS1)
- Logic here-and-there: (AS1),(AS2″)
Bringing the two together

Soundness and completeness

- A formula is a theorem of $ht$ if and only if it is $ht$-valid

In particular

- $\varphi$ and $\psi$ are $ht$-equivalent iff $\Rightarrow \varphi \leftrightarrow \psi$ is a theorem of $ht$
Bringing the two together

Soundness and completeness

- A formula is a theorem of $ht$ if and only if it is $ht$-valid

In particular

- $\varphi$ and $\psi$ are $ht$-equivalent iff $\Rightarrow \varphi \leftrightarrow \psi$ is a theorem of $ht$
Equilibrium models, Pearce 1997

- \( \langle T, T \rangle \) is an equilibrium model of a set \( A \) of formulas if
  - \( \langle T, T \rangle \models_{ht} A \), and
  - for every \( H \subseteq T \) such that \( \langle H, T \rangle \models_{ht} A \), \( H = T \)

Key connection

- A set \( M \) of atoms is an answer set of a disjunctive logic program \( P \) (general logic program \( P \)) if and only if \( \langle M, M \rangle \) is an equilibrium model for \( P \)
Equilibrium models, Pearce 1997

\( \langle T, T \rangle \) is an \emph{equilibrium model} of a set \( A \) of formulas if

1. \( \langle T, T \rangle \models_{ht} A \), and
2. for every \( H \subseteq T \) such that \( \langle H, T \rangle \models_{ht} A \), \( H = T \)

Key connection

A set \( M \) of atoms is an answer set of a disjunctive logic program \( P \) (general logic program \( P \)) if and only if \( \langle M, M \rangle \) is an equilibrium model for \( P \)
Let $P$ and $Q$ be two (general) programs. The following conditions are equivalent:

- $P$ and $Q$ are strongly equivalent
- $P$ and $Q$ are $ht$-equivalent
- $P$ and $Q$ have the same $ht$-models
- $P \leftrightarrow Q$ is $ht$-valid
- $\Rightarrow P \leftrightarrow Q$ is a theorem of $ht$
Algebraic approach
The problem

Complex landscape of nonmonotonicity

- Multitude of formalisms
- Different intuitions
- Different languages
- Different semantics
- Complexity

Needed!

- Unifying abstract foundation
The problem

Complex landscape of nonmonotonicity

- Multitude of formalisms
- Different intuitions
- Different languages
- Different semantics
- Complexity

Needed!

- Unifying abstract foundation
A triumph of universal algebra

Basic lesson for this segment

- Major nonmonotonic systems
  - logic programming
  - default logic
  - autoepistemic logics

  can be given a unified algebraic treatment

- Each system can be assigned the same family of semantics

- Key concepts: lattices and bilattices, operators and fixpoints

- Key ideas: approximating operators and stable operators

- Key tool: Knaster-Tarski Theorem
Overview of approach

Generalize Fitting’s work on logic programming

- Central role of 4-valued van Emden-Kowalski operator $\mathcal{T}_P$
- Derived stable operator, $\Psi'_P$
- 2-valued and 3-valued supported models and Kripke-Kleene semantics described by fixpoints of $\mathcal{T}_P$
- 2-valued and 3-valued stable models and well-founded semantics described by fixpoints of $\Psi'_P$
Key definitions, some notation

- $\langle L, \leq \rangle$
  - $L$ is a nonempty set
  - $\leq$ is a partial order such that every two lattice elements have $\text{lub}$ (join) and $\text{glb}$ (meet)
- Elements of $L$ express
  - degree of truth
  - measure of knowledge
- $\leq$ - order of increased truth or knowledge
- Complete lattices (both bounds defined for all sets)
- $\bot, \top$
### Lattices - examples

#### Lattice  \( \text{TWO} \)
- \( \{f, t\} \)
- \( f \leq t \)

#### Lattice  \( \mathcal{A}_2 \)
- set of all 2-valued interpretations
- componentwise extension of the ordering from \( \text{TWO} \)

#### Lattice  \( \mathcal{W} \)
- family of sets of 2-valued interpretations
- \( W_1 \subseteq W_2 \) if \( W_2 \subseteq W_1 \)
### Lattices - examples

#### Lattice \( \mathcal{TW} \)
- \( \{ f, t \} \)
- \( f \leq t \)

#### Lattice \( \mathcal{A}_2 \)
- set of all 2-valued interpretations
- componentwise extension of the ordering from \( \mathcal{TW} \)

#### Lattice \( \mathcal{W} \)
- family of sets of 2-valued interpretations
- \( W_1 \subseteq W_2 \) if \( W_2 \subseteq W_1 \)
Lattices - examples

Lattice $\mathcal{TWO}$

- $\{f, t\}$
- $f \leq t$

Lattice $\mathcal{A}_2$

- set of all 2-valued interpretations
- componentwise extension of the ordering from $\mathcal{TWO}$

Lattice $\mathcal{W}$

- family of sets of 2-valued interpretations
- $W_1 \subseteq W_2$ if $W_2 \subseteq W_1$
That’s what it’s all about!

- Truth or knowledge can be revised
- Revisions are described by operators on lattices
- Fixpoints — states of truth or knowledge that cannot be revised
An operator $O$ is monotone if $x \leq y$ implies $O(x) \leq O(y)$.

Knaster-Tarski Theorem: a monotone operator on a complete lattice has a least fixpoint.
Antimonotone operators

- An operator $O$ is antimonotone if $x \leq y$ implies $O(y) \leq O(x)$
- If $O$ is antimonotone then $O^2$ is monotone:
  \[
  x \leq y \implies O(y) \leq O(x) \implies O^2(x) \leq O^2(y)
  \]
- Oscillating pair: $(x, y)$ is an oscillating pair for an operator $O$ if $O(x) = y$ and $O^2(x) = x$
- Antimonotone operator $O$ has an extreme oscillating pair
  \[
  (\text{lfp}(O^2), \text{gfp}(O^2))
  \]
Approximations and bilattices

Key definitions, some notation

- A pair \((x, y)\) approximates an element \(z\) if \(x \leq z \leq y\)
- Orderings of approximations:
  - information (or precision) ordering: \((x_1, y_1) \leq_i (x_2, y_2)\) iff \(x_1 \leq x_2\) and \(y_2 \leq y_1\)
  - truth ordering: \((x_1, y_1) \leq_t (x_2, y_2)\) iff \(x_1 \leq x_2\) and \(y_1 \leq y_2\)
- Bilattice \(\langle L^2, \leq_i, \leq_t \rangle\)
- A pair \((x, y)\) is consistent if \(x \leq y\), and inconsistent, otherwise
- An element \((x, y)\) is complete if \(x = y\)
Bilattices - examples

Bilattice **FOUR**

- set of all pairs of 2-valued interpretations (identified with 4-valued interpretations)
- componentwise extension of the orderings from **FOUR**
Bilattices - examples

Bilattice FOUR

<table>
<thead>
<tr>
<th>≤i</th>
<th>(t, f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f, f)</td>
<td>(t, t)</td>
</tr>
<tr>
<td>(f, t)</td>
<td></td>
</tr>
</tbody>
</table>

Bilattice $A_4$

- set of all pairs of 2-valued interpretations (identified with 4-valued interpretations)
- componentwise extension of the orderings from $FOUR$
Bilattices - examples, cont’d

**Bilattice \( \mathcal{B} \)**

- Family of pairs of sets of 2-valued interpretations
- *Belief pairs*
  - \((P_1, S_1) \sqsubseteq_i (P_2, S_2)\) if \(P_2 \subseteq P_1\) and \(S_1 \subseteq S_2\)
  - \((P_1, S_1) \sqsubseteq_t (P_2, S_2)\) if \(P_2 \subseteq P_1\) and \(S_2 \subseteq S_1\)
Approximating operators

Key definitions, some notation

- $A : L^2 \to L^2$ approximates $O : L \to L$ if
  - $A(x, x) = (O(x), O(x))$
  - $A$ is $\leq_i$-monotone
  - $A$ is symmetric: $A^1(x, y) = A^2(y, x)$, where $A(x, y) = (A^1(x, y), A^2(x, y))$

Properties

- Approximating operators are consistent
- Complete fixpoints of $A$ correspond to fixpoints of $O$
- Every fixpoint of $A$ is approximated by the least fixpoint of $A$: Kripke-Kleene fixpoint of $A$
- Kripke-Kleene fixpoint of an approximating operator is consistent
Approximating operators

Key definitions, some notation

- $A : L^2 \rightarrow L^2$ approximates $O : L \rightarrow L$ if
  - $A(x, x) = (O(x), O(x))$
  - $A$ is $\leq_i$-monotone
  - $A$ is symmetric: $A^1(x, y) = A^2(y, x)$, where $A(x, y) = (A^1(x, y), A^2(x, y))$

Properties

- Approximating operators are consistent
- Complete fixpoints of $A$ correspond to fixpoints of $O$
- Every fixpoint of $A$ is approximated by the least fixpoint of $A$: Kripke-Kleene fixpoint of $A$
- Kripke-Kleene fixpoint of an approximating operator is consistent
Stable operators

- If $A : L^2 \to L^2$ is $\leq_i$-monotone then $A^1(\cdot, y)$ and $A^2(x, \cdot)$ are monotone
- For $\leq_i$-monotone operator $A : L^2 \to L^2$ define:
  \[
  C_A^l(y) = \text{lfp}(A^1(\cdot, y)) \quad \text{and} \quad C_A^u(x) = \text{lfp}(A^2(x, \cdot))
  \]
- Since $A$ is symmetric, $C_A^l = C_A^u = C_A$
- Stable operator for $A$:
  \[
  C_A(x, y) = (C_A(y), C_A(x))
  \]
- Stable fixpoints (relative to $C_A$)
- $\leq_i$-least fixpoint of $C_A$ — well-founded (WF) fixpoint of $A$
Properties of stable operators

All quite easy to prove, in fact

- $C_A$ is antimonotone
- $C_A$ is $\leq_i$-monotone and $\leq_t$-antimonotone
- Fixpoints of $C_A$ are $\leq_t$-minimal fixpoints of $A$
- Complete fixpoints of $C_A$ correspond to fixpoints of $C_A$
- Complete fixpoints of $C_A$ are fixpoints of $O$
- K-K fixpoint of $A \leq_i$ WF fixpoint of $A$
## Logic programming — case study 1

### Fitting

- Lattice $\mathcal{A}_2$, bilattice $\mathcal{A}_4$
- Operators associated with program $P$
  - 2-valued van Emden-Kowalski operator $T_P$
  - Its approximation: 4-valued van Emden-Kowalski operator $\mathcal{I}_P$
  - 2-valued stable operator (Gelfond-Lifschitz operator $GL_P$)
  - Stable operator $C_P$ of $\mathcal{I}_P$ (operator $\Psi_P'$ of Przymusinski)
- Semantics
  - Supported models: fixpoints of the operator $\mathcal{I}_P$ ($T_P$)
  - Kripke-Kleene semantics: least fixpoint of $\mathcal{I}_P$
  - Stable models: fixpoints of the operator $C_P$ ($C_P$)
  - Well-founded semantics: least fixpoint of $C_P$
Logic programming explained

Central role of $\mathcal{I}_P$

\[ \begin{array}{c}
\mathcal{I}_P \\
T_P \\
C_P \\
\end{array} \]
Truth assignment function $\mathcal{H}_{V,I}$

- For atom $p$: $\mathcal{H}_{V,I}(p) = I(p)$
- The boolean connectives — standard way
  - $\mathcal{H}_{V,I}(KF) = t$, if for every $J \in V$, $\mathcal{H}_{V,I}(F) = t$
  - $\mathcal{H}_{V,I}(KF) = f$, otherwise

AE models, expansions

- Moore’s operator $D_T: \mathcal{W} \rightarrow \mathcal{W}$
  $$D_T(V) = \{ I: \mathcal{H}_{V,I}(T) = t \}$$
- Fixpoints of $D_T$ — autoepistemic models of $T$
- Autoepistemic models generate expansions
Truth assignment function $\mathcal{H}_{V,I}$

- For atom $p$: $\mathcal{H}_{V,I}(p) = I(p)$
- The boolean connectives — standard way
  - $\mathcal{H}_{V,I}(KF) = \text{t}$, if for every $J \in V$, $\mathcal{H}_{V,J}(F) = \text{t}$
  - $\mathcal{H}_{V,I}(KF) = \text{f}$, otherwise

AE models, expansions

- Moore’s operator $D_T : \mathcal{W} \rightarrow \mathcal{W}$

  $$D_T(V) = \{I : \mathcal{H}_{V,I}(T) = \text{t}\}$$

- Fixpoints of $D_T$ — autoepistemic models of $T$
- Autoepistemic models generate expansions
The setting

- Lattice $\mathcal{W}$, bilattice $\mathcal{B}$
- $\mathcal{H}_{(V, V')}^4, I$
- Approximating operator for $D_T$ — $\mathcal{D}_T$ (DMT 98)

$$D_T(V, V') = (\{I : \mathcal{H}_{(V, V')}^4, I(T) \geq_t (f, t)\}, \{I : \mathcal{H}_{(V, V')}^4, I(T) \geq_t (t, f)\})$$

- Complete fixpoints of $\mathcal{D}_T$ — autoepistemic models of $T$
- The least fixpoint of $\mathcal{D}_T$ — Kripke-Kleene fixpoint
  - approximates all autoepistemic models of $T$
- The stable operator for $\mathcal{D}_T$: $C_T(V, V') = (C_T(V'), C_T(V))$
- What are the fixpoints of $C_T$?
Autoepistemic logic explained

Central role of $D_T$

Diagram:

$D_T$  $D_T$

$D_T$  $C_T$

$C_T$
Same setting as for AEL

- Lattice \( \mathcal{W} \), bilattice \( \mathcal{B} \)

- \( \mathcal{H}_V, l(\varphi) = I(\varphi) \), for every formula \( \varphi \)

- \( d = \frac{\alpha : \beta_1, \ldots, \beta_k}{\gamma} \)

- \( \mathcal{H}_V, l(d) = t \) iff
  - there is \( J \in V \) such that \( J(\alpha) = f \), or
  - there is \( i, 1 \leq i \leq k \) such that for every \( J \in V \), \( J(\beta_i) = f \), or
  - \( l(\gamma) = t \)

- Weak-extension operator \( E_\Delta \) (\( \Delta \) — default theory):
  \[
  E_\Delta(V) = \{ I \in \mathcal{A}_2 : \mathcal{H}_V, l(\Delta) = t \}
  \]

- Fixpoints of \( E_\Delta(V) \) — default models of weak extensions of \( \Delta \)
4-valued truth assignment, approximating operator

- $\mathcal{H}^4_{(V,V'),I}$
- Approximating operator for $E_\Delta \models E_\Delta$

$$E_\Delta(V, V') = (\{ I : \mathcal{H}^4_{(V,V'),I}(\Delta) \geq_t (f, t) \}, \{ I : \mathcal{H}^4_{(V,V'),I}(\Delta) \geq_t (t, f) \})$$

- Complete fixpoints of $E_\Delta \models$ models of weak extensions of $\Delta$
- The least fixpoint of $E_\Delta \models$ Kripke-Kleene fixpoint
  - approximates all default models of weak extensions of $\Delta$
Stable operator

- The stable operator for $E_\Delta$:
  
  $C_\Delta (V, V') = (C_\Delta (V'), C_\Delta (V))$

- $C_\Delta$ — Guerreiro-Casanova operator $\Sigma_\Delta$

- Fixpoints of $C_\Delta$ — default models of Reiter’s extensions

- Consistent fixpoints of $C_\Delta$ — stationary extensions by Przymusinski

- Well-founded fixpoint of $E_\Delta$ (least fixpoint of $C_\Delta$ — well-founded semantics of default logic by Baral and Subrahmanian)
Central role of $\mathcal{E}_\Delta$
Strong parallels!

\[ c \leftarrow a, \text{not } b \Rightarrow \frac{a \text{--} b}{c} \]
Connections

Strong parallels!

\[
\begin{align*}
T_P & \rightarrow C_P & E_\Delta & \rightarrow C_\Delta \\
C_P & \rightarrow T_P & C_\Delta & \rightarrow D_T \\
& & & C_T
\end{align*}
\]

\[
c \leftarrow a, \text{ not } b \quad \Rightarrow \quad \frac{a \rightarrow \neg b}{c}
\]

\[
\begin{align*}
\begin{array}{c}
\alpha : \beta \\
\gamma
\end{array} & \quad \Rightarrow \quad K\alpha \land \neg K\neg \beta \supset \gamma
\end{align*}
\]
Thank you!