Answer Set Programming: Computation & Characterization

Sebastian Rudolph

Computational Logic Group
Technische Universität Dresden

Slides based on a lecture by Martin Gebser and Torsten Schaub.
Potassco Slide Packages are licensed under a Creative Commons Attribution 3.0 Unported License.
1 Consequence operator
2 Computation from first principles
3 Complexity
4 Completion
5 Tightness
6 Loops and Loop Formulas
Let $P$ be a positive program and $X$ a set of atoms.

The consequence operator $T_P$ is defined as follows:

$$T_P X = \{ \text{head}(r) \mid r \in P \text{ and } \text{body}(r) \subseteq X \}$$

Iterated applications of $T_P$ are written as $T_P^j$ for $j \geq 0$, where

- $T_P^0 X = X$ and
- $T_P^i X = T_P T_P^{i-1} X$ for $i \geq 1$

For any positive program $P$, we have

- $\text{Cn}(P) = \bigcup_{i \geq 0} T_P^i \emptyset$
- $X \subseteq Y$ implies $T_P X \subseteq T_P Y$
- $\text{Cn}(P)$ is the smallest fixpoint of $T_P$
Let $P$ be a positive program and $X$ a set of atoms

The consequence operator $T_P$ is defined as follows:

$$T_P X = \{ \text{head}(r) \mid r \in P \text{ and } \text{body}(r) \subseteq X \}$$

Iterated applications of $T_P$ are written as $T_P^j$ for $j \geq 0$, where

- $T_P^0 X = X$ and
- $T_P^i X = T_P T_P^{i-1} X$ for $i \geq 1$

For any positive program $P$, we have

- $Cn(P) = \bigcup_{i \geq 0} T_P^i \emptyset$
- $X \subseteq Y$ implies $T_P X \subseteq T_P Y$
- $Cn(P)$ is the smallest fixpoint of $T_P$
Let $P$ be a positive program and $X$ a set of atoms.

The consequence operator $T_P$ is defined as follows:

$$T_P X = \{ \text{head}(r) \mid r \in P \text{ and } \text{body}(r) \subseteq X \}$$

Iterated applications of $T_P$ are written as $T_P^j$ for $j \geq 0$, where

- $T_P^0 X = X$ and
- $T_P^i X = T_P T_P^{i-1} X$ for $i \geq 1$

For any positive program $P$, we have

- $Cn(P) = \bigcup_{i \geq 0} T_P^i \emptyset$
- $X \subseteq Y$ implies $T_P X \subseteq T_P Y$
- $Cn(P)$ is the smallest fixpoint of $T_P$
Consequence operator

An example

Consider the program

\[ P = \{ p \leftarrow, q \leftarrow, r \leftarrow p, s \leftarrow q, t \leftarrow r, u \leftarrow v \} \]

We get

\[
\begin{align*}
T_P^0 \emptyset &= \emptyset \\
T_P^1 \emptyset &= \{ p, q \} = T_P T_P^0 \emptyset = T_P \emptyset \\
T_P^2 \emptyset &= \{ p, q, r \} = T_P T_P^1 \emptyset = T_P \{ p, q \} \\
T_P^3 \emptyset &= \{ p, q, r, t \} = T_P T_P^2 \emptyset = T_P \{ p, q, r \} \\
T_P^4 \emptyset &= \{ p, q, r, t, s \} = T_P T_P^3 \emptyset = T_P \{ p, q, r, t \} \\
T_P^5 \emptyset &= \{ p, q, r, t, s \} = T_P T_P^4 \emptyset = T_P \{ p, q, r, t, s \} \\
T_P^6 \emptyset &= \{ p, q, r, t, s \} = T_P T_P^5 \emptyset = T_P \{ p, q, r, t, s \}
\end{align*}
\]

\[ Cn(P) = \{ p, q, r, t, s \} \] is the smallest fixpoint of \( T_P \) because

\[
\begin{align*}
T_P \{ p, q, r, t, s \} &= \{ p, q, r, t, s \} \text{ and } \\
T_P X &\neq X \text{ for each } X \subset \{ p, q, r, t, s \}
\end{align*}
\]
Consider the program

\[ P = \{ p \leftarrow, q \leftarrow, r \leftarrow p, s \leftarrow q, t \leftarrow r, u \leftarrow v \} \]

We get

\[
\begin{align*}
T^0_P \emptyset &= \emptyset \\
T^1_P \emptyset &= \{ p, q \} &= T_P T^0_P \emptyset &= T_P \emptyset \\
T^2_P \emptyset &= \{ p, q, r \} &= T_P T^1_P \emptyset &= T_P \{ p, q \} \\
T^3_P \emptyset &= \{ p, q, r, t \} &= T_P T^2_P \emptyset &= T_P \{ p, q, r \} \\
T^4_P \emptyset &= \{ p, q, r, t, s \} &= T_P T^3_P \emptyset &= T_P \{ p, q, r, t \} \\
T^5_P \emptyset &= \{ p, q, r, t, s \} &= T_P T^4_P \emptyset &= T_P \{ p, q, r, t, s \} \\
T^6_P \emptyset &= \{ p, q, r, t, s \} &= T_P T^5_P \emptyset &= T_P \{ p, q, r, t, s \}
\end{align*}
\]

\[ Cn(P) = \{ p, q, r, t, s \} \] is the smallest fixpoint of \( T_P \) because

\[ T_P \{ p, q, r, t, s \} = \{ p, q, r, t, s \} \] and

\[ T_P X \neq X \] for each \( X \subset \{ p, q, r, t, s \} \)
Consequence operator

An example

Consider the program

\[ P = \{ p \leftarrow, q \leftarrow, r \leftarrow p, s \leftarrow q, t, t \leftarrow r, u \leftarrow v \} \]

We get

\[
\begin{align*}
T^0_P \emptyset &= \emptyset \\
T^1_P \emptyset &= \{ p, q \} = T_P T^0_P \emptyset = T_P \emptyset \\
T^2_P \emptyset &= \{ p, q, r \} = T_P T^1_P \emptyset = T_P \{ p, q \} \\
T^3_P \emptyset &= \{ p, q, r, t \} = T_P T^2_P \emptyset = T_P \{ p, q, r \} \\
T^4_P \emptyset &= \{ p, q, r, t, s \} = T_P T^3_P \emptyset = T_P \{ p, q, r, t \} \\
T^5_P \emptyset &= \{ p, q, r, t, s \} = T_P T^4_P \emptyset = T_P \{ p, q, r, t, s \} \\
T^6_P \emptyset &= \{ p, q, r, t, s \} = T_P T^5_P \emptyset = T_P \{ p, q, r, t, s \}
\end{align*}
\]

\[ Cn(P) = \{ p, q, r, t, s \} \] is the smallest fixpoint of \( T_P \) because

\[ T_P \{ p, q, r, t, s \} = \{ p, q, r, t, s \} \] and

\[ T_P X \neq X \] for each \( X \subset \{ p, q, r, t, s \} \)
Outline

1. Consequence operator
2. Computation from first principles
3. Complexity
4. Completion
5. Tightness
6. Loops and Loop Formulas
Approximating stable models

**First Idea** Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
- $L$ and $U$ constitute lower and upper bounds on $X$
- $L$ and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program

**Observation**

\[ X \subseteq Y \implies P^Y \subseteq P^X \implies Cn(P^Y) \subseteq Cn(P^X) \]

**Properties** Let $X$ be a stable model of normal logic program $P$
- If $L \subseteq X$, ...
Approximating stable models

- **First Idea** Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program

- **Observation**
  
  $X \subseteq Y$ implies $P^Y \subseteq P^X$ implies $\text{Cn}(P^Y) \subseteq \text{Cn}(P^X)$

- **Properties** Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, ...
Approximating stable models

- **First Idea**  Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(A \setminus U)$ describe a three-valued model of the program

- **Observation**

  $X \subseteq Y$ implies $P^Y \subseteq P^X$ implies $Cn(P^Y) \subseteq Cn(P^X)$

- **Properties**  Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$
Approximating stable models

First Idea  Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
- $L$ and $U$ constitute lower and upper bounds on $X$
- $L$ and $(A \setminus U)$ describe a three-valued model of the program

Observation

$X \subseteq Y$ implies $P^Y \subseteq P^X$ implies $Cn(P^Y) \subseteq Cn(P^X)$

Properties  Let $X$ be a stable model of normal logic program $P$
- If $L \subseteq X$, then $X \subseteq Cn(P^L)$
- If $X \subseteq U$, then $Cn(P^U) \subseteq X$
- If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$
Approximating stable models

- **First Idea** Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program

- **Observation**
  \[
  X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } \text{Cn}(P^Y) \subseteq \text{Cn}(P^X)
  \]

- **Properties** Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, then $X \subseteq \text{Cn}(P^L)$
  - If $X \subseteq U$, then $\text{Cn}(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup \text{Cn}(P^U) \subseteq X \subseteq U \cap \text{Cn}(P^L)$
Approximating stable models

- **First Idea** Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program

- **Observation**
  
  $$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } \text{Cn}(P^Y) \subseteq \text{Cn}(P^X)$$

- **Properties** Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, then $X \subseteq \text{Cn}(P^L)$
  - If $X \subseteq U$, then $\text{Cn}(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup \text{Cn}(P^U) \subseteq X \subseteq U \cap \text{Cn}(P^L)$
Approximating stable models

- **First Idea** Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program

- **Observation**

  $X \subseteq Y$ implies $P^Y \subseteq P^X$ implies $Cn(P^Y) \subseteq Cn(P^X)$

- **Properties** Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$
Approximating stable models

- **First Idea**: Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
  - $L$ and $U$ constitute lower and upper bounds on $X$
  - $L$ and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program

- **Observation**

$$X \subseteq Y \implies P^Y \subseteq P^X \implies \text{Cn}(P^Y) \subseteq \text{Cn}(P^X)$$

- **Properties**: Let $X$ be a stable model of normal logic program $P$
  - If $L \subseteq X$, then $X \subseteq \text{Cn}(P^L)$
  - If $X \subseteq U$, then $\text{Cn}(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup \text{Cn}(P^U) \subseteq X \subseteq U \cap \text{Cn}(P^L)$
Approximating stable models

First Idea  Approximate a stable model $X$ by two sets of atoms $L$ and $U$ such that $L \subseteq X \subseteq U$
- $L$ and $U$ constitute lower and upper bounds on $X$
- $L$ and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program

Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

Properties  Let $X$ be a stable model of normal logic program $P$
- If $L \subseteq X$, then $X \subseteq Cn(P^L)$
- If $X \subseteq U$, then $Cn(P^U) \subseteq X$
- If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$
Approximating stable models

■ Second Idea

\[ \text{repeat} \]
\[ \text{replace } L \text{ by } L \cup Cn(P^U) \]
\[ \text{replace } U \text{ by } U \cap Cn(P^L) \]
\[ \text{until } L \text{ and } U \text{ do not change anymore} \]

■ Observations

■ At each iteration step
  ■ \( L \) becomes larger (or equal)
  ■ \( U \) becomes smaller (or equal)
/// \( L \subseteq X \subseteq U \) is invariant for every stable model \( X \) of \( P \)

■ If \( L \nsubseteq U \), then \( P \) has no stable model
■ If \( L = U \), then \( L \) is a stable model of \( P \)
Approximating stable models

Second Idea

repeat
    replace $L$ by $L \cup Cn(P^U)$
    replace $U$ by $U \cap Cn(P^L)$
until $L$ and $U$ do not change anymore

Observations

- At each iteration step
  - $L$ becomes larger (or equal)
  - $U$ becomes smaller (or equal)
- $L \subseteq X \subseteq U$ is invariant for every stable model $X$ of $P$
  - If $L \nsubseteq U$, then $P$ has no stable model
  - If $L = U$, then $L$ is a stable model of $P$
Approximating stable models

- Second Idea

  repeat
  
  replace $L$ by $L \cup Cn(P^U)$
  
  replace $U$ by $U \cap Cn(P^L)$
  
  until $L$ and $U$ do not change anymore

- Observations

  - At each iteration step
    - $L$ becomes larger (or equal)
    - $U$ becomes smaller (or equal)
  
  - $L \subseteq X \subseteq U$ is invariant for every stable model $X$ of $P$

  - If $L \not\subseteq U$, then $P$ has no stable model
  
  - If $L = U$, then $L$ is a stable model of $P$
Approximating stable models

■ Second Idea

\[
\text{repeat} \\
\quad \text{replace } L \text{ by } L \cup Cn(P^U) \\
\quad \text{replace } U \text{ by } U \cap Cn(P^L) \\
\text{until } L \text{ and } U \text{ do not change anymore}
\]

■ Observations

■ At each iteration step
  ■ \( L \) becomes larger (or equal)
  ■ \( U \) becomes smaller (or equal)

■ \( L \subseteq X \subseteq U \) is invariant for every stable model \( X \) of \( P \)

■ If \( L \not\subseteq U \), then \( P \) has no stable model
■ If \( L = U \), then \( L \) is a stable model of \( P \)
The simplistic expand algorithm

\[
\text{expand}_P(L, U)\\
\text{repeat}\\
\quad L' \leftarrow L\\
\quad U' \leftarrow U\\
\quad L \leftarrow L' \cup C_n(P^{U'})\\
\quad U \leftarrow U' \cap C_n(P^{L'})\\
\quad \text{if } L \not\subseteq U \text{ then return}\\
\text{until } L = L' \text{ and } U = U'
\]
Computation from first principles

An example

\[ P = \begin{cases} 
  a \leftarrow \\
  b \leftarrow a, \sim c \\
  d \leftarrow b, \sim e \\
  e \leftarrow \sim d 
\end{cases} \]

<table>
<thead>
<tr>
<th>( L' )</th>
<th>( Cn(P^{U'}) )</th>
<th>( L )</th>
<th>( U' )</th>
<th>( Cn(P^{L'}) )</th>
<th>( U )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \emptyset )</td>
<td>{a}</td>
<td>{a}</td>
<td>{a, b, c, d, e}</td>
<td>{a, b, d, e}</td>
</tr>
<tr>
<td>2</td>
<td>{a}</td>
<td>{a, b}</td>
<td>{a, b}</td>
<td>{a, b, d, e}</td>
<td>{a, b, d, e}</td>
</tr>
<tr>
<td>3</td>
<td>{a, b}</td>
<td>{a, b}</td>
<td>{a, b}</td>
<td>{a, b, d, e}</td>
<td>{a, b, d, e}</td>
</tr>
</tbody>
</table>

**Note** We have \( \{a, b\} \subseteq X \) and \((A \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset \) for every stable model \( X \) of \( P \).
Computation from first principles

An example

\[ P = \left\{ \begin{array}{l}
a \leftarrow \\
b \leftarrow a, \sim c \\
d \leftarrow b, \sim e \\
e \leftarrow \sim d \\
\end{array} \right\} \]

\[ \begin{array}{cccccc}
L' & Cn(P^{U'}) & L & U' & Cn(P^{L'}) & U \\
1 & \emptyset & \{a\} & \{a\} & \{a, b, c, d, e\} & \{a, b, d, e\} \\
2 & \{a\} & \{a, b\} & \{a, b\} & \{a, b, d, e\} & \{a, b, d, e\} \\
3 & \{a, b\} & \{a, b\} & \{a, b\} & \{a, b, d, e\} & \{a, b, d, e\} \\
\end{array} \]

Note: We have \( \{a, b\} \subseteq X \) and \( (A \setminus \{a, b, d, e\} \cap X = (\{c\} \cap X) = \emptyset \) for every stable model \( X \) of \( P \).
**An example**

\[
\begin{align*}
P = \left\{ & a \leftarrow \\
            & b \leftarrow a, \sim c \\
            & d \leftarrow b, \sim e \\
            & e \leftarrow \sim d \\
\right. \end{align*}
\]

<table>
<thead>
<tr>
<th>(L')</th>
<th>(Cn(P^{U'}))</th>
<th>(L)</th>
<th>(U')</th>
<th>(Cn(P^{L'}))</th>
<th>(U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{a}</td>
<td>{a}</td>
<td>{a, b, c, d, e}</td>
<td>{a, b, d, e}</td>
<td>{a, b, d, e}</td>
</tr>
<tr>
<td>2</td>
<td>{a}</td>
<td>{a, b}</td>
<td>{a, b}</td>
<td>{a, b, d, e}</td>
<td>{a, b, d, e}</td>
</tr>
<tr>
<td>3</td>
<td>{a, b}</td>
<td>{a, b}</td>
<td>{a, b}</td>
<td>{a, b, d, e}</td>
<td>{a, b, d, e}</td>
</tr>
</tbody>
</table>

**Note** We have \(\{a,b\} \subseteq X\) and \((A \setminus \{a,b,d,e\}) \cap X = (\{c\} \cap X) = \emptyset\) for every stable model \(X\) of \(P\).
The simplistic expand algorithm

- $\text{expand}_P$
  - tightens the approximation on stable models
  - is stable model preserving
Let's expand with \( d \)!

\[
P = \begin{cases} 
    a \leftarrow \\
    b \leftarrow a, \sim c \\
    d \leftarrow b, \sim e \\
    e \leftarrow \sim d 
\end{cases}
\]

<table>
<thead>
<tr>
<th>( L' )</th>
<th>( Cn(P^{U'}) )</th>
<th>( L )</th>
<th>( U' )</th>
<th>( Cn(P^{L'}) )</th>
<th>( U )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 {d}</td>
<td>{a}</td>
<td>{a, d}</td>
<td>{a, b, c, d, e}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
</tr>
<tr>
<td>2 {a, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
</tr>
<tr>
<td>3 {a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
<td>{a, b, d}</td>
</tr>
</tbody>
</table>

- Note \( \{a, b, d\} \) is a stable model of \( P \).
Let's expand with $d$!

$$P = \begin{cases} 
    a \leftarrow \\
    b \leftarrow a, \sim c \\
    d \leftarrow b, \sim e \\
    e \leftarrow \sim d
\end{cases}$$

<table>
<thead>
<tr>
<th>$L'$</th>
<th>$Cn(P^{U'})$</th>
<th>$L$</th>
<th>$U'$</th>
<th>$Cn(P^{L'})$</th>
<th>$U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${d}$</td>
<td>${a}$</td>
<td>${a, d}$</td>
<td>${a, b, c, d, e}$</td>
<td>${a, b, d}$</td>
</tr>
<tr>
<td>2</td>
<td>${a, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
</tr>
<tr>
<td>3</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
</tr>
</tbody>
</table>

Note $\{a, b, d\}$ is a stable model of $P$.
Let’s expand with $d$!

\[ P = \begin{cases} 
  a \leftarrow \\
  b \leftarrow a, \sim c \\
  d \leftarrow b, \sim e \\
  e \leftarrow \sim d
\end{cases} \]

<table>
<thead>
<tr>
<th></th>
<th>$Cn(P^{U'})$</th>
<th></th>
<th></th>
<th>$Cn(P^{L'})$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${d}$</td>
<td>${a}$</td>
<td>${a, d}$</td>
<td>${a, b, c, d, e}$</td>
<td>${a, b, d}$</td>
</tr>
<tr>
<td>2</td>
<td>${a, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
</tr>
<tr>
<td>3</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
<td>${a, b, d}$</td>
</tr>
</tbody>
</table>

Note $\{a, b, d\}$ is a stable model of $P$
Computation from first principles

Let’s expand with $\sim d$!

$$P = \left\{ \begin{array}{l}
a \leftarrow \\
b \leftarrow a, \sim c \\
d \leftarrow b, \sim e \\
e \leftarrow \sim d \\
\end{array} \right\}$$

<table>
<thead>
<tr>
<th>$L'$</th>
<th>$Cn(P^{U'})$</th>
<th>$L$</th>
<th>$U'$</th>
<th>$Cn(P^{L'})$</th>
<th>$U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\emptyset$</td>
<td>${a, e}$</td>
<td>${a, e}$</td>
<td>${a, b, c, e}$</td>
<td>${a, b, d, e}$</td>
</tr>
<tr>
<td>2</td>
<td>${a, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
</tr>
<tr>
<td>3</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
</tr>
</tbody>
</table>

Note $\{a, b, e\}$ is a stable model of $P$.
Let’s expand with $\sim d$!

$$P = \begin{cases} 
  a \leftarrow \\
  b \leftarrow a, \sim c \\
  d \leftarrow b, \sim e \\
  e \leftarrow \sim d 
\end{cases}$$

<table>
<thead>
<tr>
<th></th>
<th>$L'$</th>
<th>$Cn(P^{U'})$</th>
<th>$L$</th>
<th>$U'$</th>
<th>$Cn(P^{L'})$</th>
<th>$U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\emptyset$</td>
<td>${a, e}$</td>
<td>${a, e}$</td>
<td>${a, b, c, e}$</td>
<td>${a, b, d, e}$</td>
<td>${a, b, e}$</td>
</tr>
<tr>
<td>2</td>
<td>${a, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
</tr>
<tr>
<td>3</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
</tr>
</tbody>
</table>

Note $\{a, b, e\}$ is a stable model of $P$. 

Sebastian Rudolph (TUD) Answer Set Programming: Computation & Characterization 12 / 43
Computation from first principles

Let’s expand with $\sim d$!

$$P = \begin{cases} 
  a \leftarrow \\
  b \leftarrow a, \sim c \\
  d \leftarrow b, \sim e \\
  e \leftarrow \sim d 
\end{cases}$$

<table>
<thead>
<tr>
<th>$L'$</th>
<th>$Cn(P^{U'})$</th>
<th>$L$</th>
<th>$U'$</th>
<th>$Cn(P^{L'})$</th>
<th>$U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\emptyset$</td>
<td>${a, e}$</td>
<td>${a, e}$</td>
<td>${a, b, c, e}$</td>
<td>${a, b, d, e}$</td>
</tr>
<tr>
<td>2</td>
<td>${a, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
</tr>
<tr>
<td>3</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
<td>${a, b, e}$</td>
</tr>
</tbody>
</table>

Note $\{a, b, e\}$ is a stable model of $P$.
A simplistic solving algorithm

\[
\text{solve}_P(L, U) \\
\quad (L, U) \leftarrow \text{expand}_P(L, U) \quad \text{// propagation} \\
\quad \text{if } L \not\subseteq U \text{ then failure} \quad \text{// failure} \\
\quad \text{if } L = U \text{ then output } L \quad \text{// success} \\
\quad \text{else choose } a \in U \setminus L \quad \text{// choice} \\
\quad \quad \text{solve}_P(L \cup \{a\}, U) \\
\quad \quad \text{solve}_P(L, U \setminus \{a\})
\]
A simplistic solving algorithm

- Close to the approach taken by the ASP solver smodels, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a three-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (expand)
    - making one choice at a time by appeal to a heuristic (choose)
  - Heuristic choices are made on atoms
A simplistic solving algorithm

- Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a three-valued interpretation
    - The search space is pruned by
      - deriving deterministic consequences and detecting conflicts (expand)
      - making one choice at a time by appeal to a heuristic (choose)
  - Heuristic choices are made on atoms
A simplistic solving algorithm

- Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a three-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (**expand**)
    - making one choice at a time by appeal to a heuristic (**choose**)

- Heuristic choices are made on atoms
A simplistic solving algorithm

- Close to the approach taken by the ASP solver smodels, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - Backtracking search building a binary search tree
  - A node in the search tree corresponds to a three-valued interpretation
  - The search space is pruned by
    - deriving deterministic consequences and detecting conflicts (expand)
    - making one choice at a time by appeal to a heuristic (choose)

- Heuristic choices are made on atoms
Outline

1. Consequence operator
2. Computation from first principles
3. Complexity
4. Completion
5. Tightness
6. Loops and Loop Formulas
Let $a$ be an atom and $X$ be a set of atoms

- For a positive normal logic program $P$:
  - Deciding whether $X$ is the stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in the stable model of $P$ is $P$-complete

- For a normal logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP$-complete

- For a normal logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is $co-NP$-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is $\Delta^p_2$-complete
Let $a$ be an atom and $X$ be a set of atoms

- For a positive normal logic program $P$:
  - Deciding whether $X$ is the stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in the stable model of $P$ is $P$-complete

- For a normal logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP$-complete

- For a normal logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is $co-NP$-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is $\Delta^P_2$-complete
Let $a$ be an atom and $X$ be a set of atoms

- For a positive normal logic program $P$:
  - Deciding whether $X$ is the stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in the stable model of $P$ is $P$-complete

- For a normal logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP$-complete

- For a normal logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is $co-NP$-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is $\Delta_2^p$-complete
Let $a$ be an atom and $X$ be a set of atoms

- For a positive normal logic program $P$:
  - Deciding whether $X$ is the stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in the stable model of $P$ is $P$-complete

- For a normal logic program $P$:
  - Deciding whether $X$ is a stable model of $P$ is $P$-complete
  - Deciding whether $a$ is in a stable model of $P$ is $NP$-complete

- For a normal logic program $P$ with optimization statements:
  - Deciding whether $X$ is an optimal stable model of $P$ is $co-NP$-complete
  - Deciding whether $a$ is in an optimal stable model of $P$ is $\Delta_2^p$-complete
Outline

1. Consequence operator
2. Computation from first principles
3. Complexity
4. Completion
5. Tightness
6. Loops and Loop Formulas
Motivation

- **Question** Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

- **Observation** Although each atom is defined through a set of rules, each such rule provides only a sufficient condition for its head atom.

- **Idea** The idea of program completion is to turn such implications into a definition by adding the corresponding necessary counterpart.
Motivation

- **Question** Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

- **Observation** Although each atom is defined through a set of rules, each such rule provides only a *sufficient* condition for its head atom.

- **Idea** The idea of program completion is to turn such implications into a definition by adding the corresponding *necessary* counterpart.
**Question** Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

**Observation** Although each atom is defined through a set of rules, each such rule provides only a sufficient condition for its head atom.

**Idea** The idea of program completion is to turn such implications into a definition by adding the corresponding necessary counterpart.
Let $P$ be a normal logic program

The completion $CF(P)$ of $P$ is defined as follows

$$CF(P) = \left\{ a \leftrightarrow \bigvee_{r \in P, \text{head}(r) = a} BF(\text{body}(r)) \mid a \in \text{atom}(P) \right\}$$

where

$$BF(\text{body}(r)) = \bigwedge_{a \in \text{body}(r)} a \land \bigwedge_{a \in \text{body}(r)} \neg a$$
An example

\[ P = \begin{cases} 
  a & \leftarrow \\
  b & \leftarrow \sim a \\
  c & \leftarrow a, \sim d \\
  d & \leftarrow \sim c, \sim e \\
  e & \leftarrow b, \sim f \\
  e & \leftarrow e 
\end{cases} \]

\[ CF(P) = \begin{cases} 
  a & \leftrightarrow \top \\
  b & \leftrightarrow \neg a \\
  c & \leftrightarrow a \land \neg d \\
  d & \leftrightarrow \neg c \land \neg e \\
  e & \leftrightarrow (b \land \neg f) \lor e \\
  f & \leftrightarrow \bot 
\end{cases} \]
An example

\[ P = \{ \begin{array}{l}
a \leftarrow \\
b \leftarrow \sim a \\
c \leftarrow a, \sim d \\
d \leftarrow \sim c, \sim e \\
e \leftarrow b, \sim f \\
e \leftarrow e \\
\end{array} \} \]

\[ CF(P) = \{ \begin{array}{l}
a \leftrightarrow \top \\
b \leftrightarrow \neg a \\
c \leftrightarrow a \land \neg d \\
d \leftrightarrow \neg c \land \neg e \\
e \leftrightarrow (b \land \neg f) \lor e \\
f \leftrightarrow \bot \\
\end{array} \} \]
A closer look

- $CF(P)$ is logically equivalent to $\leftarrow CF(P) \cup \rightarrow CF(P)$, where

$$\leftarrow CF(P) = \{ a \leftarrow \bigvee_{B \in body_P(a)} BF(B) \mid a \in \text{atom}(P) \}$$

$$\rightarrow CF(P) = \{ a \rightarrow \bigvee_{B \in body_P(a)} BF(B) \mid a \in \text{atom}(P) \}$$

$body_P(a) = \{ body(r) \mid r \in P \text{ and } head(r) = a \}$

- $\leftarrow CF(P)$ characterizes the classical models of $P$
- $\rightarrow CF(P)$ completes $P$ by adding necessary conditions for all atoms
A closer look

- $\leftarrow CF(P)$ is logically equivalent to $\leftarrow CF(P) \cup \rightarrow CF(P)$, where

\[
\begin{align*}
\leftarrow CF(P) &= \{ a \leftarrow \bigvee_{B \in body_P(a)} BF(B) \mid a \in \text{atom}(P) \} \\
\rightarrow CF(P) &= \{ a \rightarrow \bigvee_{B \in body_P(a)} BF(B) \mid a \in \text{atom}(P) \}
\end{align*}
\]

\[
\text{body}_P(a) = \{ \text{body}(r) \mid r \in P \text{ and } \text{head}(r) = a \}
\]

- $\leftarrow CF(P)$ characterizes the classical models of $P$
- $\rightarrow CF(P)$ completes $P$ by adding necessary conditions for all atoms
A closer look

\[ P = \{ \begin{align*}
    a & \leftarrow \\
    b & \leftarrow \sim a \\
    c & \leftarrow a, \sim d \\
    d & \leftarrow \sim c, \sim e \\
    e & \leftarrow b, \sim f \\
    e & \leftarrow e
\end{align*} \} \]
A closer look

\[ P = \begin{cases} 
  a \leftarrow \\
  b \leftarrow \neg a \\
  c \leftarrow a, \neg d \\
  d \leftarrow \neg c, \neg e \\
  e \leftarrow b, \neg f \\
  e \leftarrow e 
\end{cases} \]

\[ \overleftarrow{\text{CF}(P)} = \begin{cases} 
  a \leftarrow \top \\
  b \leftarrow \neg a \\
  c \leftarrow a \land \neg d \\
  d \leftarrow \neg c \land \neg e \\
  e \leftarrow (b \land \neg f) \lor e \\
  f \leftarrow \bot 
\end{cases} \]
Completion

A closer look

\[
\Rightarrow \text{CF}(P) = \left\{ \begin{array}{l}
a \leftarrow \top \\
b \leftarrow \neg a \\
c \leftarrow a \land \neg d \\
d \leftarrow \neg c \land \neg e \\
e \leftarrow (b \land \neg f) \lor e \\
f \leftarrow \bot \\
\end{array} \right. 
\]
A closer look

\[
\overleftarrow{\text{CF}}(P) = \begin{cases} 
  a & \leftarrow \top \\
  b & \leftarrow \neg a \\
  c & \leftarrow a \land \neg d \\
  d & \leftarrow \neg c \land \neg e \\
  e & \leftarrow (b \land \neg f) \lor e \\
  f & \leftarrow \bot 
\end{cases}
\]

\[
\overrightarrow{\text{CF}}(P) = \begin{cases} 
  a & \rightarrow \top \\
  b & \rightarrow \neg a \\
  c & \rightarrow a \land \neg d \\
  d & \rightarrow \neg c \land \neg e \\
  e & \rightarrow (b \land \neg f) \lor e \\
  f & \rightarrow \bot 
\end{cases}
\]

\[
\overrightarrow{\text{CF}}(P) = \overleftarrow{\text{CF}}(P)
\]
A closer look

$$\bar{CF}(P) = \begin{cases} 
    a \leftarrow \top \\
    b \leftarrow \neg a \\
    c \leftarrow a \land \neg d \\
    d \leftarrow \neg c \land \neg e \\
    e \leftarrow (b \land \neg f) \lor e \\
    f \leftarrow \bot 
\end{cases} \quad \frac{\longrightarrow}{\quad} \quad \begin{cases} 
    a \rightarrow \top \\
    b \rightarrow \neg a \\
    c \rightarrow a \land \neg d \\
    d \rightarrow \neg c \land \neg e \\
    e \rightarrow (b \land \neg f) \lor e \\
    f \rightarrow \bot 
\end{cases} \quad = \bar{CF}(P)$$

$$CF(P) = \begin{cases} 
    a \leftrightarrow \top \\
    b \leftrightarrow \neg a \\
    c \leftrightarrow a \land \neg d \\
    d \leftrightarrow \neg c \land \neg e \\
    e \leftrightarrow (b \land \neg f) \lor e \\
    f \leftrightarrow \bot 
\end{cases}$$
A closer look

\[
\begin{align*}
\leftarrow \text{CF}(P) &= \left\{ \\
&\begin{align*}
a &\leftarrow \top \\
b &\leftarrow \neg a \\
c &\leftarrow a \land \neg d \\
d &\leftarrow \neg c \land \neg e \\
e &\leftarrow (b \land \neg f) \lor e \\
f &\leftarrow \bot
\end{align*} \\
\rightarrow \text{CF}(P) &= \left\{ \\
&\begin{align*}
a &\rightarrow \top \\
b &\rightarrow \neg a \\
c &\rightarrow a \land \neg d \\
d &\rightarrow \neg c \land \neg e \\
e &\rightarrow (b \land \neg f) \lor e \\
f &\rightarrow \bot
\end{align*}
\end{align*}
\]

\[= \text{CF}(P) \cup \text{CF}(P)\]
Supported models

- Every stable model of $P$ is a model of $CF(P)$, but not vice versa.
- Models of $CF(P)$ are called the supported models of $P$.
- In other words, every stable model of $P$ is a supported model of $P$.
- By definition, every supported model of $P$ is also a model of $P$. 
Supported models

- Every stable model of $P$ is a model of $CF(P)$, but not vice versa
- Models of $CF(P)$ are called the supported models of $P$
- In other words, every stable model of $P$ is a supported model of $P$
- By definition, every supported model of $P$ is also a model of $P$
Supported models

- Every stable model of $P$ is a model of $CF(P)$, but not vice versa.
- Models of $CF(P)$ are called the supported models of $P$.

In other words, every stable model of $P$ is a supported model of $P$.
By definition, every supported model of $P$ is also a model of $P$. 
Supported models

- Every stable model of $P$ is a model of $CF(P)$, but not vice versa.
- Models of $CF(P)$ are called the supported models of $P$.
- In other words, every stable model of $P$ is a supported model of $P$.
- By definition, every supported model of $P$ is also a model of $P$. 
An example

\[ P = \left\{ \begin{array}{l}
a \leftarrow \\
b \leftarrow \sim a \\
c \leftarrow a, \sim d \\
d \leftarrow \sim c, \sim e \\
e \leftarrow e \\
e \leftarrow b, \sim f 
\end{array} \right\} \]

- \( P \) has 21 models, including \{a, c\}, \{a, d\}, but also \{a, b, c, d, e, f\}
- \( P \) has 3 supported models, namely \{a, c\}, \{a, d\}, and \{a, c, e\}
- \( P \) has 2 stable models, namely \{a, c\} and \{a, d\}
An example

\[ P = \left\{ \begin{array}{lll}
    a & \leftarrow & a, \sim d \\
    b & \leftarrow & \sim a \\
    c & \leftarrow & a, \sim d \\
    d & \leftarrow & \sim c, \sim e \\
    e & \leftarrow & b, \sim f \\
    e & \leftarrow & e 
\end{array} \right\} 
\]

- \( P \) has 21 models, including \( \{a, c\}, \{a, d\} \), but also \( \{a, b, c, d, e, f\} \)
- \( P \) has 3 supported models, namely \( \{a, c\}, \{a, d\} \), and \( \{a, c, e\} \)
- \( P \) has 2 stable models, namely \( \{a, c\} \) and \( \{a, d\} \)
An example

\[ P = \left\{ \begin{array}{llll}
  a & \leftarrow & c & \leftarrow & a, \sim d \\
  b & \leftarrow & \sim a & d & \leftarrow & \sim c, \sim e
\end{array} \right. \]

- \( P \) has 21 models, including \( \{a, c\}, \{a, d\} \), but also \( \{a, b, c, d, e, f\} \)
- \( P \) has 3 supported models, namely \( \{a, c\}, \{a, d\}, \) and \( \{a, c, e\} \)
- \( P \) has 2 stable models, namely \( \{a, c\} \) and \( \{a, d\} \)
An example

\[
P = \left\{ \begin{array}{ccc} 
a & \leftarrow & c \\
b & \leftarrow & \sim a \\
c & \leftarrow & a, \sim d \\
d & \leftarrow & \sim c, \sim e \\
e & \leftarrow & b, \sim f \\
e & \leftarrow & e \\
\end{array} \right\}
\]

- \(P\) has 21 models, including \(\{a, c\}\), \(\{a, d\}\), but also \(\{a, b, c, d, e, f\}\)
- \(P\) has 3 supported models, namely \(\{a, c\}\), \(\{a, d\}\), and \(\{a, c, e\}\)
- \(P\) has 2 stable models, namely \(\{a, c\}\) and \(\{a, d\}\)
Outline

1. Consequence operator
2. Computation from first principles
3. Complexity
4. Completion
5. Tightness
6. Loops and Loop Formulas
The mismatch

**Question** What causes the mismatch between supported models and stable models?

**Hint** Consider the unstable yet supported model \( \{ a, c, e \} \) of \( P \).

**Answer** Cyclic derivations are causing the mismatch between supported and stable models.

- Atoms in a stable model can be "derived" from a program in a finite number of steps.
- Atoms in a cycle (not being "supported from outside the cycle") cannot be "derived" from a program in a finite number of steps.

Note: But such atoms do not contradict the completion of a program and do thus not eliminate an unstable supported model.
The mismatch

- **Question** What causes the mismatch between supported models and stable models?
- **Hint** Consider the unstable yet supported model \{a, c, e\} of P!

- **Answer** Cyclic derivations are causing the mismatch between supported and stable models.
  - Atoms in a stable model can be "derived" from a program in a finite number of steps.
  - Atoms in a cycle (not being "supported from outside the cycle") cannot be "derived" from a program in a finite number of steps.
  - Note: But such atoms do not contradict the completion of a program and do thus not eliminate an unstable supported model.
Question What causes the mismatch between supported models and stable models?

Hint Consider the unstable yet supported model \( \{a, c, e\} \) of \( P \) !

Answer Cyclic derivations are causing the mismatch between supported and stable models

- Atoms in a stable model can be “derived” from a program in a finite number of steps
- Atoms in a cycle (not being “supported from outside the cycle”) cannot be “derived” from a program in a finite number of steps

Note But such atoms do not contradict the completion of a program and do thus not eliminate an unstable supported model
The mismatch

- **Question** What causes the mismatch between supported models and stable models?

- **Hint** Consider the unstable yet supported model \{a, c, e\} of \(P\)!

- **Answer** Cyclic derivations are causing the mismatch between supported and stable models
  
  - Atoms in a stable model can be “derived” from a program in a finite number of steps
  
  - Atoms in a cycle (not being “supported from outside the cycle”) cannot be “derived” from a program in a finite number of steps

  **Note** But such atoms do not contradict the completion of a program and do thus not eliminate an unstable supported model
The mismatch

**Question** What causes the mismatch between supported models and stable models?

**Hint** Consider the unstable yet supported model \{a, c, e\} of $P$.

**Answer** Cyclic derivations are causing the mismatch between supported and stable models.

- Atoms in a stable model can be "derived" from a program in a finite number of steps.
- Atoms in a cycle (not being "supported from outside the cycle") cannot be "derived" from a program in a finite number of steps.

*Note* But such atoms do not contradict the completion of a program and do thus not eliminate an unstable supported model.
The mismatch

- **Question** What causes the mismatch between supported models and stable models?

- **Hint** Consider the unstable yet supported model \{a, c, e\} of \(P\)!

- **Answer** Cyclic derivations are causing the mismatch between supported and stable models
  - Atoms in a stable model can be “derived” from a program in a finite number of steps
  - Atoms in a cycle (not being “supported from outside the cycle”) cannot be “derived” from a program in a finite number of steps

**Note** But such atoms do not contradict the completion of a program and do thus not eliminate an unstable supported model.
Non-cyclic derivations

Let $X$ be a stable model of normal logic program $P$

- For every atom $A \in X$, there is a finite sequence of positive rules
  $$\langle r_1, \ldots, r_n \rangle$$
  such that
  1. $\text{head}(r_1) = A$
  2. $\text{body}(r_i)^+ \subseteq \{\text{head}(r_j) \mid i < j \leq n\}$ for $1 \leq i \leq n$
  3. $r_i \in P^X$ for $1 \leq i \leq n$

That is, each atom of $X$ has a non-cyclic derivation from $P^X$

- Example There is no finite sequence of rules providing a derivation for $e$ from $P\{a,c,e\}$
Non-cyclic derivations

Let $X$ be a stable model of normal logic program $P$

- For every atom $A \in X$, there is a finite sequence of positive rules
  
  \[ \langle r_1, \ldots, r_n \rangle \]

  such that
  
  1. $\text{head}(r_1) = A$
  2. $\text{body}(r_i)^+ \subseteq \{ \text{head}(r_j) \mid i < j \leq n \}$ for $1 \leq i \leq n$
  3. $r_i \in P^X$ for $1 \leq i \leq n$

- That is, each atom of $X$ has a non-cyclic derivation from $P^X$

- Example: There is no finite sequence of rules providing a derivation for $e$ from $P\{a,c,e\}$
Non-cyclic derivations

Let $X$ be a stable model of normal logic program $P$

- For every atom $A \in X$, there is a finite sequence of positive rules

$$\langle r_1, \ldots, r_n \rangle$$

such that

1. $\text{head}(r_1) = A$
2. $\text{body}(r_i)^+ \subseteq \{ \text{head}(r_j) \mid i < j \leq n \}$ for $1 \leq i \leq n$
3. $r_i \in P^X$ for $1 \leq i \leq n$

- That is, each atom of $X$ has a non-cyclic derivation from $P^X$

- Example There is no finite sequence of rules providing a derivation for $e$ from $P\{a,c,e\}$
Positive atom dependency graph

The origin of (potential) circular derivations can be read off the positive atom dependency graph $G(P)$ of a logic program $P$ given by

$$(\text{atom}(P), \{(a, b) \mid r \in P, a \in \text{body}(r)^+, \text{head}(r) = b\})$$

A logic program $P$ is called tight, if $G(P)$ is acyclic.
The origin of (potential) circular derivations can be read off the positive atom dependency graph $G(P)$ of a logic program $P$ given by

$$(\text{atom}(P), \{(a, b) \mid r \in P, a \in \text{body}(r)^+, \text{head}(r) = b\})$$

A logic program $P$ is called **tight**, if $G(P)$ is acyclic.
Example

\[ P = \left\{ \begin{array}{l}
a \leftarrow \\
b \leftarrow \sim a \\
c \leftarrow a, \sim d \\
d \leftarrow \sim c, \sim e \\
e \leftarrow b, \sim f \\
e \leftarrow e \\
e \leftarrow e \\
\end{array} \right\} \]

\[ G(P) = (\{a, b, c, d, e\}, \{(a, c), (b, e), (e, e)\}) \]

- \( P \) has supported models: \( \{a, c\} \), \( \{a, d\} \), and \( \{a, c, e\} \)
- \( P \) has stable models: \( \{a, c\} \) and \( \{a, d\} \)
Example

\[ P = \begin{cases} 
    a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\
    b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e 
\end{cases} \]

\[ G(P) = (\{a, b, c, d, e\}, \{(a, c), (b, e), (e, e)\}) \]

- \( P \) has supported models: \( \{a, c\} \), \( \{a, d\} \), and \( \{a, c, e\} \)
- \( P \) has stable models: \( \{a, c\} \) and \( \{a, d\} \)
Example

- \( P = \{ a \leftarrow c \leftarrow a, \sim d \leftarrow d \leftarrow \sim c, \sim e \leftarrow e \leftarrow e \} \)

- \( G(P) = (\{a, b, c, d, e\}, \{(a, c), (b, e), (e, e)\}) \)

- \( P \) has supported models: \( \{a, c\}, \{a, d\}, \text{ and } \{a, c, e\} \)

- \( P \) has stable models: \( \{a, c\} \text{ and } \{a, d\} \)
Example

\[ P = \{ \begin{align*}
  a &\leftarrow & c &\leftarrow & a, &\sim & d \\
  b &\leftarrow & \sim & a & d &\leftarrow & \sim & c, &\sim & e
\end{align*}\} \]

\[ G(P) = (\{a, b, c, d, e\}, \{(a, c), (b, e), (e, e)\}) \]

\[ \begin{array}{ccc}
  a &\rightarrow & c \\
  b &\rightarrow & e \\
  & & d \\
  & & f
\end{array} \]

- \( P \) has supported models: \( \{a, c\}, \{a, d\}, \) and \( \{a, c, e\} \)
- \( P \) has stable models: \( \{a, c\} \) and \( \{a, d\} \)
A logic program $P$ is called **tight**, if $G(P)$ is acyclic.

For tight programs, stable and supported models coincide.

**Fages’ Theorem**
Let $P$ be a tight normal logic program and $X \subseteq \text{atom}(P)$.
Then, $X$ is a stable model of $P$ iff $X \models CF(P)$. 
A logic program $P$ is called **tight**, if $G(P)$ is acyclic.

For tight programs, stable and supported models coincide:

Fages’ Theorem
Let $P$ be a tight normal logic program and $X \subseteq \text{atom}(P)$.
Then, $X$ is a stable model of $P$ iff $X \models CF(P)$. 
Tight programs

- A logic program $P$ is called tight, if $G(P)$ is acyclic.
- For tight programs, stable and supported models coincide.

Fages’ Theorem

Let $P$ be a tight normal logic program and $X \subseteq \text{atom}(P)$.
Then, $X$ is a stable model of $P$ iff $X \models CF(P)$. 
Another example

\[ P = \left\{ \begin{array}{l}
a \leftarrow \sim b \\
b \leftarrow \sim a \\
c \leftarrow a, b \\
d \leftarrow a \\
e \leftarrow \sim a, \sim b \\
\end{array} \right. \]

\[ G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\}) \]

- \( P \) has supported models: \( \{a, c, d\}, \{b\}, \) and \( \{b, c, d\} \)
- \( P \) has stable models: \( \{a, c, d\} \) and \( \{b\} \)
Another example

\[ P = \begin{cases} 
  a \leftarrow \sim b \\
  c \leftarrow a, b \\
  d \leftarrow a \\
  e \leftarrow \sim a, \sim b \\
  b \leftarrow \sim a \\
  c \leftarrow d \\
  d \leftarrow b, c 
\end{cases} \]

\[ G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\}) \]

- \( P \) has supported models: \( \{a, c, d\} \), \( \{b\} \), and \( \{b, c, d\} \)
- \( P \) has stable models: \( \{a, c, d\} \) and \( \{b\} \)
Another example

\[ P = \left\{ \begin{array}{l}
  a \leftarrow \sim b \\
  c \leftarrow a, b \\
  d \leftarrow a \\
  e \leftarrow \sim a, \sim b \\
  b \leftarrow \sim a \\
  c \leftarrow d \\
  d \leftarrow b, c
\end{array} \right. \]

\[ G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\}) \]

- \( P \) has supported models: \( \{a, c, d\}, \{b\}, \) and \( \{b, c, d\} \)
- \( P \) has stable models: \( \{a, c, d\} \) and \( \{b\} \)
Another example

\[ P = \{ \begin{align*} a & \leftarrow \neg b \\
 b & \leftarrow \neg a \\
 c & \leftarrow a, b \\
 d & \leftarrow a \\
 e & \leftarrow \neg a, \neg b \\
 c & \leftarrow d \\
 d & \leftarrow b, c \end{align*} \} \]

\[ G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\}) \]

\[ P \text{ has supported models: } \{a, c, d\}, \{b\}, \text{ and } \{b, c, d\} \]

\[ P \text{ has stable models: } \{a, c, d\} \text{ and } \{b\} \]
Outline

1. Consequence operator
2. Computation from first principles
3. Complexity
4. Completion
5. Tightness
6. Loops and Loop Formulas
Motivation

Question Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

Observation Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program.

Idea Add formulas prohibiting circular support of sets of atoms.

Note Circular support between atoms $a$ and $b$ is possible, if $a$ has a path to $b$ and $b$ has a path to $a$ in the program’s positive atom dependency graph.
Motivation

- **Question** Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

- **Observation** Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program.

- **Idea** Add formulas prohibiting circular support of sets of atoms.

- **Note** Circular support between atoms $a$ and $b$ is possible, if $a$ has a path to $b$ and $b$ has a path to $a$ in the program’s positive atom dependency graph.
Motivation

**Question** Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

**Observation** Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program.

**Idea** Add formulas prohibiting circular support of sets of atoms.

**Note** Circular support between atoms $a$ and $b$ is possible, if $a$ has a path to $b$ and $b$ has a path to $a$ in the program’s positive atom dependency graph.
Motivation

- **Question** Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of $P$?

- **Observation** Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program.

- **Idea** Add formulas prohibiting circular support of sets of atoms.

- **Note** Circular support between atoms $a$ and $b$ is possible, if $a$ has a path to $b$ and $b$ has a path to $a$ in the program’s positive atom dependency graph.
Loops

Let $P$ be a normal logic program, and let $G(P) = (\text{atom}(P), E)$ be the positive atom dependency graph of $P$.

- A set $\emptyset \subset L \subseteq \text{atom}(P)$ is a loop of $P$ if it induces a non-trivial strongly connected subgraph of $G(P)$. That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$.

- We denote the set of all loops of $P$ by $\text{loop}(P)$.

- Note A program $P$ is tight iff $\text{loop}(P) = \emptyset$. 

Loops and Loop Formulas
Loops

Let $P$ be a normal logic program, and let $G(P) = (\text{atom}(P), E)$ be the positive atom dependency graph of $P$

- A set $\emptyset \subset L \subseteq \text{atom}(P)$ is a loop of $P$ if it induces a non-trivial strongly connected subgraph of $G(P)$
  
  That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$

- We denote the set of all loops of $P$ by $\text{loop}(P)$
- Note A program $P$ is tight iff $\text{loop}(P) = \emptyset$
Loops and Loop Formulas

Loops

Let $P$ be a normal logic program, and let $G(P) = (\text{atom}(P), E)$ be the positive atom dependency graph of $P$

- A set $\emptyset \subset L \subseteq \text{atom}(P)$ is a loop of $P$ if it induces a non-trivial strongly connected subgraph of $G(P)$.
  That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$.

- We denote the set of all loops of $P$ by $\text{loop}(P)$.

- Note A program $P$ is tight iff $\text{loop}(P) = \emptyset$. 
Loops

Let $P$ be a normal logic program, and let $G(P) = (\text{atom}(P), E)$ be the positive atom dependency graph of $P$.

- A set $\emptyset \subset L \subseteq \text{atom}(P)$ is a loop of $P$ if it induces a non-trivial strongly connected subgraph of $G(P)$.
  That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$.

- We denote the set of all loops of $P$ by $\text{loop}(P)$.

- Note: A program $P$ is tight iff $\text{loop}(P) = \emptyset$. 
Loops

Let $P$ be a normal logic program, and let $G(P) = (\text{atom}(P), E)$ be the positive atom dependency graph of $P$.

- A set $\emptyset \subset L \subseteq \text{atom}(P)$ is a loop of $P$ if it induces a non-trivial strongly connected subgraph of $G(P)$. That is, each pair of atoms in $L$ is connected by a path of non-zero length in $(L, E \cap (L \times L))$.

- We denote the set of all loops of $P$ by $\text{loop}(P)$.

- Note A program $P$ is tight iff $\text{loop}(P) = \emptyset$. 

Loops and Loop Formulas

Sebastian Rudolph (TUD) Answer Set Programming: Computation & Characterization 34 / 43
Example

\[ P = \begin{cases} 
  a \leftarrow c \\
  c \leftarrow a, \neg d \\
  e \leftarrow b, \neg f \\
  b \leftarrow \neg a \\
  d \leftarrow \neg c, \neg e \\
  e \leftarrow e 
\end{cases} \]

\[
\text{loop}(P) = \{\{e\}\} 
\]
Example

\[ P = \{ a \leftarrow c \leftarrow a, \sim d \leftarrow \sim b, \sim f \} \]

\[ \text{loop}(P) = \{ \{ e \} \} \]
Another example

\[
P = \left\{ \begin{array}{l}
    a \leftarrow \neg b \\
    c \leftarrow a, b \\
    d \leftarrow a \\
    e \leftarrow \neg a, \neg b \\
    b \leftarrow \neg a \\
    c \leftarrow d \\
    d \leftarrow b, c \\
\end{array} \right. \\
\]

\[
\text{loop}(P) = \{ \{c, d\} \}
\]
Another example

\[ P = \begin{cases} a &\leftarrow \sim b \\ c &\leftarrow a, b \\ d &\leftarrow a \\ e &\leftarrow \sim a, \sim b \\ b &\leftarrow \sim a \\ c &\leftarrow d \\ d &\leftarrow b, c \end{cases} \]

\[ \text{loop}(P) = \{\{c, d\}\} \]
Loops and Loop Formulas

Yet another example

\[ P = \begin{cases} a \leftarrow \neg b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, \neg a \\ b \leftarrow \neg a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{cases} \]

\[ \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \} \]
Yet another example

\[ P = \left\{ \begin{array}{l}
 a \leftarrow \neg b \\
 c \leftarrow a \\
 d \leftarrow b, c \\
 e \leftarrow b, \neg a \\
 b \leftarrow \neg a \\
 c \leftarrow b, d \\
 d \leftarrow e \\
 e \leftarrow c, d
\end{array} \right\} \]

\[ \text{loop}(P) = \left\{ \{c, d\}, \{d, e\}, \{c, d, e\} \right\} \]
Yet another example

\[ P = \left\{ \begin{array}{l}
  a \leftarrow \lnot b \\
  c \leftarrow a \\
  d \leftarrow b, c \\
  e \leftarrow b, \lnot a \\
  b \leftarrow \lnot a \\
  c \leftarrow b, d \\
  d \leftarrow e \\
  e \leftarrow c, d 
\end{array} \right\} \]

\[ \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \} \]
Loops and Loop Formulas

Let $P$ be a normal logic program

- For $L \subseteq \text{atom}(P)$, define the external supports of $L$ for $P$ as
  $$ES_P(L) = \{ r \in P \mid \text{head}(r) \in L \text{ and } \text{body}(r)^+ \cap L = \emptyset \}$$

- Define the external bodies of $L$ in $P$ as $EB_P(L) = \text{body}(ES_P(L))$

- The (disjunctive) loop formula of $L$ for $P$ is
  $$LF_P(L) = (\bigvee_{a \in L} a) \rightarrow (\bigvee_{B \in EB_P(L)} BF(B))$$
  $$\equiv (\bigwedge_{B \in EB_P(L)} \neg BF(B)) \rightarrow (\bigwedge_{a \in L} \neg a)$$

- Note: The loop formula of $L$ enforces all atoms in $L$ to be false whenever $L$ is not externally supported

- Define $LF(P) = \{ LF_P(L) \mid L \in \text{loop}(P) \}$
Loops and Loop Formulas

Loop formulas

Let $P$ be a normal logic program

- For $L \subseteq \text{atom}(P)$, define the external supports of $L$ for $P$ as

$$ES_P(L) = \{ r \in P \mid \text{head}(r) \in L \text{ and } \text{body}(r)^+ \cap L = \emptyset \}$$

- Define the external bodies of $L$ in $P$ as $EB_P(L) = \text{body}(ES_P(L))$

- The (disjunctive) loop formula of $L$ for $P$ is

$$LF_P(L) = (\bigvee_{a \in L} a) \rightarrow (\bigvee_{B \in EB_P(L)} BF(B))$$

$$\equiv (\bigwedge_{B \in EB_P(L)} \neg BF(B)) \rightarrow (\bigwedge_{a \in L} \neg a)$$

- Note: The loop formula of $L$ enforces all atoms in $L$ to be false whenever $L$ is not externally supported

- Define $LF(P) = \{ LF_P(L) \mid L \in \text{loop}(P) \}$
Loops and Loop Formulas

Loop formulas

Let $P$ be a normal logic program

- For $L \subseteq \text{atom}(P)$, define the external supports of $L$ for $P$ as
  \[
  \text{ES}_P(L) = \{ r \in P \mid \text{head}(r) \in L \text{ and } \text{body}(r)^+ \cap L = \emptyset \}\]

- Define the external bodies of $L$ in $P$ as $\text{EB}_P(L) = \text{body}(\text{ES}_P(L))$

- The (disjunctive) loop formula of $L$ for $P$ is
  \[
  \text{LF}_P(L) = (\bigvee a \in L a) \rightarrow (\bigvee B \in \text{EB}_P(L) \text{BF}(B))
  \equiv (\bigwedge B \in \text{EB}_P(L) \neg \text{BF}(B)) \rightarrow (\bigwedge a \in L \neg a)
  \]

Note The loop formula of $L$ enforces all atoms in $L$ to be false whenever $L$ is not externally supported.

- Define $\text{LF}(P) = \{ \text{LF}_P(L) \mid L \in \text{loop}(P) \}$

Let $P$ be a normal logic program

- For $L \subseteq \text{atom}(P)$, define the external supports of $L$ for $P$ as

  $$ES_P(L) = \{ r \in P \mid \text{head}(r) \in L \text{ and } \text{body}(r)^+ \cap L = \emptyset \}$$

- Define the external bodies of $L$ in $P$ as $EB_P(L) = \text{body}(ES_P(L))$

- The (disjunctive) loop formula of $L$ for $P$ is

  $$LF_P(L) = (\bigvee_{a \in L} a) \rightarrow (\bigvee_{B \in EB_P(L)} BF(B))$$
  $$\equiv (\bigwedge_{B \in EB_P(L)} \neg BF(B)) \rightarrow (\bigwedge_{a \in L} \neg a)$$

- **Note** The loop formula of $L$ enforces all atoms in $L$ to be false whenever $L$ is not externally supported

- Define $LF(P) = \{ LF_P(L) \mid L \in \text{loop}(P) \}$
Example

\[ P = \left\{ \begin{array}{lll}
  a & \leftarrow & c \\
  b & \leftarrow & \neg a \\
  c & \leftarrow & a, \neg d \\
  d & \leftarrow & \neg c, \neg e \\
  e & \leftarrow & b, \neg f
\end{array} \right\} \]

\[ \text{loop}(P) = \{ \{e\} \} \]
\[ \text{LF}(P) = \{ e \rightarrow b \land \neg f \} \]
Loops and Loop Formulas

Example

\[ P = \left\{ \begin{array}{l}
  a \leftarrow c \\
  b \leftarrow \neg a \\
  c \leftarrow a, \neg d \\
  d \leftarrow \neg c, \neg e \\
  e \leftarrow b, \neg f \\
  e \leftarrow e
\end{array} \right\} \]

\[ \text{loop}(P) = \{ \{ e \} \} \]

\[ LF(P) = \{ e \rightarrow b \land \neg f \} \]
Another example

\[
\begin{align*}
P = \left\{ & \begin{array}{llll}
a \leftarrow \sim b & c \leftarrow a, b & d \leftarrow a & e \leftarrow \sim a, \sim b \\
b \leftarrow \sim a & c \leftarrow d & d \leftarrow b, c
\end{array} \right\}
\end{align*}
\]

- \(\text{loop}(P) = \{\{c, d\}\}\)
- \(\text{LF}(P) = \{c \lor d \rightarrow (a \land b) \lor a\}\)
Another example

\[ P = \{ \begin{align*} a & \leftarrow \neg b \quad c \leftarrow a, b \quad d \leftarrow a \quad e \leftarrow \neg a, \neg b \\ b & \leftarrow \neg a \quad c \leftarrow d \quad d \leftarrow b, c \end{align*} \} \]

- \( \text{loop}(P) = \{ \{c, d\} \} \)
- \( \text{LF}(P) = \{ c \lor d \rightarrow (a \land b) \lor a \} \)
Yet another example

\[ P = \left\{ \begin{array}{l}
  a \leftarrow \sim b \\
  c \leftarrow a \\
  d \leftarrow b, c \\
  e \leftarrow b, \sim a \\
  b \leftarrow \sim a \\
  c \leftarrow b, d \\
  d \leftarrow e \\
  e \leftarrow c, d 
\end{array} \right\} \]

\[ \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\}\} \]

\[ LF(P) = \left\{ \begin{array}{l}
  c \lor d \rightarrow a \lor e \\
  d \lor e \rightarrow (b \land c) \lor (b \land \neg a) \\
  c \lor d \lor e \rightarrow a \lor (b \land \neg a) 
\end{array} \right\} \]
Yet another example

\[ P = \left\{ \begin{array}{l}
  a \leftarrow \neg b \\
  c \leftarrow a \\
  d \leftarrow b, c \\
  e \leftarrow b, \neg a \\
  b \leftarrow \neg a \\
  c \leftarrow b, d \\
  d \leftarrow e \\
  e \leftarrow c, d
\end{array} \right\} \]

\[ \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \} \]

\[ \text{LF}(P) = \left\{ \begin{array}{l}
  c \lor d \rightarrow a \lor e \\
  d \lor e \rightarrow (b \land c) \lor (b \land \neg a) \\
  c \lor d \lor e \rightarrow a \lor (b \land \neg a)
\end{array} \right\} \]
Yet another example

\[ P = \begin{cases} 
  a \leftarrow \sim b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, \sim a \\
  b \leftarrow \sim a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d
\end{cases} \]

\[ \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \} \]

\[ \text{LF}(P) = \begin{cases} 
  c \lor d \rightarrow a \lor e \\
  d \lor e \rightarrow (b \land c) \lor (b \land \sim a) \\
  c \lor d \lor e \rightarrow a \lor (b \land \sim a)
\end{cases} \]
Yet another example

\[ P = \{ \begin{array}{c}
    a \leftarrow \neg b \\
    c \leftarrow a \\
    d \leftarrow b, c \\
    e \leftarrow b, \neg a \\
    b \leftarrow \neg a \\
    c \leftarrow b, d \\
    d \leftarrow e \\
    e \leftarrow c, d
\end{array} \} \]

\[ \text{loop}(P) = \{ \{c, d\}, \{d, e\}, \{c, d, e\} \} \]

\[ \text{LF}(P) = \{ \begin{array}{c}
    c \lor d \rightarrow a \lor e \\
    d \lor e \rightarrow (b \land c) \lor (b \land \neg a) \\
    c \lor d \lor e \rightarrow a \lor (b \land \neg a)
\end{array} \} \]
Yet another example

\[ P = \begin{cases} a \leftarrow \neg b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, \neg a \\ b \leftarrow \neg a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{cases} \]

\[ \text{loop}(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\} \]

\[ \text{LF}(P) = \begin{cases} c \lor d \rightarrow a \lor e \\ d \lor e \rightarrow (b \land c) \lor (b \land \neg a) \\ c \lor d \lor e \rightarrow a \lor (b \land \neg a) \end{cases} \]
Lin-Zhao Theorem

Theorem

Let $P$ be a normal logic program and $X \subseteq \text{atom}(P)$.
Then, $X$ is a stable model of $P$ iff $X \models CF(P) \cup LF(P)$.
Loops and loop formulas: Properties

Let $X$ be a supported model of normal logic program $P$

- Then, $X$ is a stable model of $P$ iff
  - $X \models \{ LF_P(U) \mid U \subseteq \text{atom}(P) \}$;
  - $X \models \{ LF_P(U) \mid U \subseteq X \}$;
  - $X \models \{ LF_P(L) \mid L \in \text{loop}(P) \}$, that is, $X \models LF(P)$;
  - $X \models \{ LF_P(L) \mid L \in \text{loop}(P) \text{ and } L \subseteq X \}$

- Note If $X$ is not a stable model of $P$, then there is a loop $L \subseteq X \setminus \text{Cn}(P^X)$ such that $X \not\models LF_P(L)$
Loops and loop formulas: Properties

Let $X$ be a supported model of normal logic program $P$

- Then, $X$ is a stable model of $P$ iff
  - $X \models \{LF_P(U) \mid U \subseteq \text{atom}(P)\}$;
  - $X \models \{LF_P(U) \mid U \subseteq X\}$;
  - $X \models \{LF_P(L) \mid L \in \text{loop}(P)\}$, that is, $X \models LF(P)$;
  - $X \models \{LF_P(L) \mid L \in \text{loop}(P) \text{ and } L \subseteq X\}$

- Note If $X$ is not a stable model of $P$, then there is a loop $L \subseteq X \setminus \text{Cn}(P^X)$ such that $X \not\models LF_P(L)$
Loops and loop formulas: Properties

Let $X$ be a supported model of normal logic program $P$

- Then, $X$ is a stable model of $P$ iff
  - $X \models \{LF_P(U) \mid U \subseteq atom(P)\}$;
  - $X \models \{LF_P(U) \mid U \subseteq X\}$;
  - $X \models \{LF_P(L) \mid L \in loop(P)\}$, that is, $X \models LF(P)$;
  - $X \models \{LF_P(L) \mid L \in loop(P) \text{ and } L \subseteq X\}$

- Note If $X$ is not a stable model of $P$, then there is a loop $L \subseteq X \setminus Cn(P^X)$ such that $X \not\models LF_P(L)$