Answer Set Programming: Computation & Characterization

Sebastian Rudolph

Computational Logic Group
Technische Universität Dresden

Slides based on a lecture by Martin Gebser and Torsten Schaub.

Outline

Consequence operator

- Let P be a positive program and X a set of atoms
  - The consequence operator T_P is defined as follows:

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

  - 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

- Let P be a positive program and X a set of atoms
  - The consequence operator T_P is defined as follows:

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

  - 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

- Let P be a positive program and X a set of atoms
  - The consequence operator T_P is defined as follows:

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

  - 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

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

$$\begin{array}{lclclcl}
T_P^0\emptyset & = & \emptyset & & & & \\
T_P^1\emptyset & = & \{p,q\} & = & T_P T_P^0\emptyset & = & T_P\emptyset \\
T_P^2\emptyset & = & \{p,q,r\} & = & T_P T_P^1\emptyset & = & T_P\{p,q\} \\
T_P^3\emptyset & = & \{p,q,r,t\} & = & T_P T_P^2\emptyset & = & T_P\{p,q,r\} \\
T_P^4\emptyset & = & \{p,q,r,t,s\} & = & T_P T_P^3\emptyset & = & T_P\{p,q,r,t\} \\
T_P^5\emptyset & = & \{p,q,r,t,s\} & = & T_P T_P^4\emptyset & = & T_P\{p,q,r,t,s\} \\
T_P^6\emptyset & = & \{p,q,r,t,s\} & = & T_P T_P^5\emptyset & = & T_P\{p,q,r,t,s\}
\end{array}$$

- $Cn(P) = \{p,q,r,t,s\}$ 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 $X \subset \{p,q,r,t,s\}$

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

$$\begin{array}{lclclcl}
T_P^0\emptyset & = & \emptyset & & & & \\
T_P^1\emptyset & = & \{p, q\} & = & T_P T_P^0\emptyset & = & T_P\emptyset \\
T_P^2\emptyset & = & \{p, q, r\} & = & T_P T_P^1\emptyset & = & T_P\{p, q\} \\
T_P^3\emptyset & = & \{p, q, r, t\} & = & T_P T_P^2\emptyset & = & T_P\{p, q, r\} \\
T_P^4\emptyset & = & \{p, q, r, t, s\} & = & T_P T_P^3\emptyset & = & T_P\{p, q, r, t\} \\
T_P^5\emptyset & = & \{p, q, r, t, s\} & = & T_P T_P^4\emptyset & = & T_P\{p, q, r, t, s\} \\
T_P^6\emptyset & = & \{p, q, r, t, s\} & = & T_P T_P^5\emptyset & = & T_P\{p, q, r, t, s\}
\end{array}$$

- $Cn(P) = \{p, q, r, t, s\}$ 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 $X \subset \{p, q, r, t, s\}$

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

$$\begin{array}{lllllll}
T_P^0\emptyset & = & \emptyset & & & & \\
T_P^1\emptyset & = & \{p, q\} & = & T_P T_P^0\emptyset & = & T_P\emptyset \\
T_P^2\emptyset & = & \{p, q, r\} & = & T_P T_P^1\emptyset & = & T_P\{p, q\} \\
T_P^3\emptyset & = & \{p, q, r, t\} & = & T_P T_P^2\emptyset & = & T_P\{p, q, r\} \\
T_P^4\emptyset & = & \{p, q, r, t, s\} & = & T_P T_P^3\emptyset & = & T_P\{p, q, r, t\} \\
T_P^5\emptyset & = & \{p, q, r, t, s\} & = & T_P T_P^4\emptyset & = & T_P\{p, q, r, t, s\} \\
T_P^6\emptyset & = & \{p, q, r, t, s\} & = & T_P T_P^5\emptyset & = & T_P\{p, q, r, t, s\}
\end{array}$$

- $Cn(P) = \{p, q, r, t, s\}$ 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

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 $L \subseteq X$,

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 $L \subseteq X$,

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

- 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

- 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

- 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

- 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

- 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

- 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

  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

  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

  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

  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

The simplistic expand algorithm

$\textbf{expand}_P(L, U)$

repeat

$L' \leftarrow L$
$U' \leftarrow U$

$L \leftarrow L' \cup Cn(P^{U'})$
$U \leftarrow U' \cap Cn(P^{L'})$

if $L \not\subseteq U$ **then return**

until $L = L'$ and $U = U'$

An example

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

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\emptyset$	$\{a\}$	$\{a\}$	$\{a, b, c, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$
2	$\{a\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$
3	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$

- Note We have $\{a, b\} \subseteq X$ and $(\mathcal{A} \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset$ for every stable model X of P

An example

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

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\emptyset$	$\{a\}$	$\{a\}$	$\{a, b, c, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$
2	$\{a\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$
3	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$

- Note We have $\{a, b\} \subseteq X$ and $(\mathcal{A} \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset$ for every stable model X of P

An example

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

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\emptyset$	$\{a\}$	$\{a\}$	$\{a, b, c, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$
2	$\{a\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$
3	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$

- Note We have $\{a, b\} \subseteq X$ and $(\mathcal{A} \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset$ for every stable model X of P

The simplistic expand algorithm

- **expand**$_P$
  - tightens the approximation on stable models
  - is stable model preserving

Let's expand with d !

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

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\{d\}$	$\{a\}$	$\{a, d\}$	$\{a, b, c, d, e\}$	$\{a, b, d\}$	$\{a, b, d\}$
2	$\{a, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$
3	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$

- Note $\{a, b, d\}$ is a stable model of P

Let's expand with d !

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

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\{d\}$	$\{a\}$	$\{a, d\}$	$\{a, b, c, d, e\}$	$\{a, b, d\}$	$\{a, b, d\}$
2	$\{a, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$
3	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$

- Note $\{a, b, d\}$ is a stable model of P

Let's expand with d !

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

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\{d\}$	$\{a\}$	$\{a, d\}$	$\{a, b, c, d, e\}$	$\{a, b, d\}$	$\{a, b, d\}$
2	$\{a, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$
3	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$

- Note $\{a, b, d\}$ is a stable model of P

Let's expand with $\sim d$!

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

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\emptyset$	$\{a, e\}$	$\{a, e\}$	$\{a, b, c, e\}$	$\{a, b, d, e\}$	$\{a, b, e\}$
2	$\{a, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$
3	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$

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

Let's expand with $\sim d$!

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

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\emptyset$	$\{a, e\}$	$\{a, e\}$	$\{a, b, c, e\}$	$\{a, b, d, e\}$	$\{a, b, e\}$
2	$\{a, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$
3	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$

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

Let's expand with $\sim d$!

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

	L'	$Cn(P^{U'})$	L	U'	$Cn(P^{L'})$	U
1	$\emptyset$	$\{a, e\}$	$\{a, e\}$	$\{a, b, c, e\}$	$\{a, b, d, e\}$	$\{a, b, e\}$
2	$\{a, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$
3	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$

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

A simplistic solving algorithm

$solve_P(L, U)$

- $(L, U) \leftarrow expand_P(L, U)$ — *// propagation*
- **if** $L \not\subseteq U$ **then failure** — *// failure*
- **if** $L = U$ **then output** L — *// success*
- **else choose** $a \in U \setminus L$ — *// choice*
  - $solve_P(L \cup \{a\}, U)$
  - $solve_P(L, U \setminus \{a\})$

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

Complexity

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 $co\text{-}NP$-complete
  - Deciding whether a is in an optimal stable model of P is Δ_2^P-complete

Complexity

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 $co\text{-}NP$-complete
  - Deciding whether a is in an optimal stable model of P is Δ_2^P-complete

Complexity

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 $co\text{-}NP$-complete
  - Deciding whether a is in an optimal stable model of P is Δ_2^P-complete

Complexity

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 $co\text{-}NP$-complete
  - Deciding whether a is in an optimal stable model of P is Δ_2^p-complete

Outline

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

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

Program completion

Let P be a normal logic program

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

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

where

$$BF(body(r)) = \bigwedge_{a \in body(r)^+} a \wedge \bigwedge_{a \in body(r)^-} \neg a$$

An example

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow \sim a \\ c \leftarrow a, \sim d \\ d \leftarrow \sim c, \sim e \\ e \leftarrow b, \sim f \\ e \leftarrow e \end{array} \right\} \qquad CF(P) = \left\{ \begin{array}{l} a \leftrightarrow \top \\ b \leftrightarrow \neg a \\ c \leftrightarrow a \wedge \neg d \\ d \leftrightarrow \neg c \wedge \neg e \\ e \leftrightarrow (b \wedge \neg f) \vee e \\ f \leftrightarrow \bot \end{array} \right\}$$

An example

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow {\sim}a \\ c \leftarrow a, {\sim}d \\ d \leftarrow {\sim}c, {\sim}e \\ e \leftarrow b, {\sim}f \\ e \leftarrow e \end{array} \right\} \qquad CF(P) = \left\{ \begin{array}{l} a \leftrightarrow \top \\ b \leftrightarrow \neg a \\ c \leftrightarrow a \wedge \neg d \\ d \leftrightarrow \neg c \wedge \neg e \\ e \leftrightarrow (b \wedge \neg f) \vee e \\ f \leftrightarrow \bot \end{array} \right\}$$

A closer look

- $CF(P)$ is logically equivalent to $\overleftarrow{CF}(P) \cup \overrightarrow{CF}(P)$, where

$$\begin{array}{rcl} \overleftarrow{CF}(P) & = & \left\{ a \leftarrow \bigvee_{B \in body_P(a)} BF(B) \mid a \in atom(P) \right\} \\ \overrightarrow{CF}(P) & = & \left\{ a \rightarrow \bigvee_{B \in body_P(a)} BF(B) \mid a \in atom(P) \right\} \end{array}$$

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

- $\overleftarrow{CF}(P)$ characterizes the classical models of P
- $\overrightarrow{CF}(P)$ completes P by adding necessary conditions for all atoms

A closer look

- $CF(P)$ is logically equivalent to $\overleftarrow{CF}(P) \cup \overrightarrow{CF}(P)$, where

$$
\begin{array}{rcl}
\overleftarrow{CF}(P) & = & \left\{ a \leftarrow \bigvee_{B \in body_P(a)} BF(B) \mid a \in atom(P) \right\} \\
\overrightarrow{CF}(P) & = & \left\{ a \rightarrow \bigvee_{B \in body_P(a)} BF(B) \mid a \in atom(P) \right\}
\end{array}
$$

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

- $\overleftarrow{CF}(P)$ characterizes the classical models of P
- $\overrightarrow{CF}(P)$ completes P by adding necessary conditions for all atoms

A closer look

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

A closer look

$$P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow {\sim}a \\ c \leftarrow a, {\sim}d \\ d \leftarrow {\sim}c, {\sim}e \\ e \leftarrow b, {\sim}f \\ e \leftarrow e \end{array} \right\} \qquad \overleftarrow{CF}(P) = \left\{ \begin{array}{l} a \leftarrow \top \\ b \leftarrow \neg a \\ c \leftarrow a \wedge \neg d \\ d \leftarrow \neg c \wedge \neg e \\ e \leftarrow (b \wedge \neg f) \vee e \\ f \leftarrow \bot \end{array} \right\}$$

A closer look

$$\overleftarrow{CF}(P) = \left\{ \begin{array}{l} a \leftarrow \top \\ b \leftarrow \neg a \\ c \leftarrow a \wedge \neg d \\ d \leftarrow \neg c \wedge \neg e \\ e \leftarrow (b \wedge \neg f) \vee e \\ f \leftarrow \bot \end{array} \right\}$$

A closer look

$$\overleftarrow{CF}(P) = \left\{ \begin{array}{l} a \leftarrow \top \\ b \leftarrow \neg a \\ c \leftarrow a \wedge \neg d \\ d \leftarrow \neg c \wedge \neg e \\ e \leftarrow (b \wedge \neg f) \vee e \\ f \leftarrow \bot \end{array} \right\} \left\{ \begin{array}{l} a \rightarrow \top \\ b \rightarrow \neg a \\ c \rightarrow a \wedge \neg d \\ d \rightarrow \neg c \wedge \neg e \\ e \rightarrow (b \wedge \neg f) \vee e \\ f \rightarrow \bot \end{array} \right\} = \overrightarrow{CF}(P)$$

A closer look

$$\overleftarrow{CF}(P) = \left\{ \begin{array}{l} a \leftarrow \top \\ b \leftarrow \neg a \\ c \leftarrow a \wedge \neg d \\ d \leftarrow \neg c \wedge \neg e \\ e \leftarrow (b \wedge \neg f) \vee e \\ f \leftarrow \bot \end{array} \right\} \left\{ \begin{array}{l} a \rightarrow \top \\ b \rightarrow \neg a \\ c \rightarrow a \wedge \neg d \\ d \rightarrow \neg c \wedge \neg e \\ e \rightarrow (b \wedge \neg f) \vee e \\ f \rightarrow \bot \end{array} \right\} = \overrightarrow{CF}(P)$$

$$CF(P) = \left\{ \begin{array}{l} a \leftrightarrow \top \\ b \leftrightarrow \neg a \\ c \leftrightarrow a \wedge \neg d \\ d \leftrightarrow \neg c \wedge \neg e \\ e \leftrightarrow (b \wedge \neg f) \vee e \\ f \leftrightarrow \bot \end{array} \right\}$$

A closer look

$$\overleftarrow{CF}(P) = \left\{ \begin{array}{l} a \leftarrow \top \\ b \leftarrow \neg a \\ c \leftarrow a \wedge \neg d \\ d \leftarrow \neg c \wedge \neg e \\ e \leftarrow (b \wedge \neg f) \vee e \\ f \leftarrow \bot \end{array} \right\} \left\{ \begin{array}{l} a \rightarrow \top \\ b \rightarrow \neg a \\ c \rightarrow a \wedge \neg d \\ d \rightarrow \neg c \wedge \neg e \\ e \rightarrow (b \wedge \neg f) \vee e \\ f \rightarrow \bot \end{array} \right\} = \overrightarrow{CF}(P)$$

$$CF(P) = \left\{ \begin{array}{l} a \leftrightarrow \top \\ b \leftrightarrow \neg a \\ c \leftrightarrow a \wedge \neg d \\ d \leftrightarrow \neg c \wedge \neg e \\ e \leftrightarrow (b \wedge \neg f) \vee e \\ f \leftrightarrow \bot \end{array} \right\} \equiv \overleftarrow{CF}(P) \cup \overrightarrow{CF}(P)$$

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

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

- 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

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

- 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

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

- 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

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

- 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

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

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

  $$\langle r_1, \dots, r_n \rangle$$

  such that

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

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

- A logic program P is called tight, if $G(P)$ is acyclic

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

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

- A logic program P is called tight, if $G(P)$ is acyclic

Example

- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e\}, \{(a, c), (b, e), (e, e)\})$

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

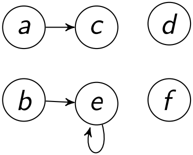
Example

- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e\}, \{(a, c), (b, e), (e, e)\})$

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

Example

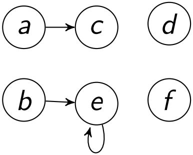
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e\}, \{(a, c), (b, e), (e, e)\})$



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

Example

- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e\}, \{(a, c), (b, e), (e, e)\})$



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

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 $X \subseteq atom(P)$.
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 $X \subseteq atom(P)$
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 $X \subseteq atom(P)$
Then, X is a stable model of P iff $X \models CF(P)$

Another example

- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a, b & d \leftarrow a & e \leftarrow \sim a, \sim b \\ b \leftarrow \sim a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e\}, \{(a,c), (a,d), (b,c), (b,d), (c,d), (d,c)\})$

- P has supported models: $\{a, c, d\}$, $\{b\}$, and $\{b, c, d\}$
- P has stable models: $\{a, c, d\}$ and $\{b\}$

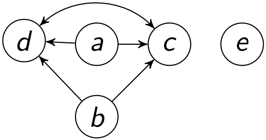
Another example

- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a, b & d \leftarrow a & e \leftarrow {\sim}a, {\sim}b \\ b \leftarrow {\sim}a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\})$

- P has supported models: $\{a, c, d\}$, $\{b\}$, and $\{b, c, d\}$
- P has stable models: $\{a, c, d\}$ and $\{b\}$

Another example

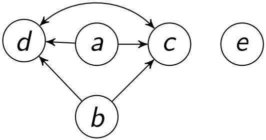
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a, b & d \leftarrow a & e \leftarrow {\sim}a, {\sim}b \\ b \leftarrow {\sim}a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\})$



- P has supported models: $\{a, c, d\}$, $\{b\}$, and $\{b, c, d\}$
- P has stable models: $\{a, c, d\}$ and $\{b\}$

Another example

- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a, b & d \leftarrow a & e \leftarrow \sim a, \sim b \\ b \leftarrow \sim a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\})$



- P has supported models: $\{a, c, d\}$, $\{b\}$, and $\{b, c, d\}$
- P has stable models: $\{a, c, d\}$ and $\{b\}$

Outline

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 $G(P) = (atom(P), E)$ be the positive atom dependency graph of P

- A set $\emptyset \subset L \subseteq atom(P)$ 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 $loop(P)$
- Note A program P is tight iff $loop(P) = \emptyset$

Loops

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

- A set $\emptyset \subset L \subseteq atom(P)$ 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 $loop(P)$
- Note A program P is tight iff $loop(P) = \emptyset$

Loops

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

- A set $\emptyset \subset L \subseteq atom(P)$ 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 $loop(P)$

- Note A program P is tight iff $loop(P) = \emptyset$

Loops

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

- A set $\emptyset \subset L \subseteq atom(P)$ 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 $loop(P)$

- Note A program P is tight iff $loop(P) = \emptyset$

Loops

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

- A set $\emptyset \subset L \subseteq atom(P)$ is a loop of P
  if it induces a non-trivial strongly connected subgraph of $G(P)$

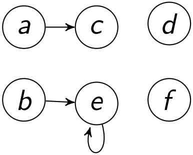
  That is, each pair of atoms in L is connected by a path of non-zero length in $(L, E \cap (L \times L))$

- We denote the set of all loops of P by $loop(P)$

- Note A program P is tight iff $loop(P) = \emptyset$

Example

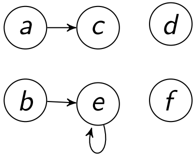
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$



- $loop(P) = \{\{e\}\}$

Example

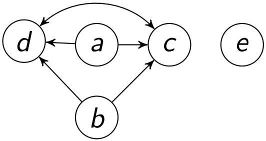
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$



- $loop(P) = \{\{e\}\}$

Another example

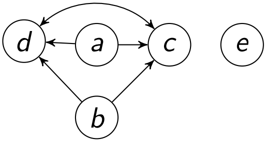
- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a, b & d \leftarrow a & e \leftarrow \sim a, \sim b \\ b \leftarrow \sim a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$



- $loop(P) = \{\{c, d\}\}$

Another example

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



- $loop(P) = \{\{c, d\}\}$

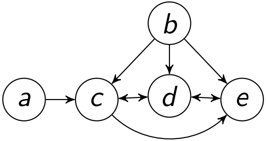
Yet another example

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

- $loop(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$

Yet another example

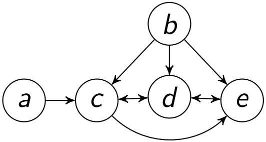
- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, \sim a \\ b \leftarrow \sim a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



- $loop(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$

Yet another example

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



- $loop(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$

Loop formulas

Let P be a normal logic program

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

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

- Define the external bodies of L in P as $EB_P(L) = body(ES_P(L))$
- The (disjunctive) loop formula of L for P is

$$\begin{array}{rcl} LF_P(L) & = & (\bigvee_{a\in L} a) \rightarrow (\bigvee_{B\in EB_P(L)} BF(B)) \\ & \equiv & (\bigwedge_{B\in EB_P(L)} \neg BF(B)) \rightarrow (\bigwedge_{a\in L} \neg a) \end{array}$$

- Note The loop formula of L enforces all atoms in L to be *false* whenever L is not externally supported
- Define $LF(P) = \{LF_P(L) \mid L \in loop(P)\}$

Loop formulas

Let P be a normal logic program

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

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

- Define the external bodies of L in P as $EB_P(L) = body(ES_P(L))$
- The (disjunctive) loop formula of L for P is

$$\begin{aligned} LF_P(L) &= \left(\bigvee_{a \in L} a\right) \rightarrow \left(\bigvee_{B \in EB_P(L)} BF(B)\right) \\ &\equiv \left(\bigwedge_{B \in EB_P(L)} \neg BF(B)\right) \rightarrow \left(\bigwedge_{a \in L} \neg a\right) \end{aligned}$$

- Note The loop formula of L enforces all atoms in L to be *false* whenever L is not externally supported
- Define $LF(P) = \{LF_P(L) \mid L \in loop(P)\}$

Loop formulas

Let P be a normal logic program

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

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

- Define the external bodies of L in P as $EB_P(L) = body(ES_P(L))$
- The (disjunctive) loop formula of L for P is

$$\begin{array}{rcl} LF_P(L) & = & (\bigvee_{a \in L} a) \rightarrow (\bigvee_{B \in EB_P(L)} BF(B)) \\ & \equiv & (\bigwedge_{B \in EB_P(L)} \neg BF(B)) \rightarrow (\bigwedge_{a \in L} \neg a) \end{array}$$

- Note The loop formula of L enforces all atoms in L to be *false* whenever L is not externally supported
- Define $LF(P) = \{LF_P(L) \mid L \in loop(P)\}$

Loop formulas

Let P be a normal logic program

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

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

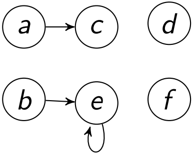
- Define the external bodies of L in P as $EB_P(L) = body(ES_P(L))$
- The (disjunctive) loop formula of L for P is

$$\begin{aligned} LF_P(L) &= \left(\bigvee_{a \in L} a\right) \rightarrow \left(\bigvee_{B \in EB_P(L)} BF(B)\right) \\ &\equiv \left(\bigwedge_{B \in EB_P(L)} \neg BF(B)\right) \rightarrow \left(\bigwedge_{a \in L} \neg a\right) \end{aligned}$$

- Note The loop formula of L enforces all atoms in L to be *false* whenever L is not externally supported
- Define $LF(P) = \{LF_P(L) \mid L \in loop(P)\}$

Example

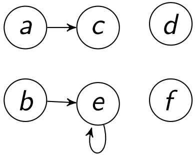
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$



- $loop(P) = \{\{e\}\}$
- $LF(P) = \{e \rightarrow b \land \neg f\}$

Example

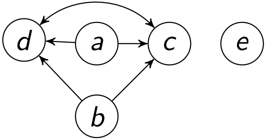
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$



- $loop(P) = \{\{e\}\}$
- $LF(P) = \{e \rightarrow b \wedge \neg f\}$

Another example

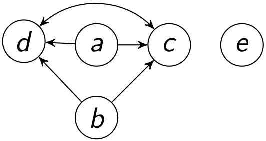
- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a, b & d \leftarrow a & e \leftarrow \sim a, \sim b \\ b \leftarrow \sim a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$



- $loop(P) = \{\{c, d\}\}$
- $LF(P) = \{c \vee d \rightarrow (a \wedge b) \vee a\}$

Another example

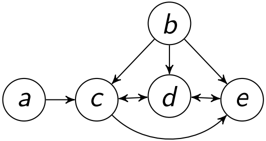
- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a, b & d \leftarrow a & e \leftarrow \sim a, \sim b \\ b \leftarrow \sim a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$



- $loop(P) = \{\{c, d\}\}$
- $LF(P) = \{c \vee d \rightarrow (a \wedge b) \vee a\}$

Yet another example

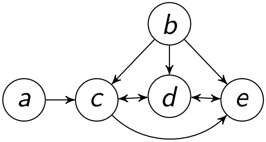
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, {\sim}a \\ b \leftarrow {\sim}a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



- $loop(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$
- $LF(P) = \left\{ \begin{array}{l} c \vee d \rightarrow a \vee e \\ d \vee e \rightarrow (b \wedge c) \vee (b \wedge \neg a) \\ c \vee d \vee e \rightarrow a \vee (b \wedge \neg a) \end{array} \right\}$

Yet another example

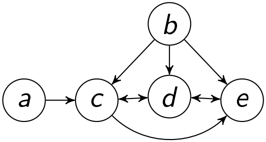
- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, \sim a \\ b \leftarrow \sim a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



- $loop(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$
- $LF(P) = \left\{ \begin{array}{l} c \vee d \rightarrow a \vee e \\ d \vee e \rightarrow (b \wedge c) \vee (b \wedge \neg a) \\ c \vee d \vee e \rightarrow a \vee (b \wedge \neg a) \end{array} \right\}$

Yet another example

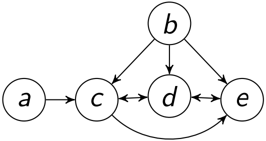
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, {\sim}a \\ b \leftarrow {\sim}a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



- $loop(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$
- $LF(P) = \left\{ \begin{array}{l} c \vee d \rightarrow a \vee e \\ d \vee e \rightarrow (b \wedge c) \vee (b \wedge \neg a) \\ c \vee d \vee e \rightarrow a \vee (b \wedge \neg a) \end{array} \right\}$

Yet another example

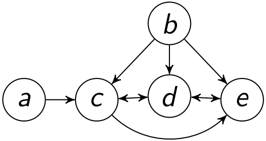
- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, \sim a \\ b \leftarrow \sim a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



- $loop(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$
- $LF(P) = \left\{ \begin{array}{l} c \vee d \rightarrow a \vee e \\ d \vee e \rightarrow (b \wedge c) \vee (b \wedge \neg a) \\ c \vee d \vee e \rightarrow a \vee (b \wedge \neg a) \end{array} \right\}$

Yet another example

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



- $loop(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$
- $LF(P) = \left\{ \begin{array}{l} c \vee d \rightarrow a \vee e \\ d \vee e \rightarrow (b \wedge c) \vee (b \wedge \neg a) \\ c \vee d \vee e \rightarrow a \vee (b \wedge \neg a) \end{array} \right\}$

Lin-Zhao Theorem

Theorem

Let P be a normal logic program and $X \subseteq atom(P)$
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
  - $X \models \{LF_P(U) \mid U \subseteq atom(P)\}$;
  - $X \models \{LF_P(U) \mid U \subseteq X\}$;
  - $X \models \{LF_P(L) \mid L \in loop(P)\}$, that is, $X \models LF(P)$;
  - $X \models \{LF_P(L) \mid L \in loop(P)$ 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 $X \not\models LF_P(L)$

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 $X \not\models LF_P(L)$

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 $X \not\models LF_P(L)$