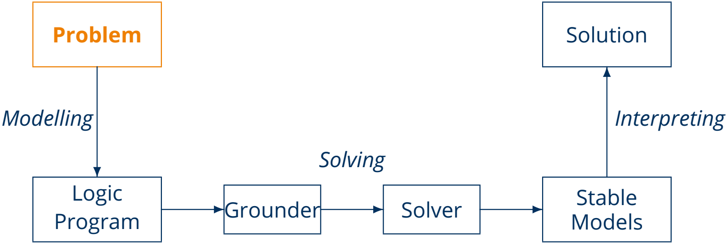
Hannes Strass
Faculty of Computer Science, Institute of Artificial Intelligence, Computational Logic Group

ASP: Computation and Characterisation

Lecture 12, 23rd Jan 2023 // Foundations of Logic Programming, WS 2022/23

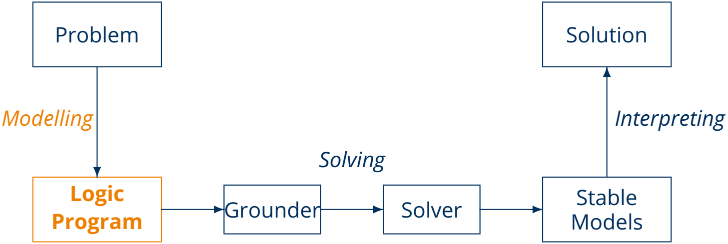
Previously ...

- The language of normal logic programs can be extended by constructs:
  - **Integrity constraints** for eliminating unwanted solution candidates
  - **Choice rules** for choosing subsets of atoms
  - **Cardinality rules** for counting certain present/absent atoms
- All of them can be translated back into normal logic program rules.
- The modelling methodology of ASP is **generate and test**:
  - Generate solution candidates, eliminate infeasible ones.



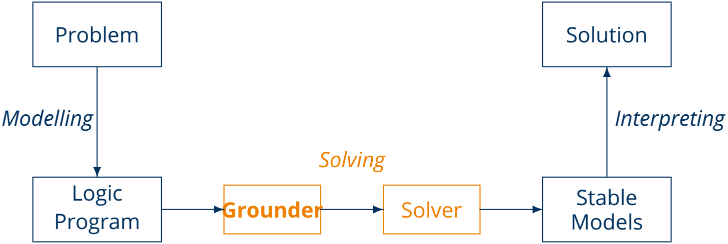
Previously . . .

- The language of normal logic programs can be extended by constructs:
  - **Integrity constraints** for eliminating unwanted solution candidates
  - **Choice rules** for choosing subsets of atoms
  - **Cardinality rules** for counting certain present/absent atoms
- All of them can be translated back into normal logic program rules.
- The modelling methodology of ASP is **generate and test**:
  - Generate solution candidates, eliminate infeasible ones.



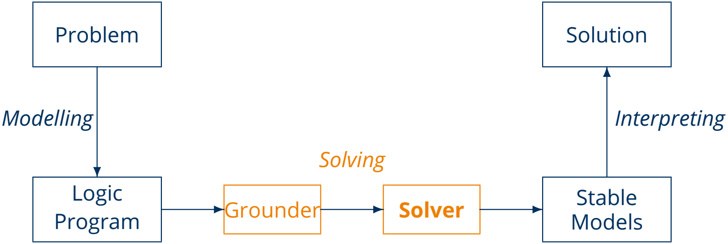
Previously . . .

- The language of normal logic programs can be extended by constructs:
  - **Integrity constraints** for eliminating unwanted solution candidates
  - **Choice rules** for choosing subsets of atoms
  - **Cardinality rules** for counting certain present/absent atoms
- All of them can be translated back into normal logic program rules.
- The modelling methodology of ASP is **generate and test**:
  - Generate solution candidates, eliminate infeasible ones.



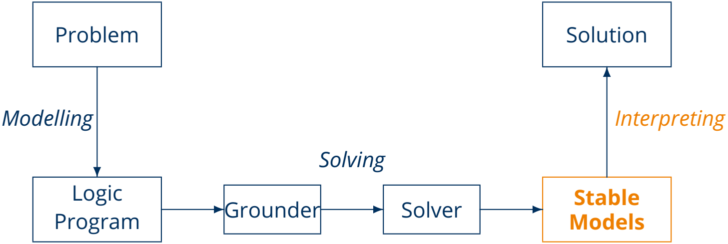
Previously . . .

- The language of normal logic programs can be extended by constructs:
  - **Integrity constraints** for eliminating unwanted solution candidates
  - **Choice rules** for choosing subsets of atoms
  - **Cardinality rules** for counting certain present/absent atoms
- All of them can be translated back into normal logic program rules.
- The modelling methodology of ASP is **generate and test**:
  - Generate solution candidates, eliminate infeasible ones.



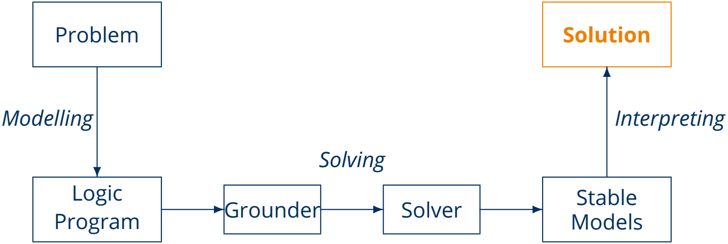
Previously ...

- The language of normal logic programs can be extended by constructs:
  - **Integrity constraints** for eliminating unwanted solution candidates
  - **Choice rules** for choosing subsets of atoms
  - **Cardinality rules** for counting certain present/absent atoms
- All of them can be translated back into normal logic program rules.
- The modelling methodology of ASP is **generate and test**:
  - Generate solution candidates, eliminate infeasible ones.



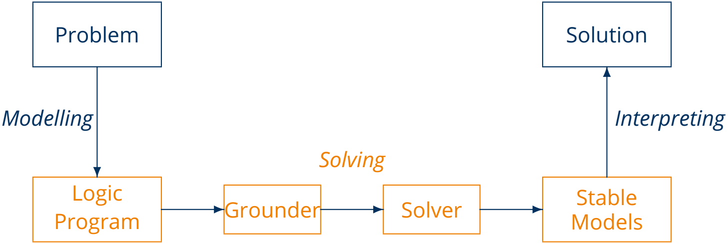
Previously . . .

- The language of normal logic programs can be extended by constructs:
  - **Integrity constraints** for eliminating unwanted solution candidates
  - **Choice rules** for choosing subsets of atoms
  - **Cardinality rules** for counting certain present/absent atoms
- All of them can be translated back into normal logic program rules.
- The modelling methodology of ASP is **generate and test**:
  - Generate solution candidates, eliminate infeasible ones.



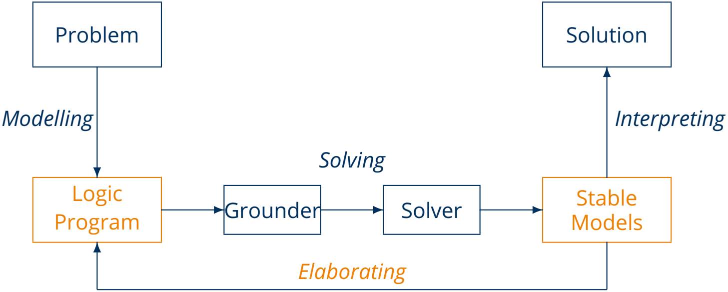
Previously ...

- The language of normal logic programs can be extended by constructs:
  - **Integrity constraints** for eliminating unwanted solution candidates
  - **Choice rules** for choosing subsets of atoms
  - **Cardinality rules** for counting certain present/absent atoms
- All of them can be translated back into normal logic program rules.
- The modelling methodology of ASP is **generate and test**:
  - Generate solution candidates, eliminate infeasible ones.



Previously ...

- The language of normal logic programs can be extended by constructs:
  - **Integrity constraints** for eliminating unwanted solution candidates
  - **Choice rules** for choosing subsets of atoms
  - **Cardinality rules** for counting certain present/absent atoms
- All of them can be translated back into normal logic program rules.
- The modelling methodology of ASP is **generate and test**:
  - Generate solution candidates, eliminate infeasible ones.



Overview

Computation
- Consequence Operator
- Computation from First Principles

Axiomatic Characterisation
- Completion
- Tightness
- Loops and Loop Formulas

Computation

Consequence Operator

Definition

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

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

Consequence Operator

Definition

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 $i \geq 1$

Consequence Operator

Definition

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 $\subseteq$-least fixpoint of T_P

An Example

- Consider the program

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

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

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

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 $\subseteq$-least 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 $X \subset \{p,q,r,t,s\}$

Approximating Stable Models

First Idea

Approximate a stable model X by two atom sets 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

Approximating Stable Models

First Idea

Approximate a stable model X by two atom sets 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

$$L \subseteq U \text{ implies } P^U \subseteq P^L \text{ implies } Cn(P^U) \subseteq Cn(P^L)$$

Approximating Stable Models

First Idea

Approximate a stable model X by two atom sets 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

$$L \subseteq U \text{ implies } P^U \subseteq P^L \text{ implies } Cn(P^U) \subseteq Cn(P^L)$$

Properties

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

Approximating Stable Models

First Idea

Approximate a stable model X by two atom sets 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

$$L \subseteq U \text{ implies } P^U \subseteq P^L \text{ implies } Cn(P^U) \subseteq Cn(P^L)$$

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 atom sets 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

$$L \subseteq U \text{ implies } P^U \subseteq P^L \text{ implies } Cn(P^U) \subseteq Cn(P^L)$$

Properties

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

- If $L \subseteq X$, then $X \subseteq Cn(P^L)$

Approximating Stable Models

First Idea

Approximate a stable model X by two atom sets 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

$$L \subseteq U \text{ implies } P^U \subseteq P^L \text{ implies } Cn(P^U) \subseteq Cn(P^L)$$

Properties

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

- If $L \subseteq X$, then $X \subseteq Cn(P^L)$
- If $X \subseteq U$,

Approximating Stable Models

First Idea

Approximate a stable model X by two atom sets 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

$$L \subseteq U \text{ implies } P^U \subseteq P^L \text{ implies } Cn(P^U) \subseteq Cn(P^L)$$

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 $Cn(P^U) \subseteq X$

Approximating Stable Models

First Idea

Approximate a stable model X by two atom sets 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

$$L \subseteq U \text{ implies } P^U \subseteq P^L \text{ implies } Cn(P^U) \subseteq Cn(P^L)$$

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 $L \subseteq X \subseteq U$,

Approximating Stable Models

First Idea

Approximate a stable model X by two atom sets 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

$$L \subseteq U \text{ implies } P^U \subseteq P^L \text{ implies } Cn(P^U) \subseteq Cn(P^L)$$

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*

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

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

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

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 $U = U'$

The algorithm:

- tightens the approximation on stable models
- is stable model preserving

An Example

Consider $P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, {\sim}c \\ d \leftarrow b, {\sim}e \\ e \leftarrow \ {\sim}d \end{array} \right\}$ over atoms $\mathcal{A} = \{a, b, c, d, e\}$.

An Example

Consider $P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, \sim c \\ d \leftarrow b, \sim e \\ e \leftarrow \ \sim d \end{array} \right\}$ over atoms $\mathcal{A} = \{a, b, c, d, e\}$.

The **expand** algorithm – started on the trivial pair $(\emptyset, \mathcal{A})$ – yields:

	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	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$	$\{a, b, d, e\}$

An Example

Consider $P = \left\{ \begin{array}{l} a \leftarrow \\ b \leftarrow a, \sim c \\ d \leftarrow b, \sim e \\ e \leftarrow \ \sim d \end{array} \right\}$ over atoms $\mathcal{A} = \{a, b, c, d, e\}$.

The **expand** algorithm – started on the trivial pair $(\emptyset, \mathcal{A})$ – yields:

	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.

Let us expand with d ...

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

Let us expand with d ...

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

	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	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$	$\{a, b, d\}$

Let us 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 us expand with $\sim d$ ...

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

Let us 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	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$	$\{a, b, e\}$

Let us 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 for SAT solving:

A Simplistic Solving Algorithm

Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure for SAT solving:

- Backtracking search building a binary search tree
- A node in the search tree corresponds to a three-valued interpretation

A Simplistic Solving Algorithm

Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure for SAT solving:

- 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**)

A Simplistic Solving Algorithm

Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure for SAT solving:

- 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

Quiz: Solving

solve$_P(L, U)$
  $(L, U) \leftarrow expand_P(L, U)$
  if $L \not\subseteq U$ **then failure**
  if $L = U$ **then output** L
  else choose $a \in U \setminus L$
    $solve_P(L \cup \{a\}, U)$
    $solve_P(L, U \setminus \{a\})$

expand$_P(L, U)$
  repeat
    $L' \leftarrow L;\ \ L \leftarrow L' \cup Cn(P^{U'})$
    $U' \leftarrow U;\ \ U \leftarrow U' \cap Cn(P^{L'})$
    if $L \not\subseteq U$ **then return**
  until $L = L'$ and $U = U'$

Quiz

...

Axiomatic Characterisation

Motivation

- There exist sophisticated algorithms and efficient implementations for SATisfiability testing in propositional logic
- Can we harness these systems for answer set programming?

Question

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

Motivation

- There exist sophisticated algorithms and efficient implementations for SATisfiability testing in propositional logic
- Can we harness these systems for answer set programming?

Question

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

Recall

- For every normal program P, there is a propositional theory $comp(P)$ such that its models correspond one-to-one to the supported models of P.

Motivation

- There exist sophisticated algorithms and efficient implementations for SATisfiability testing in propositional logic
- Can we harness these systems for answer set programming?

Question

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

Recall

- For every normal program P, there is a propositional theory $comp(P)$ such that its models correspond one-to-one to the supported models of P.
- Every stable model is a supported model, but not vice versa.

Motivation

- There exist sophisticated algorithms and efficient implementations for SATisfiability testing in propositional logic
- Can we harness these systems for answer set programming?

Question

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

Recall

- For every normal program P, there is a propositional theory $comp(P)$ such that its models correspond one-to-one to the supported models of P.
- Every stable model is a supported model, but not vice versa.

$\rightsquigarrow$ Can we add a second theory $T(P)$ such that the models of $comp(P) \cup T(P)$ correspond one-to-one to the stable models of P?

Program Completion: A Closer Look

The theory $comp(P)$ is logically equivalent to $\overleftarrow{comp}(P) \cup \overrightarrow{comp}(P)$, where

$$\begin{aligned}
\overleftarrow{comp}(P) &= \left\{ a \leftarrow \bigvee\nolimits_{B \in body_P(a)} BF(B) \;\middle|\; a \in atom(P) \right\} \\
\overrightarrow{comp}(P) &= \left\{ a \rightarrow \bigvee\nolimits_{B \in body_P(a)} BF(B) \;\middle|\; a \in atom(P) \right\}
\end{aligned}$$

$$\begin{aligned}
body_P(a) &= \{ body(r) \mid r \in P \text{ and } head(r) = a \} \\
BF(body(r)) &= \bigwedge\nolimits_{a \in body(r)^+} a \wedge \bigwedge\nolimits_{a \in body(r)^-} \neg a
\end{aligned}$$

- $\overleftarrow{comp}(P)$ characterises the classical models of P.
- $\overrightarrow{comp}(P)$ characterises that all true atoms must be supported.
- $\rightsquigarrow$ How to axiomatise that all true atoms must be well-supported?

Stable vs. Supported Models: An Example

Example

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

Stable vs. Supported Models: An Example

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 $\{a, b, c, d, e, f\}$.

Stable vs. Supported Models: An Example

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 $\{a, c, e\}$.

Stable vs. Supported Models: An Example

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 $\{a, d\}$.

Stable vs. Supported Models: An Example

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 $\{a, d\}$.
- The model $\{a, c, e\}$ is not well-supported (stable) because e supports itself.

Stable vs. Supported Models: An Example

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 $\{a, d\}$.
- The model $\{a, c, e\}$ is not well-supported (stable) because e supports itself.

Observation

Atoms in a strictly positive cycle (not being “supported from outside the cycle”) cannot be “derived” from a program in a finite number of steps.

Positive Atom Dependency Graph

Definition

The **positive atom dependency graph** $G(P)$ of a logic program P is given by

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

A logic program P is called **tight** $:\Longleftrightarrow$ $G(P)$ is acyclic.

Example

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

Theorem (Fages)

For tight normal logic programs, stable and supported models coincide.

Positive Atom Dependency Graph

Definition

The **positive atom dependency graph** $G(P)$ of a logic program P is given by

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

A logic program P is called **tight** $:\Longleftrightarrow$ $G(P)$ is acyclic.

Example

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

Theorem (Fages)

For tight normal logic programs, stable and supported models coincide.

Positive Atom Dependency Graph

Definition

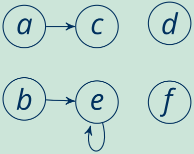
The **positive atom dependency graph** $G(P)$ of a logic program P is given by

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

A logic program P is called **tight** $:\Longleftrightarrow$ $G(P)$ is acyclic.

Example

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



Theorem (Fages)

For tight normal logic programs, stable and supported models coincide.

Positive Atom Dependency Graph

Definition

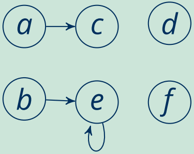
The **positive atom dependency graph** $G(P)$ of a logic program P is given by

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

A logic program P is called **tight** $:\Longleftrightarrow$ $G(P)$ 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 $\{a, d\}$



Theorem (Fages)

For tight normal logic programs, stable and supported models coincide.

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.

Idea

Add formulas prohibiting circular support of sets of atoms.

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

Definition

Let P be a normal logic program with positive atom dependency graph $G(P) = (atom(P), E)$.

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

Observation

A program P is tight iff $loops(P) = \emptyset$.

Loops

Definition

Let P be a normal logic program with positive atom dependency graph $G(P) = (atom(P), E)$.

- A non-empty set $L \subseteq atom(P)$ is a **loop** of P
  $:\Longleftrightarrow$ it induces a non-trivial strongly connected subgraph of $G(P)$.

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

Observation

A program P is tight iff $loops(P) = \emptyset$.

Loops

Definition

Let P be a normal logic program with positive atom dependency graph $G(P) = (atom(P), E)$.

- A non-empty set $L \subseteq atom(P)$ is a **loop** of P
  $:\Longleftrightarrow$ it induces a non-trivial strongly connected subgraph of $G(P)$.
- We denote the set of all loops of P by $loops(P)$.

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

Observation

A program P is tight iff $loops(P) = \emptyset$.

Loops: Examples (1)

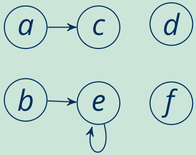
Example

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

Loops: Examples (1)

Example

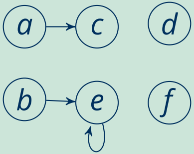
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$



Loops: Examples (1)

Example

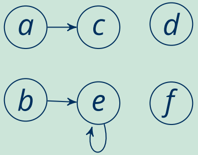
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$
- $loops(P) = \{\{e\}\}$



Loops: Examples (1)

Example

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



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

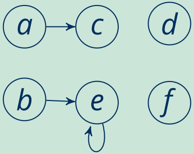
Example

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

Loops: Examples (1)

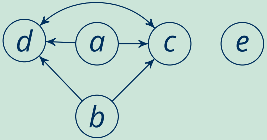
Example

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



Example

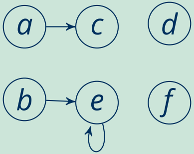
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a, b & d \leftarrow a & e \leftarrow {\sim}a, {\sim}b \\ b \leftarrow {\sim}a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$



Loops: Examples (1)

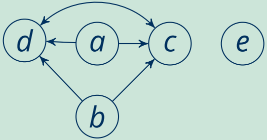
Example

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



Example

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



Loops: Examples (2)

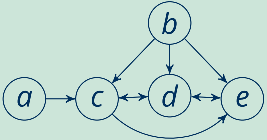
Example

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

Loops: Examples (2)

Example

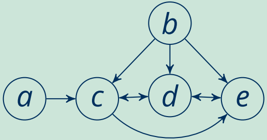
- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, \sim a \\ b \leftarrow \sim a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



Loops: Examples (2)

Example

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



Loop Formulas

Definition

Let P be a normal logic program.

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

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

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

Loop Formulas

Definition

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

Loop Formulas

Definition

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

$$LF_P(L) := \left(\bigvee_{a \in L} a\right) \to \left(\bigvee_{B \in EB_P(L)} BF(B)\right) \equiv \left(\bigwedge_{B \in EB_P(L)} \neg BF(B)\right) \to \left(\bigwedge_{a \in L} \neg a\right)$$

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

Loop Formulas

Definition

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

$$LF_P(L) := \left(\bigvee_{a \in L} a\right) \to \left(\bigvee_{B \in EB_P(L)} BF(B)\right) \equiv \left(\bigwedge_{B \in EB_P(L)} \neg BF(B)\right) \to \left(\bigwedge_{a \in L} \neg a\right)$$

- Define $LF(P) := \{LF_P(L) \mid L \in loops(P)\}$.

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

Loop Formulas: Examples (1)

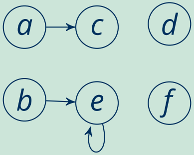
Example

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

Loop Formulas: Examples (1)

Example

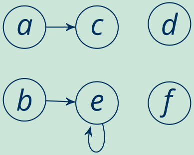
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, {\sim}d & e \leftarrow b, {\sim}f \\ b \leftarrow {\sim}a & d \leftarrow {\sim}c, {\sim}e & e \leftarrow e \end{array} \right\}$



Loop Formulas: Examples (1)

Example

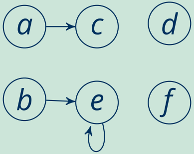
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, {\sim}d & e \leftarrow b, {\sim}f \\ b \leftarrow {\sim}a & d \leftarrow {\sim}c, {\sim}e & e \leftarrow e \end{array} \right\}$
- $loops(P) = \{\{e\}\}$
- $LF(P) = \{e \rightarrow b \wedge \neg f\}$



Loop Formulas: Examples (1)

Example

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



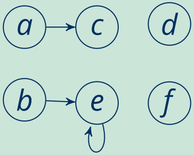
Example

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

Loop Formulas: Examples (1)

Example

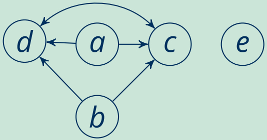
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$



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

Example

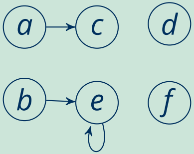
- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a, b & d \leftarrow a & e \leftarrow \sim a, \sim b \\ b \leftarrow \sim a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$



Loop Formulas: Examples (1)

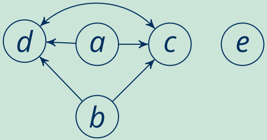
Example

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



Example

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



Loops: Examples (2)

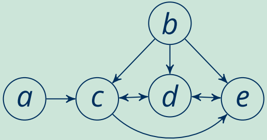
Example

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

Loops: Examples (2)

Example

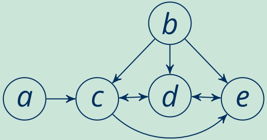
- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, \sim a \\ b \leftarrow \sim a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$
- $loops(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$



Loops: Examples (2)

Example

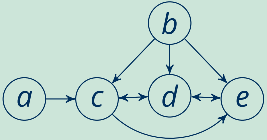
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, {\sim}a \\ b \leftarrow {\sim}a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$
- $loops(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$
- $LF(P) = \left\{ \begin{array}{l}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



Loops: Examples (2)

Example

- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, {\sim}a \\ b \leftarrow {\sim}a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$
- $loops(P) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$
- $LF(P) = \left\{ \begin{array}{l} c \lor d \rightarrow a \lor e \\ d \lor e \rightarrow (b \land c) \lor (b \land \neg a) \\ c \lor d \lor e \rightarrow a \lor (b \land \neg a) \end{array} \right\}$



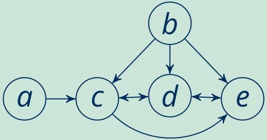
Loops: Examples (2)

Example

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

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

- $LF(P) = \left\{ \begin{array}{l} c \lor d \rightarrow a \lor e \\ d \lor e \rightarrow (b \land c) \lor (b \land \neg a) \\ c \lor d \lor e \rightarrow a \lor (b \land \neg a) \end{array} \right\}$



Lin-Zhao Theorem and Properties

Theorem (Lin and Zhao, 2004)

Let P be a normal logic program and $X \subseteq \mathit{atom}(P)$. Then:

$$X \text{ is a stable model of } P \text{ iff } X \models \mathit{comp}(P) \cup \mathit{LF}(P).$$

Lin-Zhao Theorem and Properties

Theorem (Lin and Zhao, 2004)

Let P be a normal logic program and $X \subseteq atom(P)$. Then:

$$X \text{ is a stable model of } P \text{ iff } X \models comp(P) \cup LF(P).$$

Properties of Loop Formulas

Let X be a supported model of normal LP P. Then, X is a stable model of P iff

- $X \models \{LF_P(U) \mid U \subseteq atom(P)\}$;
- $X \models \{LF_P(U) \mid U \subseteq X\}$;
- $X \models \{LF_P(L) \mid L \in loops(P)\}$, that is, $X \models LF(P)$;
- $X \models \{LF_P(L) \mid L \in loops(P) \text{ and } L \subseteq X\}$.

- If supported X is not stable for P, there is a loop $L \subseteq X \setminus Cn(P^X)$ with $X \not\models LF_P(L)$.
- There might be exponentially many loop formulas.
- Blowup seems to be unavoidable in general [Lifschitz and Razborov, 2006].

Conclusion

Summary

- The stable models of P can be approximated using the operator T_P:
$$(L, U) \rightsquigarrow \left(L \cup \bigcup_{i \geq 0} T^i_{P^U}(\emptyset), U \cap \bigcup_{i \geq 0} T^i_{P^L}(\emptyset)\right)$$
- Solving may use non-deterministic choice, propagation, and backtracking.
- Supported non-stable models are caused by loops in the program.
- A **loop** is a non-empty set of atoms that mutually depend on each other.
- The **loop formulas** $LF(P)$ of P enforce that every support is well-founded.
- The stable models of P can be characterised by $comp(P) \cup LF(P)$.

Suggested action points:

- Prove the properties on Slide 7.
- Try the algorithm on Slide 13 for some example programs.

Course Summary

- LPs are a declarative language for knowledge representation and reasoning.
- PROLOG-based logic programming focuses on theorem proving.
- PROLOG is also a programming language (via non-logical side effects).
- For definite LPs, SLD resolution is a sound and complete proof theory.
- For normal LPs, SLDNF resolution is sound and (sometimes) complete.
- Stable models are recognised as the "standard" semantics for normal LPs.
- ASP-based logic programming focuses on model generation.
- ASP is a modelling language for problem solving.
- Its modelling methodology is based on the generate-and-test paradigm.
- ASP solvers can make use of technology from propositional satisfiability.