Answer Set Solving in Practice

Torsten Schaub
University of Potsdam
torsten@cs.uni-potsdam.de

Axiomatic Characterization: Overview

1. Completion

2. Tightness

3. Loops and Loop Formulas

Outline

1 Completion

2 Tightness

3 Loops and Loop Formulas

Motivation

- Question Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of P ?
- Observation Although each atom is defined through a set of rules, each such rule provides only a sufficient condition for its head atom
- Idea The idea of program completion is to turn such implications into a definition by adding the corresponding necessary counterpart

Motivation

- Question Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of P ?
- Observation Although each atom is defined through a set of rules, each such rule provides only a sufficient condition for its head atom
- Idea The idea of program completion is to turn such implications into a definition by adding the corresponding necessary counterpart

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

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

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

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

- $\overleftarrow{CF}(P)$ 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

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

Potassco

A closer look

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

Potassco

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

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

Potassco

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 $\{a, d\}$

Potassco

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

1 Completion

2 Tightness

3 Loops and Loop Formulas

The mismatch

- Question What causes the mismatch between supported models and stable models?
- Hint Consider the unstable yet supported model $\{a, c, e\}$ of P !
- Answer Cyclic derivations are causing the mismatch between supported and stable models

    Atoms in a stable model can be "derived" from a program in a finite number of steps

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

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

Potassco

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
    - 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

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

  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 $P^{\{a,c,e\}}$

Potassco

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 $P^{\{a,c,e\}}$

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

Potassco

Positive atom dependency graph

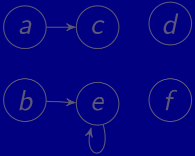
- The origin of (potential) circular derivations can be read off the positive atom dependency graph $G(P)$ of a logic program P given by

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

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

Example

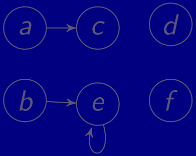
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e, f\}, \{(a, c), (b, e), (e, e)\})$



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

Example

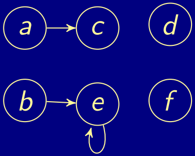
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e, f\}, \{(a, c), (b, e), (e, e)\})$



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

Example

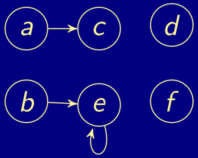
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e, f\}, \{(a, c), (b, e), (e, e)\})$



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

Example

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



- 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

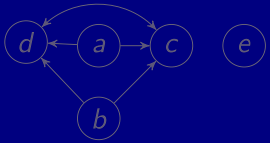
- A logic program P is called tight, if $G(P)$ is acyclic
- For tight programs, stable and supported models coincide:

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

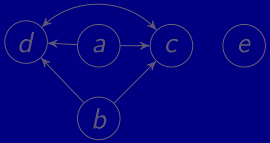
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a, b & d \leftarrow a & e \leftarrow {\sim}a, {\sim}b \\ b \leftarrow {\sim}a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\})$



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

Another example

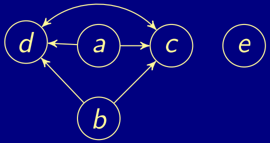
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a, b & d \leftarrow a & e \leftarrow {\sim}a, {\sim}b \\ b \leftarrow {\sim}a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\})$



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

Another example

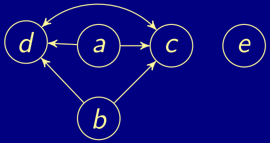
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a, b & d \leftarrow a & e \leftarrow {\sim}a, {\sim}b \\ b \leftarrow {\sim}a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$
- $G(P) = (\{a, b, c, d, e\}, \{(a, c), (a, d), (b, c), (b, d), (c, d), (d, c)\})$



- 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

1 Completion

2 Tightness

3 Loops and Loop Formulas

Motivation

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

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

- Idea Add formulas prohibiting circular support of sets of atoms
- Note Circular support between atoms a and b is possible, if a has a path to b and b has a path to a in the program's positive atom dependency graph

Motivation

- Question Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of P ?
- Observation Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program
- Idea Add formulas prohibiting circular support of sets of atoms
- Note Circular support between atoms a and b is possible, if a has a path to b and b has a path to a in the program's positive atom dependency graph

Motivation

- Question Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of P ?
- Observation Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program
- Idea Add formulas prohibiting circular support of sets of atoms
- Note Circular support between atoms a and b is possible, if a has a path to b and b has a path to a in the program's positive atom dependency graph

Motivation

- Question Is there a propositional formula $F(P)$ such that the models of $F(P)$ correspond to the stable models of P ?
- Observation Starting from the completion of a program, the problem boils down to eliminating the circular support of atoms holding in the supported models of the program
- Idea Add formulas prohibiting circular support of sets of atoms
- Note Circular support between atoms a and b is possible, if a has a path to b and b has a path to a in the program's positive atom dependency graph

Loops

Let P be a normal logic program, and
let $G(P) = (atom(P), E)$ 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 $G(P)$

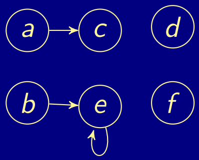
  That is, each pair of atoms in L is connected by a path of non-zero length in $(L, E \cap (L \times L))$

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

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

Example

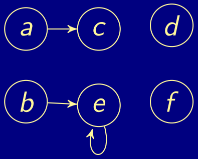
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$



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

Example

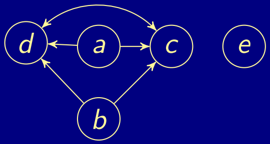
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$



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

Another example

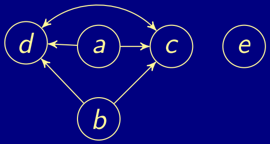
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a, b & d \leftarrow a & e \leftarrow {\sim}a, {\sim}b \\ b \leftarrow {\sim}a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$



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

Another example

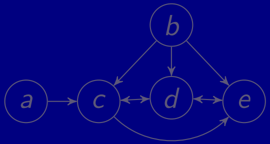
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a, b & d \leftarrow a & e \leftarrow {\sim}a, {\sim}b \\ b \leftarrow {\sim}a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$



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

Yet another example

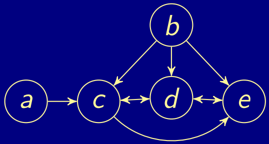
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, {\sim}a \\ b \leftarrow {\sim}a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



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

Yet another example

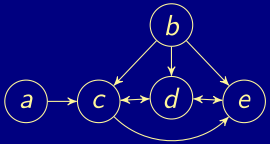
- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, \sim a \\ b \leftarrow \sim a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



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

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

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

- 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

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

- 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

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

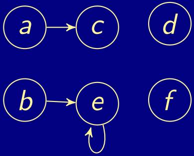
- Define the external bodies of L in P as $EB_P(L) = body(ES_P(L))$
- The (disjunctive) loop formula of L for P is

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

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

Example

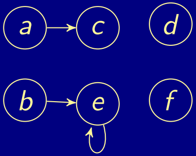
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$



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

Example

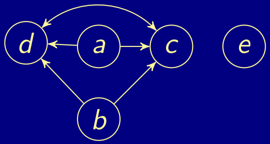
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, {\sim}d & e \leftarrow b, {\sim}f \\ b \leftarrow {\sim}a & d \leftarrow {\sim}c, {\sim}e & e \leftarrow e \end{array} \right\}$



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

Another example

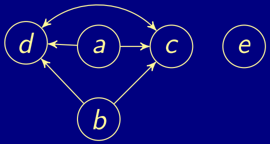
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a, b & d \leftarrow a & e \leftarrow {\sim}a, {\sim}b \\ b \leftarrow {\sim}a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$



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

Another example

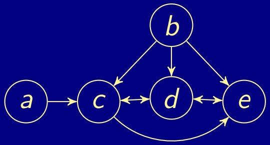
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a, b & d \leftarrow a & e \leftarrow {\sim}a, {\sim}b \\ b \leftarrow {\sim}a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$



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

Yet another example

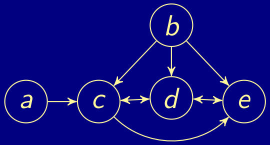
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, {\sim}a \\ b \leftarrow {\sim}a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



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

Yet another example

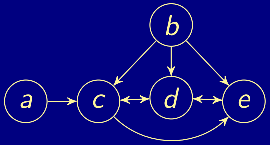
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, {\sim}a \\ b \leftarrow {\sim}a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



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

Yet another example

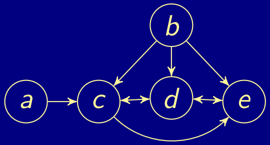
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, {\sim}a \\ b \leftarrow {\sim}a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



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

Yet another example

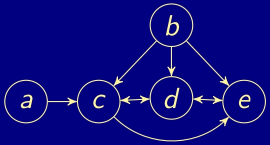
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, {\sim}a \\ b \leftarrow {\sim}a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



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

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, $X \models LF(P)$
  - $X \models \{LF_P(L) \mid L \in loop(P) \text{ and } L \subseteq X\}$

- 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, $X \models LF(P)$
  - $X \models \{LF_P(L) \mid L \in loop(P)$ 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 (ctd)

- Result If $\mathcal{P} \not\subseteq \mathcal{NC}^1/poly$,[2] then there is no translation $\mathcal{T}$ from logic programs to propositional formulas such that, for each normal logic program P, both of the following conditions hold:
  1. The propositional variables in $\mathcal{T}[P]$ are a subset of $atom(P)$
  2. The size of $\mathcal{T}[P]$ is polynomial in the size of P

- Note Every vocabulary-preserving translation from normal logic programs to propositional formulas must be exponential (in the worst case)

- Observations
  - Translation $CF(P) \cup LF(P)$ preserves the vocabulary of P
  - The number of loops in $loop(P)$ may be exponential in $|atom(P)|$

[2] A conjecture from complexity theory that is believed to be true

Loops and loop formulas: Properties (ctd)

- Result If $\mathcal{P} \not\subseteq \mathcal{NC}^1/poly$,[2] then there is no translation $\mathcal{T}$ from logic programs to propositional formulas such that, for each normal logic program P, both of the following conditions hold:
  1. The propositional variables in $\mathcal{T}[P]$ are a subset of $atom(P)$
  2. The size of $\mathcal{T}[P]$ is polynomial in the size of P

- Note Every vocabulary-preserving translation from normal logic programs to propositional formulas must be exponential (in the worst case)

- Observations
  - Translation $CF(P) \cup LF(P)$ preserves the vocabulary of P
  - The number of loops in $loop(P)$ may be exponential in $|atom(P)|$

[2] A conjecture from complexity theory that is believed to be true

Potassco

Loops and loop formulas: Properties (ctd)

- Result If $\mathcal{P} \not\subseteq \mathcal{NC}^1/poly$,[2] then there is no translation $\mathcal{T}$ from logic programs to propositional formulas such that, for each normal logic program P, both of the following conditions hold:
  1. The propositional variables in $\mathcal{T}[P]$ are a subset of $atom(P)$
  2. The size of $\mathcal{T}[P]$ is polynomial in the size of P

- Note Every vocabulary-preserving translation from normal logic programs to propositional formulas must be exponential (in the worst case)

- Observations
  - Translation $CF(P) \cup LF(P)$ preserves the vocabulary of P
  - The number of loops in $loop(P)$ may be exponential in $|atom(P)|$

[2] A conjecture from complexity theory that is believed to be true

Operational Characterization: Overview

4 Partial Interpretations

5 Fitting Operator

6 Unfounded Sets

7 Well-Founded Operator

Outline

4 Partial Interpretations

5 Fitting Operator

6 Unfounded Sets

7 Well-Founded Operator

Interlude: Partial interpretations

or: 3-valued interpretations

A partial interpretation maps atoms onto truth values *true*, *false*, and *unknown*

- Representation $\langle T, F \rangle$, where
  - T is the set of all *true* atoms and
  - F is the set of all *false* atoms
  - Truth of atoms in $\mathcal{A} \setminus (T \cup F)$ is *unknown*
- Properties
  - $\langle T, F \rangle$ is conflicting if $T \cap F \neq \emptyset$
  - $\langle T, F \rangle$ is total if $T \cup F = \mathcal{A}$ and $T \cap F = \emptyset$
- Definition For $\langle T_1, F_1 \rangle$ and $\langle T_2, F_2 \rangle$, define
  - $\langle T_1, F_1 \rangle \sqsubseteq \langle T_2, F_2 \rangle$ iff $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$
  - $\langle T_1, F_1 \rangle \sqcup \langle T_2, F_2 \rangle = \langle T_1 \cup T_2, F_1 \cup F_2 \rangle$

Interlude: Partial interpretations

or: 3-valued interpretations

A partial interpretation maps atoms onto truth values *true*, *false*, and *unknown*

- Representation $\langle T, F\rangle$, where
  - T is the set of all *true* atoms and
  - F is the set of all *false* atoms
  - Truth of atoms in $\mathcal{A} \setminus (T \cup F)$ is *unknown*
- Properties
  - $\langle T, F\rangle$ is conflicting if $T \cap F \neq \emptyset$
  - $\langle T, F\rangle$ is total if $T \cup F = \mathcal{A}$ and $T \cap F = \emptyset$
- Definition For $\langle T_1, F_1\rangle$ and $\langle T_2, F_2\rangle$, define
  - $\langle T_1, F_1\rangle \sqsubseteq \langle T_2, F_2\rangle$ iff $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$
  - $\langle T_1, F_1\rangle \sqcup \langle T_2, F_2\rangle = \langle T_1 \cup T_2, F_1 \cup F_2\rangle$

Interlude: Partial interpretations

or: 3-valued interpretations

A partial interpretation maps atoms onto truth values *true*, *false*, and *unknown*

- Representation $\langle T, F \rangle$, where
  - T is the set of all *true* atoms and
  - F is the set of all *false* atoms
  - Truth of atoms in $\mathcal{A} \setminus (T \cup F)$ is *unknown*
- Properties
  - $\langle T, F \rangle$ is conflicting if $T \cap F \neq \emptyset$
  - $\langle T, F \rangle$ is total if $T \cup F = \mathcal{A}$ and $T \cap F = \emptyset$
- Definition For $\langle T_1, F_1 \rangle$ and $\langle T_2, F_2 \rangle$, define
  - $\langle T_1, F_1 \rangle \sqsubseteq \langle T_2, F_2 \rangle$ iff $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$
  - $\langle T_1, F_1 \rangle \sqcup \langle T_2, F_2 \rangle = \langle T_1 \cup T_2, F_1 \cup F_2 \rangle$

Interlude: Partial interpretations

or: 3-valued interpretations

A partial interpretation maps atoms onto truth values *true*, *false*, and *unknown*

- Representation $\langle T, F \rangle$, where
  - T is the set of all *true* atoms and
  - F is the set of all *false* atoms
  - Truth of atoms in $\mathcal{A} \setminus (T \cup F)$ is *unknown*
- Properties
  - $\langle T, F \rangle$ is conflicting if $T \cap F \neq \emptyset$
  - $\langle T, F \rangle$ is total if $T \cup F = \mathcal{A}$ and $T \cap F = \emptyset$
- Definition For $\langle T_1, F_1 \rangle$ and $\langle T_2, F_2 \rangle$, define
  - $\langle T_1, F_1 \rangle \sqsubseteq \langle T_2, F_2 \rangle$ iff $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$
  - $\langle T_1, F_1 \rangle \sqcup \langle T_2, F_2 \rangle = \langle T_1 \cup T_2, F_1 \cup F_2 \rangle$

Outline

4 Partial Interpretations

5 Fitting Operator

6 Unfounded Sets

7 Well-Founded Operator

Basic idea

- Idea Extend T_P to normal logic programs
- Logical background The idea is to turn a program's completion into an operator such that
  - the head atom of a rule must be *true*
    if the rule's body is *true*
  - an atom must be *false*
    if the body of each rule having it as head is *false*

Definition

- Let P be a normal logic program
- Define

$$\mathbf{\Phi}_P\langle T,F\rangle = \langle \mathbf{T}_P\langle T,F\rangle, \mathbf{F}_P\langle T,F\rangle\rangle$$

where

$$\mathbf{T}_P\langle T,F\rangle = \{head(r) \mid r \in P, body(r)^+ \subseteq T, body(r)^- \subseteq F\}$$
$$\mathbf{F}_P\langle T,F\rangle = \{a \in atom(P) \mid \\ body(r)^+ \cap F \neq \emptyset \text{ or } body(r)^- \cap T \neq \emptyset \\ \text{for each } r \in P \text{ such that } head(r) = a\}$$

Potassco

Definition

- Let P be a normal logic program
- Define

$$\mathbf{\Phi}_P\langle T, F\rangle = \langle \mathbf{T}_P\langle T, F\rangle, \mathbf{F}_P\langle T, F\rangle\rangle$$

where

$$\begin{aligned}
\mathbf{T}_P\langle T, F\rangle &= \{\mathit{head}(r) \mid r \in P, \mathit{body}(r)^+ \subseteq T, \mathit{body}(r)^- \subseteq F\}\\
\mathbf{F}_P\langle T, F\rangle &= \{a \in \mathit{atom}(P) \mid \\
&\qquad \mathit{body}(r)^+ \cap F \neq \emptyset \text{ or } \mathit{body}(r)^- \cap T \neq \emptyset\\
&\qquad\quad \text{for each } r \in P \text{ such that } \mathit{head}(r) = a\ \}
\end{aligned}$$

Example

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

- Let's iterate $\mathbf{\Phi}_P$ on $\langle \{a\}, \{d\} \rangle$:

$$\begin{array}{rcl} \mathbf{\Phi}_P \langle \{a\}, \{d\} \rangle & = & \langle \{a, c\}, \{b, f\} \rangle \\ \mathbf{\Phi}_P \langle \{a, c\}, \{b, f\} \rangle & = & \langle \{a\}, \{b, d, f\} \rangle \\ \mathbf{\Phi}_P \langle \{a\}, \{b, d, f\} \rangle & = & \langle \{a, c\}, \{b, f\} \rangle \\ & \vdots & \end{array}$$

Example

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

- Let's iterate $\mathbf{\Phi}_P$ on $\langle \{a\}, \{d\} \rangle$:

$$\begin{array}{rcl} \mathbf{\Phi}_P \langle \{a\}, \{d\} \rangle & = & \langle \{a, c\}, \{b, f\} \rangle \\ \mathbf{\Phi}_P \langle \{a, c\}, \{b, f\} \rangle & = & \langle \{a\}, \{b, d, f\} \rangle \\ \mathbf{\Phi}_P \langle \{a\}, \{b, d, f\} \rangle & = & \langle \{a, c\}, \{b, f\} \rangle \\ & \vdots & \end{array}$$

Fitting semantics

- Define the iterative variant of $\mathbf{\Phi}_P$ analogously to T_P:

$$\mathbf{\Phi}_P^0\langle T, F\rangle = \langle T, F\rangle \qquad \mathbf{\Phi}_P^{i+1}\langle T, F\rangle = \mathbf{\Phi}_P\mathbf{\Phi}_P^i\langle T, F\rangle$$

- Define the Fitting semantics of a normal logic program P as the partial interpretation:

$$\bigsqcup_{i\geq 0}\mathbf{\Phi}_P^i\langle\emptyset,\emptyset\rangle$$

Fitting semantics

- Define the iterative variant of $\mathbf{\Phi}_P$ analogously to T_P:

$$\mathbf{\Phi}_P^0\langle T, F\rangle = \langle T, F\rangle \qquad \mathbf{\Phi}_P^{i+1}\langle T, F\rangle = \mathbf{\Phi}_P\mathbf{\Phi}_P^i\langle T, F\rangle$$

- Define the Fitting semantics of a normal logic program P as the partial interpretation:

$$\bigsqcup_{i\geq 0}\mathbf{\Phi}_P^i\langle\emptyset,\emptyset\rangle$$

Potassco

Example

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

$$\begin{array}{lcccc} \mathbf{\Phi}^0\langle\emptyset,\emptyset\rangle & = & & & \langle\emptyset,\emptyset\rangle \\ \mathbf{\Phi}^1\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{f\}\rangle \\ \mathbf{\Phi}^2\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}\langle\{a\},\{f\}\rangle & = & \langle\{a\},\{b,f\}\rangle \\ \mathbf{\Phi}^3\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}\langle\{a\},\{b,f\}\rangle & = & \langle\{a\},\{b,f\}\rangle \end{array}$$

$$\bigsqcup_{i\geq 0}\mathbf{\Phi}^i\langle\emptyset,\emptyset\rangle = \langle\{a\},\{b,f\}\rangle$$

Example

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

$$\begin{array}{rcccl} \mathbf{\Phi}^0\langle\emptyset,\emptyset\rangle & = & & & \langle\emptyset,\emptyset\rangle \\ \mathbf{\Phi}^1\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{f\}\rangle \\ \mathbf{\Phi}^2\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}\langle\{a\},\{f\}\rangle & = & \langle\{a\},\{b,f\}\rangle \\ \mathbf{\Phi}^3\langle\emptyset,\emptyset\rangle & = & \mathbf{\Phi}\langle\{a\},\{b,f\}\rangle & = & \langle\{a\},\{b,f\}\rangle \end{array}$$

$$\bigsqcup_{i\geq 0}\mathbf{\Phi}^i\langle\emptyset,\emptyset\rangle = \langle\{a\},\{b,f\}\rangle$$

Potassco

Properties

Let P be a normal logic program

- $\mathbf{\Phi}_P\langle\emptyset,\emptyset\rangle$ is monotonic
  That is, $\mathbf{\Phi}^i_P\langle\emptyset,\emptyset\rangle \sqsubseteq \mathbf{\Phi}^{i+1}_P\langle\emptyset,\emptyset\rangle$
- The Fitting semantics of P is
  - not conflicting,
  - and generally not total

Fitting fixpoints

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Define $\langle T, F\rangle$ as a Fitting fixpoint of P if $\mathbf{\Phi}_P\langle T, F\rangle = \langle T, F\rangle$
  - The Fitting semantics is the $\sqsubseteq$-least Fitting fixpoint of P
  - Any other Fitting fixpoint extends the Fitting semantics
  - Total Fitting fixpoints correspond to supported models

Fitting fixpoints

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Define $\langle T, F\rangle$ as a Fitting fixpoint of P if $\mathbf{\Phi}_P\langle T, F\rangle = \langle T, F\rangle$
  - The Fitting semantics is the $\sqsubseteq$-least Fitting fixpoint of P
  - Any other Fitting fixpoint extends the Fitting semantics
  - Total Fitting fixpoints correspond to supported models

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 three total Fitting fixpoints:
  $\langle\{a, c\}, \{b, d, e, f\}\rangle$
  $\langle\{a, d\}, \{b, c, e, f\}\rangle$
  $\langle\{a, c, e\}, \{b, d, f\}\rangle$

  P has three supported models, two of them are stable models

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 three total Fitting fixpoints:
  1. $\langle \{a, c\}, \{b, d, e, f\} \rangle$
  2. $\langle \{a, d\}, \{b, c, e, f\} \rangle$
  3. $\langle \{a, c, e\}, \{b, d, f\} \rangle$
- P has three supported models, two of them are stable models

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 three total Fitting fixpoints:
  1. $\langle\{a, c\}, \{b, d, e, f\}\rangle$
  2. $\langle\{a, d\}, \{b, c, e, f\}\rangle$
  3. $\langle\{a, c, e\}, \{b, d, f\}\rangle$
- P has three supported models, two of them are stable models

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 three total Fitting fixpoints:
  1. $\langle \{a, c\}, \{b, d, e, f\} \rangle$
  2. $\langle \{a, d\}, \{b, c, e, f\} \rangle$
  3. $\langle \{a, c, e\}, \{b, d, f\} \rangle$
- P has three supported models, two of them are stable models

Properties

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Let $\mathbf{\Phi}_P\langle T, F\rangle = \langle T', F'\rangle$
- If X is a stable model of P such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

  That is, $\mathbf{\Phi}_P$ is stable model preserving

  Hence, $\mathbf{\Phi}_P$ can be used for approximating stable models and so for propagation in ASP-solvers

- However, $\mathbf{\Phi}_P$ is still insufficient, because total fixpoints correspond to supported models, not necessarily stable models

  Note The problem is the same as with program completion

  The missing piece is non-circularity of derivations !

Properties

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Let $\mathbf{\Phi}_P\langle T, F\rangle = \langle T', F'\rangle$
- If X is a stable model of P such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

  That is, $\mathbf{\Phi}_P$ is stable model preserving

  Hence, $\mathbf{\Phi}_P$ can be used for approximating stable models and so for propagation in ASP-solvers

- However, $\mathbf{\Phi}_P$ is still insufficient, because total fixpoints correspond to supported models, not necessarily stable models

  Note The problem is the same as with program completion

  The missing piece is non-circularity of derivations !

Properties

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Let $\mathbf{\Phi}_P\langle T, F\rangle = \langle T', F'\rangle$
- If X is a stable model of P such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

  That is, $\mathbf{\Phi}_P$ is stable model preserving

  Hence, $\mathbf{\Phi}_P$ can be used for approximating stable models and so for propagation in ASP-solvers

- However, $\mathbf{\Phi}_P$ is still insufficient, because total fixpoints correspond to supported models, not necessarily stable models

  Note The problem is the same as with program completion

  The missing piece is non-circularity of derivations !

Properties

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Let $\mathbf{\Phi}_P\langle T, F\rangle = \langle T', F'\rangle$
- If X is a stable model of P such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

  That is, $\mathbf{\Phi}_P$ is stable model preserving

  Hence, $\mathbf{\Phi}_P$ can be used for approximating stable models and so for propagation in ASP-solvers

- However, $\mathbf{\Phi}_P$ is still insufficient, because total fixpoints correspond to supported models, not necessarily stable models

  Note The problem is the same as with program completion

  The missing piece is non-circularity of derivations !

Potassco

Properties

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Let $\mathbf{\Phi}_P\langle T, F\rangle = \langle T', F'\rangle$
- If X is a stable model of P such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

  That is, $\mathbf{\Phi}_P$ is stable model preserving

  Hence, $\mathbf{\Phi}_P$ can be used for approximating stable models and so for propagation in ASP-solvers

- However, $\mathbf{\Phi}_P$ is still insufficient, because total fixpoints correspond to supported models, not necessarily stable models

  Note The problem is the same as with program completion

  The missing piece is non-circularity of derivations !

Properties

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Let $\mathbf{\Phi}_P\langle T, F\rangle = \langle T', F'\rangle$
- If X is a stable model of P such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

  That is, $\mathbf{\Phi}_P$ is stable model preserving

  Hence, $\mathbf{\Phi}_P$ can be used for approximating stable models and so for propagation in ASP-solvers

- However, $\mathbf{\Phi}_P$ is still insufficient, because total fixpoints correspond to supported models, not necessarily stable models

  Note The problem is the same as with program completion

  The missing piece is non-circularity of derivations !

Example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

$$\begin{array}{rcl} \Phi_P^0\langle\emptyset,\emptyset\rangle & = & \langle\emptyset,\emptyset\rangle \\ \Phi_P^1\langle\emptyset,\emptyset\rangle & = & \langle\emptyset,\emptyset\rangle \end{array}$$

- That is, Fitting semantics cannot assign *false* to a and b, although they can never become *true* !

Example

$$P = \left\{ \begin{array}{ccc} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

$$\begin{array}{rcl} \Phi_P^0\langle\emptyset,\emptyset\rangle & = & \langle\emptyset,\emptyset\rangle \\ \Phi_P^1\langle\emptyset,\emptyset\rangle & = & \langle\emptyset,\emptyset\rangle \end{array}$$

- That is, Fitting semantics cannot assign *false* to a and b, although they can never become *true* !

Example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

$$\begin{array}{lcl} \Phi_P^0\langle\emptyset,\emptyset\rangle & = & \langle\emptyset,\emptyset\rangle \\ \Phi_P^1\langle\emptyset,\emptyset\rangle & = & \langle\emptyset,\emptyset\rangle \end{array}$$

- That is, Fitting semantics cannot assign *false* to a and b, although they can never become *true* !

Outline

4 Partial Interpretations

5 Fitting Operator

6 Unfounded Sets

7 Well-Founded Operator

Unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- A set $U \subseteq atom(P)$ is an unfounded set of P wrt $\langle T, F\rangle$, if for each rule $r \in P$ such that $head(r) \in U$, we have that

  1. $body(r)^+ \cap F \neq \emptyset$ or $body(r)^- \cap T \neq \emptyset$ or
  2. $body(r)^+ \cap U \neq \emptyset$

- Intuitively, $\langle T, F\rangle$ is what we already know about P
- Rules satisfying Condition 1 are not usable for further derivations
- Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true

Potassco

Unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- A set $U \subseteq atom(P)$ is an unfounded set of P wrt $\langle T, F\rangle$,
  if for each rule $r \in P$ such that $head(r) \in U$, we have that

  $body(r)^+ \cap F \neq \emptyset$ or $body(r)^- \cap T \neq \emptyset$ or
  $body(r)^+ \cap U \neq \emptyset$

  Intuitively, $\langle T, F\rangle$ is what we already know about P

  Rules satisfying Condition 1 are not usable for further derivations

  Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true

Unfounded sets

Let P be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation

- A set $U \subseteq atom(P)$ is an unfounded set of P wrt $\langle T, F \rangle$,
  if for each rule $r \in P$ such that $head(r) \in U$, we have that

  $body(r)^+ \cap F \neq \emptyset$ or $body(r)^- \cap T \neq \emptyset$ or
  $body(r)^+ \cap U \neq \emptyset$

- Intuitively, $\langle T, F \rangle$ is what we already know about P
  - Rules satisfying Condition 1 are not usable for further derivations
  - Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true

Unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- A set $U \subseteq atom(P)$ is an unfounded set of P wrt $\langle T, F\rangle$, if for each rule $r \in P$ such that $head(r) \in U$, we have that
  1. $body(r)^+ \cap F \neq \emptyset$ or $body(r)^- \cap T \neq \emptyset$ or
  2. $body(r)^+ \cap U \neq \emptyset$
- Intuitively, $\langle T, F\rangle$ is what we already know about P
- Rules satisfying Condition 1 are not usable for further derivations
- Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true

Unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- A set $U \subseteq atom(P)$ is an unfounded set of P wrt $\langle T, F\rangle$,
if for each rule $r \in P$ such that $head(r) \in U$, we have that
  1. $body(r)^+ \cap F \neq \emptyset$ or $body(r)^- \cap T \neq \emptyset$ or
  $body(r)^+ \cap U \neq \emptyset$

Intuitively, $\langle T, F\rangle$ is what we already know about P

Rules satisfying Condition 1 are not usable for further derivations

Condition 2 is the unfounded set condition treating cyclic derivations:
All rules still being usable to derive an atom in U require an(other)
atom in U to be true

Unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- A set $U \subseteq atom(P)$ is an unfounded set of P wrt $\langle T, F\rangle$, if for each rule $r \in P$ such that $head(r) \in U$, we have that
  1. $body(r)^+ \cap F \neq \emptyset$ or $body(r)^- \cap T \neq \emptyset$ or
  2. $body(r)^+ \cap U \neq \emptyset$
- Intuitively, $\langle T, F\rangle$ is what we already know about P
- Rules satisfying Condition 1 are not usable for further derivations
- Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true

Unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- A set $U \subseteq atom(P)$ is an unfounded set of P wrt $\langle T, F\rangle$, if for each rule $r \in P$ such that $head(r) \in U$, we have that
  1. $body(r)^+ \cap F \neq \emptyset$ or $body(r)^- \cap T \neq \emptyset$ or
  2. $body(r)^+ \cap U \neq \emptyset$

  Intuitively, $\langle T, F\rangle$ is what we already know about P
- Rules satisfying Condition 1 are not usable for further derivations
- Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true

Unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- A set $U \subseteq atom(P)$ is an unfounded set of P wrt $\langle T, F\rangle$, if for each rule $r \in P$ such that $head(r) \in U$, we have that
  1. $body(r)^+ \cap F \neq \emptyset$ or $body(r)^- \cap T \neq \emptyset$ or
  2. $body(r)^+ \cap U \neq \emptyset$
- Intuitively, $\langle T, F\rangle$ is what we already know about P
- Rules satisfying Condition 1 are not usable for further derivations
- Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true

Unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- A set $U \subseteq atom(P)$ is an unfounded set of P wrt $\langle T, F\rangle$, if for each rule $r \in P$ such that $head(r) \in U$, we have that
  1. $body(r)^+ \cap F \neq \emptyset$ or $body(r)^- \cap T \neq \emptyset$ or
  2. $body(r)^+ \cap U \neq \emptyset$
- Intuitively, $\langle T, F\rangle$ is what we already know about P
- Rules satisfying Condition 1 are not usable for further derivations
- Condition 2 is the unfounded set condition treating cyclic derivations: All rules still being usable to derive an atom in U require an(other) atom in U to be true

Example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition)
- $\{a\}$ is not an unfounded set of P wrt $\langle \emptyset, \emptyset \rangle$
- $\{a\}$ is an unfounded set of P wrt $\langle \emptyset, \{b\} \rangle$
- $\{a\}$ is not an unfounded set of P wrt $\langle \{b\}, \emptyset \rangle$
- Analogously for $\{b\}$
- $\{a, b\}$ is an unfounded set of P wrt $\langle \emptyset, \emptyset \rangle$
- $\{a, b\}$ is an unfounded set of P wrt any partial interpretation

Example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition)
- $\{a\}$ is not an unfounded set of P wrt $\langle \emptyset, \emptyset \rangle$
- $\{a\}$ is an unfounded set of P wrt $\langle \emptyset, \{b\} \rangle$
- $\{a\}$ is not an unfounded set of P wrt $\langle \{b\}, \emptyset \rangle$
- Analogously for $\{b\}$
- $\{a, b\}$ is an unfounded set of P wrt $\langle \emptyset, \emptyset \rangle$
- $\{a, b\}$ is an unfounded set of P wrt any partial interpretation

Example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition)
- $\{a\}$ is not an unfounded set of P wrt $\langle\emptyset, \emptyset\rangle$
- $\{a\}$ is an unfounded set of P wrt $\langle\emptyset, \{b\}\rangle$
- $\{a\}$ is not an unfounded set of P wrt $\langle\{b\}, \emptyset\rangle$
- Analogously for $\{b\}$
- $\{a, b\}$ is an unfounded set of P wrt $\langle\emptyset, \emptyset\rangle$
- $\{a, b\}$ is an unfounded set of P wrt any partial interpretation

Potassco

Example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition)
- $\{a\}$ is not an unfounded set of P wrt $\langle \emptyset, \emptyset \rangle$
- $\{a\}$ is an unfounded set of P wrt $\langle \emptyset, \{b\} \rangle$
- $\{a\}$ is not an unfounded set of P wrt $\langle \{b\}, \emptyset \rangle$
- Analogously for $\{b\}$
- $\{a, b\}$ is an unfounded set of P wrt $\langle \emptyset, \emptyset \rangle$
- $\{a, b\}$ is an unfounded set of P wrt any partial interpretation

Example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition)
- $\{a\}$ is not an unfounded set of P wrt $\langle\emptyset, \emptyset\rangle$
- $\{a\}$ is an unfounded set of P wrt $\langle\emptyset, \{b\}\rangle$
- $\{a\}$ is not an unfounded set of P wrt $\langle\{b\}, \emptyset\rangle$
- Analogously for $\{b\}$
- $\{a, b\}$ is an unfounded set of P wrt $\langle\emptyset, \emptyset\rangle$
- $\{a, b\}$ is an unfounded set of P wrt any partial interpretation

Example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition)
- $\{a\}$ is not an unfounded set of P wrt $\langle\emptyset,\emptyset\rangle$
- $\{a\}$ is an unfounded set of P wrt $\langle\emptyset,\{b\}\rangle$
- $\{a\}$ is not an unfounded set of P wrt $\langle\{b\},\emptyset\rangle$
- Analogously for $\{b\}$
- $\{a,b\}$ is an unfounded set of P wrt $\langle\emptyset,\emptyset\rangle$
- $\{a,b\}$ is an unfounded set of P wrt any partial interpretation

Potassco

Example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition)
- $\{a\}$ is not an unfounded set of P wrt $\langle \emptyset, \emptyset \rangle$
- $\{a\}$ is an unfounded set of P wrt $\langle \emptyset, \{b\} \rangle$
- $\{a\}$ is not an unfounded set of P wrt $\langle \{b\}, \emptyset \rangle$
- Analogously for $\{b\}$
- $\{a, b\}$ is an unfounded set of P wrt $\langle \emptyset, \emptyset \rangle$
- $\{a, b\}$ is an unfounded set of P wrt any partial interpretation

Example

$$P = \left\{ \begin{array}{lcl} a & \leftarrow & b \\ b & \leftarrow & a \end{array} \right\}$$

- $\emptyset$ is an unfounded set (by definition)
- $\{a\}$ is not an unfounded set of P wrt $\langle \emptyset, \emptyset \rangle$
- $\{a\}$ is an unfounded set of P wrt $\langle \emptyset, \{b\} \rangle$
- $\{a\}$ is not an unfounded set of P wrt $\langle \{b\}, \emptyset \rangle$
- Analogously for $\{b\}$
- $\{a, b\}$ is an unfounded set of P wrt $\langle \emptyset, \emptyset \rangle$
- $\{a, b\}$ is an unfounded set of P wrt any partial interpretation

Greatest unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Observation The union of two unfounded sets is an unfounded set
- The greatest unfounded set of P wrt $\langle T, F\rangle$ is the union of all unfounded sets of P wrt $\langle T, F\rangle$

  It is denoted by $\mathbf{U}_P\langle T, F\rangle$
- Alternatively, we may define

$$\mathbf{U}_P\langle T, F\rangle = atom(P) \setminus Cn(\{r \in P \mid body(r)^+ \cap F = \emptyset\}^T)$$

- Note $Cn(\{r \in P \mid body(r)^+ \cap F = \emptyset\}^T)$ contains all non-circularly derivable atoms from P wrt $\langle T, F\rangle$

Potassco

Greatest unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- **Observation** The union of two unfounded sets is an unfounded set
- The greatest unfounded set of P wrt $\langle T, F\rangle$ is the union of all unfounded sets of P wrt $\langle T, F\rangle$

  It is denoted by $\mathbf{U}_P\langle T, F\rangle$
- Alternatively, we may define

$$\mathbf{U}_P\langle T, F\rangle = atom(P) \setminus Cn(\{r \in P \mid body(r)^+ \cap F = \emptyset\}^T)$$

- **Note** $Cn(\{r \in P \mid body(r)^+ \cap F = \emptyset\}^T)$ contains all non-circularly derivable atoms from P wrt $\langle T, F\rangle$

Greatest unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Observation The union of two unfounded sets is an unfounded set
- The greatest unfounded set of P wrt $\langle T, F\rangle$ is the union of all unfounded sets of P wrt $\langle T, F\rangle$

  It is denoted by $\mathbf{U}_P\langle T, F\rangle$
- Alternatively, we may define

$$\mathbf{U}_P\langle T, F\rangle = atom(P) \setminus Cn(\{r \in P \mid body(r)^+ \cap F = \emptyset\}^T)$$

- Note $Cn(\{r \in P \mid body(r)^+ \cap F = \emptyset\}^T)$ contains all non-circularly derivable atoms from P wrt $\langle T, F\rangle$

Greatest unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Observation The union of two unfounded sets is an unfounded set
- The greatest unfounded set of P wrt $\langle T, F\rangle$ is the union of all unfounded sets of P wrt $\langle T, F\rangle$

  It is denoted by $\mathbf{U}_P\langle T, F\rangle$
- Alternatively, we may define

$$\mathbf{U}_P\langle T, F\rangle = atom(P) \setminus Cn(\{r \in P \mid body(r)^+ \cap F = \emptyset\}^T)$$

- Note $Cn(\{r \in P \mid body(r)^+ \cap F = \emptyset\}^T)$ contains all non-circularly derivable atoms from P wrt $\langle T, F\rangle$

Greatest unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Observation The union of two unfounded sets is an unfounded set
- The greatest unfounded set of P wrt $\langle T, F\rangle$ is the union of all unfounded sets of P wrt $\langle T, F\rangle$

  It is denoted by $\mathbf{U}_P\langle T, F\rangle$
- Alternatively, we may define

$$\mathbf{U}_P\langle T, F\rangle = atom(P) \setminus Cn(\{r \in P \mid body(r)^+ \cap F = \emptyset\}^T)$$

- Note $Cn(\{r \in P \mid body(r)^+ \cap F = \emptyset\}^T)$ contains all non-circularly derivable atoms from P wrt $\langle T, F\rangle$

Greatest unfounded sets

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Observation The union of two unfounded sets is an unfounded set
- The greatest unfounded set of P wrt $\langle T, F\rangle$ is the union of all unfounded sets of P wrt $\langle T, F\rangle$

  It is denoted by $\mathbf{U}_P\langle T, F\rangle$
- Alternatively, we may define

$$\mathbf{U}_P\langle T, F\rangle = atom(P) \setminus Cn(\{r \in P \mid body(r)^+ \cap F = \emptyset\}^T)$$

- Note $Cn(\{r \in P \mid body(r)^+ \cap F = \emptyset\}^T)$ contains all non-circularly derivable atoms from P wrt $\langle T, F\rangle$

Outline

4 Partial Interpretations

5 Fitting Operator

6 Unfounded Sets

7 Well-Founded Operator

Well-founded operator

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Observation Condition 1 (in the definition of an unfounded set) corresponds to $\mathbf{F}_P\langle T, F\rangle$ of Fitting's $\mathbf{\Phi}_P\langle T, F\rangle$
- Idea Extend (negative part of) Fitting's operator $\mathbf{\Phi}_P$

  That is,
  - keep definition of $\mathbf{T}_P\langle T, F\rangle$ from $\mathbf{\Phi}_P\langle T, F\rangle$ and
  - replace $\mathbf{F}_P\langle T, F\rangle$ from $\mathbf{\Phi}_P\langle T, F\rangle$ by $\mathbf{U}_P\langle T, F\rangle$
- In words, an atom must be *false*
  if it belongs to the greatest unfounded set
- Definition $\mathbf{\Omega}_P\langle T, F\rangle = \langle \mathbf{T}_P\langle T, F\rangle, \mathbf{U}_P\langle T, F\rangle\rangle$

$$\mathbf{\Phi}_P\langle T, F\rangle \sqsubseteq \mathbf{\Omega}_P\langle T, F\rangle$$

Well-founded operator

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Observation Condition 1 (in the definition of an unfounded set) corresponds to $\mathbf{F}_P\langle T, F\rangle$ of Fitting's $\mathbf{\Phi}_P\langle T, F\rangle$
- Idea Extend (negative part of) Fitting's operator $\mathbf{\Phi}_P$

  That is,
  - keep definition of $\mathbf{T}_P\langle T, F\rangle$ from $\mathbf{\Phi}_P\langle T, F\rangle$ and
  - replace $\mathbf{F}_P\langle T, F\rangle$ from $\mathbf{\Phi}_P\langle T, F\rangle$ by $\mathbf{U}_P\langle T, F\rangle$
- In words, an atom must be *false* if it belongs to the greatest unfounded set
- Definition $\mathbf{\Omega}_P\langle T, F\rangle = \langle \mathbf{T}_P\langle T, F\rangle, \mathbf{U}_P\langle T, F\rangle\rangle$
- Property $\mathbf{\Phi}_P\langle T, F\rangle \sqsubseteq \mathbf{\Omega}_P\langle T, F\rangle$

Well-founded operator

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Observation Condition 1 (in the definition of an unfounded set) corresponds to $\mathbf{F}_P\langle T, F\rangle$ of Fitting's $\mathbf{\Phi}_P\langle T, F\rangle$
- Idea Extend (negative part of) Fitting's operator $\mathbf{\Phi}_P$

  That is,
  - keep definition of $\mathbf{T}_P\langle T, F\rangle$ from $\mathbf{\Phi}_P\langle T, F\rangle$ and
  - replace $\mathbf{F}_P\langle T, F\rangle$ from $\mathbf{\Phi}_P\langle T, F\rangle$ by $\mathbf{U}_P\langle T, F\rangle$
- In words, an atom must be *false* if it belongs to the greatest unfounded set
- Definition $\mathbf{\Omega}_P\langle T, F\rangle = \langle \mathbf{T}_P\langle T, F\rangle, \mathbf{U}_P\langle T, F\rangle\rangle$
- Property $\mathbf{\Phi}_P\langle T, F\rangle \sqsubseteq \mathbf{\Omega}_P\langle T, F\rangle$

Potassco

Well-founded operator

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Observation Condition 1 (in the definition of an unfounded set) corresponds to $\mathbf{F}_P\langle T, F\rangle$ of Fitting's $\mathbf{\Phi}_P\langle T, F\rangle$
- Idea Extend (negative part of) Fitting's operator $\mathbf{\Phi}_P$

  That is,
  - keep definition of $\mathbf{T}_P\langle T, F\rangle$ from $\mathbf{\Phi}_P\langle T, F\rangle$ and
  - replace $\mathbf{F}_P\langle T, F\rangle$ from $\mathbf{\Phi}_P\langle T, F\rangle$ by $\mathbf{U}_P\langle T, F\rangle$
- In words, an atom must be *false* if it belongs to the greatest unfounded set
- Definition $\mathbf{\Omega}_P\langle T, F\rangle = \langle \mathbf{T}_P\langle T, F\rangle, \mathbf{U}_P\langle T, F\rangle\rangle$
- Property $\mathbf{\Phi}_P\langle T, F\rangle \sqsubseteq \mathbf{\Omega}_P\langle T, F\rangle$

Well-founded operator

Let P be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation

- Observation Condition 1 (in the definition of an unfounded set) corresponds to $\mathbf{F}_P\langle T, F \rangle$ of Fitting's $\mathbf{\Phi}_P\langle T, F \rangle$
- Idea Extend (negative part of) Fitting's operator $\mathbf{\Phi}_P$
  That is,
  - keep definition of $\mathbf{T}_P\langle T, F \rangle$ from $\mathbf{\Phi}_P\langle T, F \rangle$ and
  - replace $\mathbf{F}_P\langle T, F \rangle$ from $\mathbf{\Phi}_P\langle T, F \rangle$ by $\mathbf{U}_P\langle T, F \rangle$
- In words, an atom must be *false* if it belongs to the greatest unfounded set

- Definition $\mathbf{\Omega}_P\langle T, F \rangle = \langle \mathbf{T}_P\langle T, F \rangle, \mathbf{U}_P\langle T, F \rangle \rangle$
- Property $\mathbf{\Phi}_P\langle T, F \rangle \sqsubseteq \mathbf{\Omega}_P\langle T, F \rangle$

Example

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

- Let's iterate $\mathbf{\Omega}_{P_1}$ on $\langle \{c\}, \emptyset \rangle$:

$$\begin{array}{rcl} \mathbf{\Omega}_P \langle \{c\}, \emptyset \rangle & = & \langle \{a\}, \{d, f\} \rangle \\ \mathbf{\Omega}_P \langle \{a\}, \{d, f\} \rangle & = & \langle \{a, c\}, \{b, e, f\} \rangle \\ \mathbf{\Omega}_P \langle \{a, c\}, \{b, e, f\} \rangle & = & \langle \{a\}, \{b, d, e, f\} \rangle \\ \mathbf{\Omega}_P \langle \{a\}, \{b, d, e, f\} \rangle & = & \langle \{a, c\}, \{b, e, f\} \rangle \\ & \vdots & \end{array}$$

Example

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

- Let's iterate $\mathbf{\Omega}_{P_1}$ on $\langle\{c\},\emptyset\rangle$:

$$\begin{array}{rcl} \mathbf{\Omega}_P\langle\{c\},\emptyset\rangle & = & \langle\{a\},\{d,f\}\rangle \\ \mathbf{\Omega}_P\langle\{a\},\{d,f\}\rangle & = & \langle\{a,c\},\{b,e,f\}\rangle \\ \mathbf{\Omega}_P\langle\{a,c\},\{b,e,f\}\rangle & = & \langle\{a\},\{b,d,e,f\}\rangle \\ \mathbf{\Omega}_P\langle\{a\},\{b,d,e,f\}\rangle & = & \langle\{a,c\},\{b,e,f\}\rangle \\ & \vdots & \end{array}$$

Well-founded semantics

- Define the iterative variant of $\mathbf{\Omega}_P$ analogously to $\mathbf{\Phi}_P$:

$$\mathbf{\Omega}_P^0\langle T, F\rangle = \langle T, F\rangle \qquad \mathbf{\Omega}_P^{i+1}\langle T, F\rangle = \mathbf{\Omega}_P\mathbf{\Omega}_P^i\langle T, F\rangle$$

- Define the well-founded semantics of a normal logic program P as the partial interpretation:

$$\bigsqcup_{i\geq 0}\mathbf{\Omega}_P^i\langle\emptyset,\emptyset\rangle$$

Well-founded semantics

- Define the iterative variant of $\mathbf{\Omega}_P$ analogously to $\mathbf{\Phi}_P$:

$$\mathbf{\Omega}_P^0\langle T, F\rangle = \langle T, F\rangle \qquad \mathbf{\Omega}_P^{i+1}\langle T, F\rangle = \mathbf{\Omega}_P\mathbf{\Omega}_P^i\langle T, F\rangle$$

- Define the well-founded semantics of a normal logic program P as the partial interpretation:

$$\bigsqcup_{i\geq 0}\mathbf{\Omega}_P^i\langle\emptyset, \emptyset\rangle$$

Example

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

$$\begin{array}{lclcl} \mathbf{\Omega}^0 \langle \emptyset, \emptyset \rangle & = & & & \langle \emptyset, \emptyset \rangle \\ \mathbf{\Omega}^1 \langle \emptyset, \emptyset \rangle & = & \mathbf{\Omega} \langle \emptyset, \emptyset \rangle & = & \langle \{a\}, \{f\} \rangle \\ \mathbf{\Omega}^2 \langle \emptyset, \emptyset \rangle & = & \mathbf{\Omega} \langle \{a\}, \{f\} \rangle & = & \langle \{a\}, \{b, e, f\} \rangle \\ \mathbf{\Omega}^3 \langle \emptyset, \emptyset \rangle & = & \mathbf{\Omega} \langle \{a\}, \{b, e, f\} \rangle & = & \langle \{a\}, \{b, e, f\} \rangle \end{array}$$

$$\bigsqcup_{i \geq 0} \mathbf{\Omega}^i \langle \emptyset, \emptyset \rangle = \langle \{a\}, \{b, e, f\} \rangle$$

Example

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

$$\begin{array}{rclcl} \mathbf{\Omega}^0\langle\emptyset,\emptyset\rangle & = & & & \langle\emptyset,\emptyset\rangle \\ \mathbf{\Omega}^1\langle\emptyset,\emptyset\rangle & = & \mathbf{\Omega}\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{f\}\rangle \\ \mathbf{\Omega}^2\langle\emptyset,\emptyset\rangle & = & \mathbf{\Omega}\langle\{a\},\{f\}\rangle & = & \langle\{a\},\{b,e,f\}\rangle \\ \mathbf{\Omega}^3\langle\emptyset,\emptyset\rangle & = & \mathbf{\Omega}\langle\{a\},\{b,e,f\}\rangle & = & \langle\{a\},\{b,e,f\}\rangle \\ & & & & \\ \bigsqcup_{i\geq 0}\mathbf{\Omega}^i\langle\emptyset,\emptyset\rangle & = & \langle\{a\},\{b,e,f\}\rangle & & \end{array}$$

Properties

Let P be a normal logic program

- $\mathbf{\Omega}_P\langle\emptyset,\emptyset\rangle$ is monotonic
  That is, $\mathbf{\Omega}_P^i\langle\emptyset,\emptyset\rangle \sqsubseteq \mathbf{\Omega}_P^{i+1}\langle\emptyset,\emptyset\rangle$
- The well-founded semantics of P is
  - not conflicting,
  - and generally not total
- We have $\bigsqcup_{i\geq 0}\mathbf{\Phi}_P^i\langle\emptyset,\emptyset\rangle \sqsubseteq \bigsqcup_{i\geq 0}\mathbf{\Omega}_P^i\langle\emptyset,\emptyset\rangle$

Well-founded fixpoints

Let P be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation

- Define $\langle T, F \rangle$ as a well-founded fixpoint of P if $\mathbf{\Omega}_P\langle T, F \rangle = \langle T, F \rangle$
  - The well-founded semantics is the $\sqsubseteq$-least well-founded fixpoint of P
  - Any other well-founded fixpoint extends the well-founded semantics
  - Total well-founded fixpoints correspond to stable models

Well-founded fixpoints

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Define $\langle T, F\rangle$ as a well-founded fixpoint of P if $\mathbf{\Omega}_P\langle T, F\rangle = \langle T, F\rangle$
  - The well-founded semantics is the $\sqsubseteq$-least well-founded fixpoint of P
  - Any other well-founded fixpoint extends the well-founded semantics
  - Total well-founded fixpoints correspond to stable models

Well-founded fixpoints: 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 two total well-founded fixpoints:
  - $\langle\{a, c\}, \{b, d, e, f\}\rangle$
  - $\langle\{a, d\}, \{b, c, e, f\}\rangle$
- Both of them represent stable models

Well-founded fixpoints: 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 two total well-founded fixpoints:
  1. $\langle \{a, c\}, \{b, d, e, f\} \rangle$
  2. $\langle \{a, d\}, \{b, c, e, f\} \rangle$
- Both of them represent stable models

Well-founded fixpoints: 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 two total well-founded fixpoints:
  1. $\langle \{a, c\}, \{b, d, e, f\} \rangle$
  2. $\langle \{a, d\}, \{b, c, e, f\} \rangle$
- Both of them represent stable models

Well-founded fixpoints: 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 two total well-founded fixpoints:
  1. $\langle \{a, c\}, \{b, d, e, f\} \rangle$
  2. $\langle \{a, d\}, \{b, c, e, f\} \rangle$
- Both of them represent stable models

Potassco

Properties

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Let $\mathbf{\Omega}_P\langle T, F\rangle = \langle T', F'\rangle$
- If X is a stable model of P such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

  That is, $\mathbf{\Omega}_P$ is stable model preserving

  Hence, $\mathbf{\Omega}_P$ can be used for approximating stable models and so for propagation in ASP-solvers
- In contrast to $\mathbf{\Phi}_P$, operator $\mathbf{\Omega}_P$ is sufficient for propagation because total fixpoints correspond to stable models
- Note In addition to $\mathbf{\Omega}_P$, most ASP-solvers apply backward propagation, originating from program completion (although this is unnecessary from a formal point of view)

Properties

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Let $\mathbf{\Omega}_P\langle T, F\rangle = \langle T', F'\rangle$
- If X is a stable model of P such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

  That is, $\mathbf{\Omega}_P$ is stable model preserving

  Hence, $\mathbf{\Omega}_P$ can be used for approximating stable models and so for propagation in ASP-solvers

- In contrast to $\mathbf{\Phi}_P$, operator $\mathbf{\Omega}_P$ is sufficient for propagation because total fixpoints correspond to stable models

- Note In addition to $\mathbf{\Omega}_P$, most ASP-solvers apply backward propagation, originating from program completion (although this is unnecessary from a formal point of view)

Properties

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Let $\mathbf{\Omega}_P\langle T, F\rangle = \langle T', F'\rangle$
- If X is a stable model of P such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

  That is, $\mathbf{\Omega}_P$ is stable model preserving

  Hence, $\mathbf{\Omega}_P$ can be used for approximating stable models and so for propagation in ASP-solvers

- In contrast to $\mathbf{\Phi}_P$, operator $\mathbf{\Omega}_P$ is sufficient for propagation because total fixpoints correspond to stable models

- Note In addition to $\mathbf{\Omega}_P$, most ASP-solvers apply backward propagation, originating from program completion (although this is unnecessary from a formal point of view)

Properties

Let P be a normal logic program,
and let $\langle T, F \rangle$ be a partial interpretation

- Let $\mathbf{\Omega}_P\langle T, F \rangle = \langle T', F' \rangle$
- If X is a stable model of P such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

  That is, $\mathbf{\Omega}_P$ is stable model preserving

  Hence, $\mathbf{\Omega}_P$ can be used for approximating stable models and so for propagation in ASP-solvers
- In contrast to $\mathbf{\Phi}_P$, operator $\mathbf{\Omega}_P$ is sufficient for propagation because total fixpoints correspond to stable models
- Note In addition to $\mathbf{\Omega}_P$, most ASP-solvers apply backward propagation, originating from program completion (although this is unnecessary from a formal point of view)

Potassco

Properties

Let P be a normal logic program,
and let $\langle T, F\rangle$ be a partial interpretation

- Let $\mathbf{\Omega}_P\langle T, F\rangle = \langle T', F'\rangle$
- If X is a stable model of P such that $T \subseteq X$ and $X \cap F = \emptyset$, then $T' \subseteq X$ and $X \cap F' = \emptyset$

  That is, $\mathbf{\Omega}_P$ is stable model preserving

  Hence, $\mathbf{\Omega}_P$ can be used for approximating stable models and so for propagation in ASP-solvers
- In contrast to $\mathbf{\Phi}_P$, operator $\mathbf{\Omega}_P$ is sufficient for propagation because total fixpoints correspond to stable models
- Note In addition to $\mathbf{\Omega}_P$, most ASP-solvers apply backward propagation, originating from program completion (although this is unnecessary from a formal point of view)

Proof-theoretic Characterization: Overview

8 Tableau Calculi

9 Tableau Calculi for ASP

10 Tableau Calculi characterizing ASP solvers

11 Proof complexity

Outline

8 Tableau Calculi

9 Tableau Calculi for ASP

10 Tableau Calculi characterizing ASP solvers

11 Proof complexity

Motivation

- Goal Analyze computations in ASP solvers
- Wanted A declarative and fine-grained instrument for characterizing operations as well as strategies of ASP solvers
- Idea View stable model computations as derivations in an inference system

  Consider Tableau-based proof systems for analyzing ASP solving

Motivation

- Goal Analyze computations in ASP solvers
- Wanted A declarative and fine-grained instrument for characterizing operations as well as strategies of ASP solvers
- Idea View stable model computations as derivations in an inference system

  Consider Tableau-based proof systems for analyzing ASP solving

Motivation

- Goal Analyze computations in ASP solvers
- Wanted A declarative and fine-grained instrument for characterizing operations as well as strategies of ASP solvers
- Idea View stable model computations as derivations in an inference system

  Consider Tableau-based proof systems for analyzing ASP solving

Tableau calculi

- Traditionally, tableau calculi are used for
  - automated theorem proving and
  - proof theoretical analysis

  in classical as well as non-classical logics
- General idea Given an input, prove some property by decomposition
  Decomposition is done by applying deduction rules
- For details, see *Handbook of Tableau Methods*, Kluwer, 1999

General definitions

- A tableau is a (mostly binary) tree
- A branch in a tableau is a path from the root to a leaf
- A branch containing $\gamma_1, \dots, \gamma_m$ can be extended by applying tableau rules of form

$$\frac{\gamma_1, \dots, \gamma_m}{\begin{array}{c}\alpha_1\\ \vdots\\ \alpha_n\end{array}} \qquad\qquad \frac{\gamma_1, \dots, \gamma_m}{\beta_1 \mid \dots \mid \beta_n}$$

Rules of the former format append entries $\alpha_1, \dots, \alpha_n$ to the branch

Rules of the latter format create multiple sub-branches for $\beta_1, \dots, \beta_n$

General definitions

- A tableau is a (mostly binary) tree
- A branch in a tableau is a path from the root to a leaf
- A branch containing $\gamma_1, \dots, \gamma_m$ can be extended by applying tableau rules of form

$$\frac{\gamma_1, \dots, \gamma_m}{\begin{array}{c}\alpha_1 \\ \vdots \\ \alpha_n\end{array}} \qquad\qquad \frac{\gamma_1, \dots, \gamma_m}{\beta_1 \mid \dots \mid \beta_n}$$

- Rules of the former format append entries $\alpha_1, \dots, \alpha_n$ to the branch
- Rules of the latter format create multiple sub-branches for $\beta_1, \dots, \beta_n$

General definitions

- A tableau is a (mostly binary) tree
- A branch in a tableau is a path from the root to a leaf
- A branch containing $\gamma_1, \dots, \gamma_m$ can be extended by applying tableau rules of form

$$\frac{\gamma_1, \dots, \gamma_m}{\begin{array}{c}\alpha_1 \\ \vdots \\ \alpha_n\end{array}} \qquad\qquad \frac{\gamma_1, \dots, \gamma_m}{\beta_1 \mid \dots \mid \beta_n}$$

- Rules of the former format append entries $\alpha_1, \dots, \alpha_n$ to the branch
- Rules of the latter format create multiple sub-branches for $\beta_1, \dots, \beta_n$

Example

- A simple tableau calculus for proving unsatisfiability of propositional formulas, composed from $\neg$, $\wedge$, and $\vee$, consists of rules

$$\frac{\neg\neg\alpha}{\alpha} \qquad \frac{\alpha_1 \wedge \alpha_2}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta_1 \vee \beta_2}{\beta_1 \mid \beta_2}$$

- All rules are semantically valid, when interpreting entries in a branch conjunctively and distinct (sub-)branches as connected disjunctively
- A propositional formula φ is unsatisfiable iff there is a tableau with φ as the root node such that
  1. all other entries can be produced by tableau rules and
  2. every branch contains some formulas α and $\neg\alpha$

Example

- A simple tableau calculus for proving unsatisfiability of propositional formulas, composed from $\neg$, $\wedge$, and $\vee$, consists of rules

$$\frac{\neg\neg\alpha}{\alpha} \qquad \frac{\alpha_1 \wedge \alpha_2}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta_1 \vee \beta_2}{\beta_1 \mid \beta_2}$$

- All rules are semantically valid, when interpreting entries in a branch conjunctively and distinct (sub-)branches as connected disjunctively
- A propositional formula φ is unsatisfiable iff there is a tableau with φ as the root node such that
  1. all other entries can be produced by tableau rules and
  2. every branch contains some formulas α and $\neg\alpha$

Example

- A simple tableau calculus for proving unsatisfiability of propositional formulas, composed from $\neg$, $\wedge$, and $\vee$, consists of rules

$$\frac{\neg\neg\alpha}{\alpha} \qquad \frac{\alpha_1 \wedge \alpha_2}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta_1 \vee \beta_2}{\beta_1 \mid \beta_2}$$

- All rules are semantically valid, when interpreting entries in a branch conjunctively and distinct (sub-)branches as connected disjunctively
- A propositional formula φ is unsatisfiable iff there is a tableau with φ as the root node such that
  1. all other entries can be produced by tableau rules and
  2. every branch contains some formulas α and $\neg\alpha$

Example

(1)	$a \land ((\neg b \land (\neg a \lor b)) \lor \neg\neg\neg a)$	$[\varphi]$
(2)	a	[1]
(3)	$(\neg b \land (\neg a \lor b)) \lor \neg\neg\neg a$	[1]

(4)	$\neg b \land (\neg a \lor b)$	[3]	**(9)**	$\neg\neg\neg a$	[3]
(5)	$\neg b$	[4]	**(10)**	$\neg a$	[9]
(6)	$\neg a \lor b$	[4]			

(7)	$\neg a$	[6]	**(8)**	b	[6]

All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)

Hence, $a \land ((\neg b \land (\neg a \lor b)) \lor \neg\neg\neg a)$ is unsatisfiable

Example

(1)	$a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$	$[\varphi]$
(2)	a	[1]
(3)	$(\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a$	[1]

(4)	$\neg b \wedge (\neg a \vee b)$	[3]	**(9)**	$\neg\neg\neg a$	[3]
(5)	$\neg b$	[4]	**(10)**	$\neg a$	[9]
(6)	$\neg a \vee b$	[4]			

(7)	$\neg a$	[6]	**(8)**	b	[6]

All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)

Hence, $a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$ is unsatisfiable

Example

(1)	$a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$	$[\varphi]$
(2)	a	[1]
(3)	$(\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a$	[1]

(4)	$\neg b \wedge (\neg a \vee b)$	[3]	**(9)**	$\neg\neg\neg a$	[3]
(5)	$\neg b$	[4]	**(10)**	$\neg a$	[9]
(6)	$\neg a \vee b$	[4]			

(7)	$\neg a$	[6]	**(8)**	b	[6]

All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)

Hence, $a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$ is unsatisfiable

Example

(1)	$a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$	[φ]
(2)	a	[1]
(3)	$(\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a$	[1]

(4)	$\neg b \wedge (\neg a \vee b)$	[3]		(9)	$\neg\neg\neg a$	[3]
(5)	$\neg b$	[4]		(10)	$\neg a$	[9]
(6)	$\neg a \vee b$	[4]				

(7)	$\neg a$	[6]		(8)	b	[6]

All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)

Hence, $a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$ is unsatisfiable

Potassco

Example

(1)	$a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$	$[\varphi]$
(2)	a	[1]
(3)	$(\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a$	[1]

(4)	$\neg b \wedge (\neg a \vee b)$	[3]	**(9)**	$\neg\neg\neg a$	[3]
(5)	$\neg b$	[4]	**(10)**	$\neg a$	[9]
(6)	$\neg a \vee b$	[4]			

(7)	$\neg a$	[6]	**(8)**	b	[6]

- All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)
- Hence, $a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$ is unsatisfiable

Example

(1)	$a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$	$[\varphi]$
(2)	a	[1]
(3)	$(\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a$	[1]

(4)	$\neg b \wedge (\neg a \vee b)$	[3]	**(9)**	$\neg\neg\neg a$	[3]
(5)	$\neg b$	[4]	**(10)**	$\neg a$	[9]
(6)	$\neg a \vee b$	[4]			

(7)	$\neg a$	[6]	**(8)**	b	[6]

- All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)
- Hence, $a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$ is unsatisfiable

Example

(1)	$a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$	$[\varphi]$
(2)	a	[1]
(3)	$(\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a$	[1]

(4)	$\neg b \wedge (\neg a \vee b)$	[3]	**(9)**	$\neg\neg\neg a$	[3]
(5)	$\neg b$	[4]	**(10)**	$\neg a$	[9]
(6)	$\neg a \vee b$	[4]			

(7)	$\neg a$	[6]	**(8)**	b	[6]

- All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)
- Hence, $a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$ is unsatisfiable

Example

(1)	$a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$	$[\varphi]$
(2)	a	[1]
(3)	$(\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a$	[1]

Left branch:

(4)	$\neg b \wedge (\neg a \vee b)$	[3]
(5)	$\neg b$	[4]
(6)	$\neg a \vee b$	[4]

(7)	$\neg a$	[6]	**(8)**	b	[6]

Right branch:

(9)	$\neg\neg\neg a$	[3]
(10)	$\neg a$	[9]

- All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)
- Hence, $a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$ is unsatisfiable

Example

(1)	$a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$	$[\varphi]$
(2)	a	[1]
(3)	$(\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a$	[1]

(4)	$\neg b \wedge (\neg a \vee b)$	[3]	**(9)**	$\neg\neg\neg a$	[3]
(5)	$\neg b$	[4]	**(10)**	$\neg a$	[9]
(6)	$\neg a \vee b$	[4]			

(7)	$\neg a$	[6]	**(8)**	b	[6]

- All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)
- Hence, $a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$ is unsatisfiable

Example

(1)	$a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$	$[\varphi]$
(2)	a	[1]
(3)	$(\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a$	[1]

(4)	$\neg b \wedge (\neg a \vee b)$	[3]	**(9)**	$\neg\neg\neg a$	[3]
(5)	$\neg b$	[4]	**(10)**	$\neg a$	[9]
(6)	$\neg a \vee b$	[4]			

(7)	$\neg a$	[6]	**(8)**	b	[6]

- All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)
- Hence, $a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$ is unsatisfiable

Example

(1)	$a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$	$[\varphi]$
(2)	a	[1]
(3)	$(\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a$	[1]

(4)	$\neg b \wedge (\neg a \vee b)$	[3]	**(9)**	$\neg\neg\neg a$	[3]
(5)	$\neg b$	[4]	**(10)**	$\neg a$	[9]
(6)	$\neg a \vee b$	[4]			

(7)	$\neg a$	[6]	**(8)**	b	[6]

- All three branches of the tableau are contradictory (cf 2, 5, 7, 8, 10)
- Hence, $a \wedge ((\neg b \wedge (\neg a \vee b)) \vee \neg\neg\neg a)$ is unsatisfiable

Outline

8 Tableau Calculi

9 Tableau Calculi for ASP

10 Tableau Calculi characterizing ASP solvers

11 Proof complexity

Tableaux and ASP

- A tableau rule captures an elementary inference scheme in an ASP solver
- A branch in a tableau corresponds to a successful or unsuccessful computation of a stable model
- An entire tableau represents a traversal of the search space

Tableaux and ASP

- A tableau rule captures an elementary inference scheme in an ASP solver
- A branch in a tableau corresponds to a successful or unsuccessful computation of a stable model
- An entire tableau represents a traversal of the search space

Tableaux and ASP

- A tableau rule captures an elementary inference scheme in an ASP solver
- A branch in a tableau corresponds to a successful or unsuccessful computation of a stable model
- An entire tableau represents a traversal of the search space

ASP-specific definitions

- A (signed) tableau for a logic program P is a binary tree such that
  - the root node of the tree consists of the rules in P;
  - the other nodes in the tree are entries of the form $\boldsymbol{T}v$ or $\boldsymbol{F}v$, called signed literals, where v is a variable,
  - generated by extending a tableau using deduction rules (given below)
- An entry $\boldsymbol{T}v$ ($\boldsymbol{F}v$) reflects that variable v is *true* (*false*) in a corresponding variable assignment

  A set of signed literals constitutes a partial assignment
- For a normal logic program P,
  - atoms of P in $atom(P)$ and
  - bodies of P in $body(P)$

  can occur as variables in signed literals

ASP-specific definitions

- A (signed) tableau for a logic program P is a binary tree such that
  - the root node of the tree consists of the rules in P;
  - the other nodes in the tree are entries of the form $\boldsymbol{T}v$ or $\boldsymbol{F}v$, called signed literals, where v is a variable,
  - generated by extending a tableau using deduction rules (given below)
- An entry $\boldsymbol{T}v$ ($\boldsymbol{F}v$) reflects that variable v is *true* (*false*) in a corresponding variable assignment

  A set of signed literals constitutes a partial assignment
- For a normal logic program P,
  - atoms of P in $atom(P)$ and
  - bodies of P in $body(P)$

  can occur as variables in signed literals

ASP-specific definitions

- A (signed) tableau for a logic program P is a binary tree such that
  - the root node of the tree consists of the rules in P;
  - the other nodes in the tree are entries of the form $\boldsymbol{T}v$ or $\boldsymbol{F}v$, called signed literals, where v is a variable,
  - generated by extending a tableau using deduction rules (given below)
- An entry $\boldsymbol{T}v$ ($\boldsymbol{F}v$) reflects that variable v is *true* (*false*) in a corresponding variable assignment

  A set of signed literals constitutes a partial assignment
- For a normal logic program P,
  - atoms of P in $atom(P)$ and
  - bodies of P in $body(P)$

  can occur as variables in signed literals

Potassco

Tableau rules for ASP at a glance

$$\text{(FTB)}\quad \frac{p \leftarrow l_1,\dots,l_n \qquad \boldsymbol{t}l_1,\dots,\boldsymbol{t}l_n}{\boldsymbol{T}\{l_1,\dots,l_n\}}$$

$$\text{(BFB)}\quad \frac{\boldsymbol{F}\{l_1,\dots,l_i,\dots,l_n\} \qquad \boldsymbol{t}l_1,\dots,\boldsymbol{t}l_{i-1},\boldsymbol{t}l_{i+1},\dots,\boldsymbol{t}l_n}{\boldsymbol{f}l_i}$$

$$\text{(FTA)}\quad \frac{p \leftarrow l_1,\dots,l_n \qquad \boldsymbol{T}\{l_1,\dots,l_n\}}{\boldsymbol{T}p}$$

$$\text{(BFA)}\quad \frac{p \leftarrow l_1,\dots,l_n \qquad \boldsymbol{F}p}{\boldsymbol{F}\{l_1,\dots,l_n\}}$$

$$\text{(FFB)}\quad \frac{p \leftarrow l_1,\dots,l_i,\dots,l_n \qquad \boldsymbol{f}l_i}{\boldsymbol{F}\{l_1,\dots,l_i,\dots,l_n\}}$$

$$\text{(BTB)}\quad \frac{\boldsymbol{T}\{l_1,\dots,l_i,\dots,l_n\}}{\boldsymbol{t}l_i}$$

$$\text{(FFA)}\quad \frac{\boldsymbol{F}B_1,\dots,\boldsymbol{F}B_m}{\boldsymbol{F}p}\quad (\S)$$

$$\text{(BTA)}\quad \frac{\boldsymbol{T}p \qquad \boldsymbol{F}B_1,\dots,\boldsymbol{F}B_{i-1},\boldsymbol{F}B_{i+1},\dots,\boldsymbol{F}B_m}{\boldsymbol{T}B_i}\quad (\S)$$

$$\text{(WFN)}\quad \frac{\boldsymbol{F}B_1,\dots,\boldsymbol{F}B_m}{\boldsymbol{F}p}\quad (\dagger)$$

$$\text{(WFJ)}\quad \frac{\boldsymbol{T}p \qquad \boldsymbol{F}B_1,\dots,\boldsymbol{F}B_{i-1},\boldsymbol{F}B_{i+1},\dots,\boldsymbol{F}B_m}{\boldsymbol{T}B_i}\quad (\dagger)$$

$$\text{(FL)}\quad \frac{\boldsymbol{F}B_1,\dots,\boldsymbol{F}B_m}{\boldsymbol{F}p}\quad (\ddagger)$$

$$\text{(BL)}\quad \frac{\boldsymbol{T}p \qquad \boldsymbol{F}B_1,\dots,\boldsymbol{F}B_{i-1},\boldsymbol{F}B_{i+1},\dots,\boldsymbol{F}B_m}{\boldsymbol{T}B_i}\quad (\ddagger)$$

$$\text{(Cut}[X]\text{)}\quad \frac{}{\boldsymbol{T}v \mid \boldsymbol{F}v}\quad (\sharp[X])$$

More concepts

- A tableau calculus is a set of tableau rules
- A branch in a tableau is conflicting,
  if it contains both $\boldsymbol{T}v$ and $\boldsymbol{F}v$ for some variable v
- A branch in a tableau is total for a program P,
  if it contains either $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for each $v \in atom(P) \cup body(P)$
- A branch in a tableau of some calculus $\mathcal{T}$ is closed,
  if no rule in $\mathcal{T}$ other than *Cut* can produce any new entries
- A branch in a tableau is complete,
  if it is either conflicting or both total and closed
- A tableau is complete, if all its branches are complete
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program P,
  if every branch in the tableau is conflicting

More concepts

- A tableau calculus is a set of tableau rules
- A branch in a tableau is conflicting,
  if it contains both $\boldsymbol{T}v$ and $\boldsymbol{F}v$ for some variable v
- A branch in a tableau is total for a program P,
  if it contains either $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for each $v \in atom(P) \cup body(P)$
- A branch in a tableau of some calculus $\mathcal{T}$ is closed,
  if no rule in $\mathcal{T}$ other than *Cut* can produce any new entries
- A branch in a tableau is complete,
  if it is either conflicting or both total and closed
- A tableau is complete, if all its branches are complete
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program P,
  if every branch in the tableau is conflicting

More concepts

- A tableau calculus is a set of tableau rules
- A branch in a tableau is conflicting,
  if it contains both $\boldsymbol{T}v$ and $\boldsymbol{F}v$ for some variable v
- A branch in a tableau is total for a program P,
  if it contains either $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for each $v \in atom(P) \cup body(P)$
- A branch in a tableau of some calculus $\mathcal{T}$ is closed,
  if no rule in $\mathcal{T}$ other than *Cut* can produce any new entries
- A branch in a tableau is complete,
  if it is either conflicting or both total and closed
- A tableau is complete, if all its branches are complete
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program P,
  if every branch in the tableau is conflicting

More concepts

- A tableau calculus is a set of tableau rules
- A branch in a tableau is conflicting,
  if it contains both $\boldsymbol{T}v$ and $\boldsymbol{F}v$ for some variable v
- A branch in a tableau is total for a program P,
  if it contains either $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for each $v \in atom(P) \cup body(P)$
- A branch in a tableau of some calculus $\mathcal{T}$ is closed,
  if no rule in $\mathcal{T}$ other than Cut can produce any new entries
- A branch in a tableau is complete,
  if it is either conflicting or both total and closed
- A tableau is complete, if all its branches are complete
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program P,
  if every branch in the tableau is conflicting

More concepts

- A tableau calculus is a set of tableau rules
- A branch in a tableau is conflicting,
  if it contains both $\boldsymbol{T}v$ and $\boldsymbol{F}v$ for some variable v
- A branch in a tableau is total for a program P,
  if it contains either $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for each $v \in atom(P) \cup body(P)$
- A branch in a tableau of some calculus $\mathcal{T}$ is closed,
  if no rule in $\mathcal{T}$ other than *Cut* can produce any new entries
- A branch in a tableau is complete,
  if it is either conflicting or both total and closed
- A tableau is complete, if all its branches are complete
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program P,
  if every branch in the tableau is conflicting

More concepts

- A tableau calculus is a set of tableau rules
- A branch in a tableau is conflicting,
  if it contains both $\boldsymbol{T}v$ and $\boldsymbol{F}v$ for some variable v
- A branch in a tableau is total for a program P,
  if it contains either $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for each $v \in atom(P) \cup body(P)$
- A branch in a tableau of some calculus $\mathcal{T}$ is closed,
  if no rule in $\mathcal{T}$ other than *Cut* can produce any new entries
- A branch in a tableau is complete,
  if it is either conflicting or both total and closed
- A tableau is complete, if all its branches are complete
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program P,
  if every branch in the tableau is conflicting

More concepts

- A tableau calculus is a set of tableau rules
- A branch in a tableau is conflicting,
  if it contains both $\boldsymbol{T}v$ and $\boldsymbol{F}v$ for some variable v
- A branch in a tableau is total for a program P,
  if it contains either $\boldsymbol{T}v$ or $\boldsymbol{F}v$ for each $v \in atom(P) \cup body(P)$
- A branch in a tableau of some calculus $\mathcal{T}$ is closed,
  if no rule in $\mathcal{T}$ other than *Cut* can produce any new entries
- A branch in a tableau is complete,
  if it is either conflicting or both total and closed
- A tableau is complete, if all its branches are complete
- A tableau of some calculus $\mathcal{T}$ is a refutation of $\mathcal{T}$ for a program P,
  if every branch in the tableau is conflicting

Example

- Consider the program

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

having stable models $\{a, c\}$ and $\{a, d\}$

(Previewed) Example

$$
\begin{array}{c}
a \leftarrow \\
c \leftarrow \sim b, \sim d \\
d \leftarrow a, \sim c
\end{array}
$$

$$
\begin{array}{rccccc}
(\mathrm{FTB}) & & & \boldsymbol{T}\emptyset & & \\
(\mathrm{FTA}) & & & \boldsymbol{T}a & & \\
(\mathrm{FFA}) & & & \boldsymbol{F}b & & \\
(\mathrm{Cut}[atom(P)]) & & \boldsymbol{T}c & & & \boldsymbol{F}c \\
& (\mathrm{BTA}) & \boldsymbol{T}\{\sim b, \sim d\} & & (\mathrm{BFA}) & \boldsymbol{F}\{\sim b, \sim d\} \\
& (\mathrm{BTB}) & \boldsymbol{F}d & & (\mathrm{BFB}) & \boldsymbol{T}d \\
& (\mathrm{FFB}) & \boldsymbol{F}\{a, \sim c\} & & (\mathrm{FTB}) & \boldsymbol{T}\{a, \sim c\}
\end{array}
$$

(Previewed) Example

$$a \leftarrow$$
$$c \leftarrow \sim b, \sim d$$
$$d \leftarrow a, \sim c$$

(FTB)		$\boldsymbol{T}\emptyset$		
(FTA)		$\boldsymbol{T}a$		
(FFA)		$\boldsymbol{F}b$		
(Cut[$atom(P)$])		$\boldsymbol{T}c$		$\boldsymbol{F}c$
	(BTA)	$\boldsymbol{T}\{\sim b, \sim d\}$	(BFA)	$\boldsymbol{F}\{\sim b, \sim d\}$
	(BTB)	$\boldsymbol{F}d$	(BFB)	$\boldsymbol{T}d$
	(FFB)	$\boldsymbol{F}\{a, \sim c\}$	(FTB)	$\boldsymbol{T}\{a, \sim c\}$

Potassco

(Previewed) Example

$$a \leftarrow$$
$$c \leftarrow \sim b, \sim d$$
$$d \leftarrow a, \sim c$$

(FTB)		$\boldsymbol{T}\emptyset$		
(FTA)		$\boldsymbol{T}a$		
(FFA)		$\boldsymbol{F}b$		
(Cut[$atom(P)$])		$\boldsymbol{T}c$		$\boldsymbol{F}c$
	(BTA)	$\boldsymbol{T}\{\sim b, \sim d\}$	(BFA)	$\boldsymbol{F}\{\sim b, \sim d\}$
	(BTB)	$\boldsymbol{F}d$	(BFB)	$\boldsymbol{T}d$
	(FFB)	$\boldsymbol{F}\{a, \sim c\}$	(FTB)	$\boldsymbol{T}\{a, \sim c\}$

(Previewed) Example

$$a \leftarrow$$
$$c \leftarrow \sim b, \sim d$$
$$d \leftarrow a, \sim c$$

(FTB)		$\boldsymbol{T}\emptyset$		
(FTA)		$\boldsymbol{T}a$		
(FFA)		$\boldsymbol{F}b$		
(Cut[$atom(P)$])		$\boldsymbol{T}c$		$\boldsymbol{F}c$
	(BTA)	$\boldsymbol{T}\{\sim b, \sim d\}$	(BFA)	$\boldsymbol{F}\{\sim b, \sim d\}$
	(BTB)	$\boldsymbol{F}d$	(BFB)	$\boldsymbol{T}d$
	(FFB)	$\boldsymbol{F}\{a, \sim c\}$	(FTB)	$\boldsymbol{T}\{a, \sim c\}$

(Previewed) Example

$$
\begin{array}{rcrc}
 & & \multicolumn{2}{l}{a \leftarrow} \\
 & & \multicolumn{2}{l}{c \leftarrow \sim b, \sim d} \\
 & & \multicolumn{2}{l}{d \leftarrow a, \sim c} \\
(FTB) & & \multicolumn{2}{l}{\boldsymbol{T}\emptyset} \\
(FTA) & & \multicolumn{2}{l}{\boldsymbol{T}a} \\
(FFA) & & \multicolumn{2}{l}{\boldsymbol{F}b} \\
(\mathrm{Cut}[atom(P)]) & \boldsymbol{T}c & & \boldsymbol{F}c \\
(BTA) & \boldsymbol{T}\{\sim b, \sim d\} & (BFA) & \boldsymbol{F}\{\sim b, \sim d\} \\
(BTB) & \boldsymbol{F}d & (BFB) & \boldsymbol{T}d \\
(FFB) & \boldsymbol{F}\{a, \sim c\} & (FTB) & \boldsymbol{T}\{a, \sim c\}
\end{array}
$$

(Previewed) Example

$$
\begin{array}{c}
a \leftarrow \\
c \leftarrow \sim b, \sim d \\
d \leftarrow a, \sim c
\end{array}
$$

(FTB)		$\boldsymbol{T}\emptyset$		
(FTA)		$\boldsymbol{T}a$		
(FFA)		$\boldsymbol{F}b$		
(Cut[$atom(P)$])		$\boldsymbol{T}c$		$\boldsymbol{F}c$
	(BTA)	$\boldsymbol{T}\{\sim b, \sim d\}$	(BFA)	$\boldsymbol{F}\{\sim b, \sim d\}$
	(BTB)	$\boldsymbol{F}d$	(BFB)	$\boldsymbol{T}d$
	(FFB)	$\boldsymbol{F}\{a, \sim c\}$	(FTB)	$\boldsymbol{T}\{a, \sim c\}$

Potassco

Auxiliary definitions

- For a literal l, define conjugation functions $\boldsymbol{t}$ and $\boldsymbol{f}$ as follows

$$\boldsymbol{t}l = \begin{cases} \boldsymbol{T}l & \text{if } l \text{ is an atom} \\ \boldsymbol{F}a & \text{if } l = {\sim}a \text{ for an atom } a \end{cases}$$

$$\boldsymbol{f}l = \begin{cases} \boldsymbol{F}l & \text{if } l \text{ is an atom} \\ \boldsymbol{T}a & \text{if } l = {\sim}a \text{ for an atom } a \end{cases}$$

- Examples $\boldsymbol{t}a = \boldsymbol{T}a$, $\boldsymbol{f}a = \boldsymbol{F}a$, $\boldsymbol{t}{\sim}a = \boldsymbol{F}a$, and $\boldsymbol{f}{\sim}a = \boldsymbol{T}a$

Auxiliary definitions

- For a literal l, define conjugation functions $\boldsymbol{t}$ and $\boldsymbol{f}$ as follows

$$\boldsymbol{t}l = \begin{cases} \boldsymbol{T}l & \text{if } l \text{ is an atom} \\ \boldsymbol{F}a & \text{if } l = {\sim}a \text{ for an atom } a \end{cases}$$

$$\boldsymbol{f}l = \begin{cases} \boldsymbol{F}l & \text{if } l \text{ is an atom} \\ \boldsymbol{T}a & \text{if } l = {\sim}a \text{ for an atom } a \end{cases}$$

- Examples $\boldsymbol{t}a = \boldsymbol{T}a$, $\boldsymbol{f}a = \boldsymbol{F}a$, $\boldsymbol{t}{\sim}a = \boldsymbol{F}a$, and $\boldsymbol{f}{\sim}a = \boldsymbol{T}a$

Auxiliary definitions

- Some tableau rules require conditions for their application
- Such conditions are specified as provisos

$$\frac{\mathit{prerequisites}}{\mathit{consequence}}\ (\mathit{proviso})$$

proviso: some condition(s)

- Note All tableau rules given in the sequel are stable model preserving

Forward True Body (FTB)

- Prerequisites All of a body's literals are *true*
- Consequence The body is *true*
- Tableau Rule FTB

$$\frac{\begin{array}{c} p \leftarrow l_1, \dots, l_n \\ \boldsymbol{t}l_1, \dots, \boldsymbol{t}l_n \end{array}}{\boldsymbol{T}\{l_1, \dots, l_n\}}$$

- Example

$$\frac{\begin{array}{c} a \leftarrow b, \sim c \\ \boldsymbol{T}b \\ \boldsymbol{F}c \end{array}}{\boldsymbol{T}\{b, \sim c\}}$$

Forward True Body (FTB)

- Prerequisites All of a body's literals are *true*
- Consequence The body is *true*
- Tableau Rule FTB

$$\frac{\begin{array}{c} p \leftarrow l_1, \ldots, l_n \\ \boldsymbol{t} l_1, \ldots, \boldsymbol{t} l_n \end{array}}{\boldsymbol{T}\{l_1, \ldots, l_n\}}$$

- Example

$$\frac{\begin{array}{c} a \leftarrow b, \sim c \\ \boldsymbol{T} b \\ \boldsymbol{F} c \end{array}}{\boldsymbol{T}\{b, \sim c\}}$$

Backward False Body (BFB)

- Prerequisites A body is *false*, and all its literals except for one are *true*
- Consequence The residual body literal is *false*
- Tableau Rule BFB

$$\frac{\begin{array}{c}\boldsymbol{F}\{l_1,\dots,l_i,\dots,l_n\}\\ \boldsymbol{t}l_1,\dots,\boldsymbol{t}l_{i-1},\boldsymbol{t}l_{i+1},\dots,\boldsymbol{t}l_n\end{array}}{\boldsymbol{f}l_i}$$

- Examples

$$\frac{\begin{array}{c}\boldsymbol{F}\{b,\sim c\}\\ \boldsymbol{T}b\end{array}}{\boldsymbol{T}c} \qquad \frac{\begin{array}{c}\boldsymbol{F}\{b,\sim c\}\\ \boldsymbol{F}c\end{array}}{\boldsymbol{F}b}$$

Backward False Body (BFB)

- Prerequisites A body is *false*, and all its literals except for one are *true*
- Consequence The residual body literal is *false*
- Tableau Rule BFB

$$\frac{\begin{array}{c}\boldsymbol{F}\{l_1,\dots,l_i,\dots,l_n\}\\ \boldsymbol{t}l_1,\dots,\boldsymbol{t}l_{i-1},\boldsymbol{t}l_{i+1},\dots,\boldsymbol{t}l_n\end{array}}{\boldsymbol{f}l_i}$$

- Examples

$$\frac{\begin{array}{c}\boldsymbol{F}\{b,\sim c\}\\ \boldsymbol{T}b\end{array}}{\boldsymbol{T}c} \qquad \frac{\begin{array}{c}\boldsymbol{F}\{b,\sim c\}\\ \boldsymbol{F}c\end{array}}{\boldsymbol{F}b}$$

Forward False Body (FFB)

- Prerequisites Some literal of a body is *false*
- Consequence The body is *false*
- Tableau Rule FFB

$$\frac{p \leftarrow l_1, \ldots, l_i, \ldots, l_n \qquad \boldsymbol{f}\, l_i}{\boldsymbol{F}\{l_1, \ldots, l_i, \ldots, l_n\}}$$

- Examples

$$\frac{a \leftarrow b, \sim c \qquad \boldsymbol{F}\, b}{\boldsymbol{F}\{b, \sim c\}} \qquad \frac{a \leftarrow b, \sim c \qquad \boldsymbol{T}\, c}{\boldsymbol{F}\{b, \sim c\}}$$

Forward False Body (FFB)

- Prerequisites Some literal of a body is *false*
- Consequence The body is *false*
- Tableau Rule FFB

$$\frac{p \leftarrow l_1, \dots, l_i, \dots, l_n \quad \boldsymbol{f} l_i}{\boldsymbol{F}\{l_1, \dots, l_i, \dots, l_n\}}$$

- Examples

$$\frac{a \leftarrow b, \sim c \quad \boldsymbol{F} b}{\boldsymbol{F}\{b, \sim c\}} \qquad \frac{a \leftarrow b, \sim c \quad \boldsymbol{T} c}{\boldsymbol{F}\{b, \sim c\}}$$

Backward True Body (BTB)

- Prerequisites A body is *true*
- Consequence The body's literals are *true*
- Tableau Rule BTB

$$\frac{\boldsymbol{T}\{l_1, \ldots, l_i, \ldots, l_n\}}{\boldsymbol{t}\, l_i}$$

- Examples

$$\frac{\boldsymbol{T}\{b, \sim c\}}{\boldsymbol{T}\, b} \qquad \frac{\boldsymbol{T}\{b, \sim c\}}{\boldsymbol{F}\, c}$$

Backward True Body (BTB)

- Prerequisites A body is *true*
- Consequence The body's literals are *true*
- Tableau Rule BTB

$$\frac{\boldsymbol{T}\{l_1, \dots, l_i, \dots, l_n\}}{\boldsymbol{t}\, l_i}$$

- Examples

$$\frac{\boldsymbol{T}\{b, {\sim}c\}}{\boldsymbol{T}\, b} \qquad \frac{\boldsymbol{T}\{b, {\sim}c\}}{\boldsymbol{F}\, c}$$

Tableau rules for bodies

Consider rule body $B = \{l_1, \dots, l_n\}$

- Rules FTB and BFB amount to implication

$$l_1 \wedge \dots \wedge l_n \rightarrow B$$

- Rules FFB and BTB amount to implication

$$B \rightarrow l_1 \wedge \dots \wedge l_n$$

- Together they yield

$$B \equiv l_1 \wedge \dots \wedge l_n$$

Tableau rules for bodies

Consider rule body $B = \{l_1, \dots, l_n\}$

- Rules FTB and BFB amount to implication

  $$l_1 \wedge \dots \wedge l_n \rightarrow B$$

- Rules FFB and BTB amount to implication

  $$B \rightarrow l_1 \wedge \dots \wedge l_n$$

- Together they yield

  $$B \equiv l_1 \wedge \dots \wedge l_n$$

Potassco

Tableau rules for bodies

Consider rule body $B = \{l_1, \dots, l_n\}$

- Rules FTB and BFB amount to implication

  $$l_1 \wedge \cdots \wedge l_n \rightarrow B$$

- Rules FFB and BTB amount to implication

  $$B \rightarrow l_1 \wedge \cdots \wedge l_n$$

- Together they yield

  $$B \equiv l_1 \wedge \cdots \wedge l_n$$

Forward True Atom (FTA)

- Prerequisites Some of an atom's bodies is *true*
- Consequence The atom is *true*
- Tableau Rule FTA

$$\frac{\begin{array}{c} p \leftarrow l_1, \dots, l_n \\ \boldsymbol{T}\{l_1, \dots, l_n\} \end{array}}{\boldsymbol{T}p}$$

- Examples

$$\frac{\begin{array}{c} a \leftarrow b, \sim c \\ \boldsymbol{T}\{b, \sim c\} \end{array}}{\boldsymbol{T}a} \qquad \frac{\begin{array}{c} a \leftarrow d, \sim e \\ \boldsymbol{T}\{d, \sim e\} \end{array}}{\boldsymbol{T}a}$$

Forward True Atom (FTA)

- Prerequisites Some of an atom's bodies is *true*
- Consequence The atom is *true*
- Tableau Rule FTA

$$\frac{\begin{array}{c} p \leftarrow l_1, \dots, l_n \\ \boldsymbol{T}\{l_1, \dots, l_n\} \end{array}}{\boldsymbol{T}p}$$

- Examples

$$\frac{\begin{array}{c} a \leftarrow b, \sim c \\ \boldsymbol{T}\{b, \sim c\} \end{array}}{\boldsymbol{T}a} \qquad \frac{\begin{array}{c} a \leftarrow d, \sim e \\ \boldsymbol{T}\{d, \sim e\} \end{array}}{\boldsymbol{T}a}$$

Backward False Atom (BFA)

- Prerequisites An atom is *false*
- Consequence The bodies of all rules with the atom as head are *false*
- Tableau Rule BFA

$$\frac{\begin{array}{c} p \leftarrow l_1, \dots, l_n \\ \boldsymbol{F}p \end{array}}{\boldsymbol{F}\{l_1, \dots, l_n\}}$$

- Examples

$$\frac{\begin{array}{c} a \leftarrow b, \sim c \\ \boldsymbol{F}a \end{array}}{\boldsymbol{F}\{b, \sim c\}} \qquad \frac{\begin{array}{c} a \leftarrow d, \sim e \\ \boldsymbol{F}a \end{array}}{\boldsymbol{F}\{d, \sim e\}}$$

Backward False Atom (BFA)

- Prerequisites An atom is *false*
- Consequence The bodies of all rules with the atom as head are *false*
- Tableau Rule BFA

$$\frac{\begin{array}{c} p \leftarrow l_1, \ldots, l_n \\ \boldsymbol{F}p \end{array}}{\boldsymbol{F}\{l_1, \ldots, l_n\}}$$

- Examples

$$\frac{\begin{array}{c} a \leftarrow b, {\sim}c \\ \boldsymbol{F}a \end{array}}{\boldsymbol{F}\{b, {\sim}c\}} \qquad \frac{\begin{array}{c} a \leftarrow d, {\sim}e \\ \boldsymbol{F}a \end{array}}{\boldsymbol{F}\{d, {\sim}e\}}$$

Forward False Atom (FFA)

- Prerequisites For some atom, the bodies of all rules with the atom as head are *false*
- Consequence The atom is *false*
- Tableau Rule FFA

$$\frac{\boldsymbol{F}B_1, \dots, \boldsymbol{F}B_m}{\boldsymbol{F}p} \quad (body_P(p) = \{B_1, \dots, B_m\})$$

- Example

$$\frac{\begin{array}{c}\boldsymbol{F}\{b, \sim c\} \\ \boldsymbol{F}\{d, \sim e\}\end{array}}{\boldsymbol{F}a} \quad (body_P(a) = \{\{b, \sim c\}, \{d, \sim e\}\})$$

Potassco

Forward False Atom (FFA)

- Prerequisites For some atom, the bodies of all rules with the atom as head are *false*
- Consequence The atom is *false*
- Tableau Rule FFA

$$\frac{\boldsymbol{F}B_1, \dots, \boldsymbol{F}B_m}{\boldsymbol{F}p} \; (body_P(p) = \{B_1, \dots, B_m\})$$

- Example

$$\frac{\begin{array}{c}\boldsymbol{F}\{b, \sim c\} \\ \boldsymbol{F}\{d, \sim e\}\end{array}}{\boldsymbol{F}a} \; (body_P(a) = \{\{b, \sim c\}, \{d, \sim e\}\})$$

Backward True Atom (BTA)

- Prerequisites An atom is *true*, and the bodies of all rules with the atom as head except for one are *false*
- Consequence The residual body is *true*
- Tableau Rule BTA

$$\frac{\boldsymbol{T}p \quad \boldsymbol{F}B_1,\dots,\boldsymbol{F}B_{i-1},\boldsymbol{F}B_{i+1},\dots,\boldsymbol{F}B_m}{\boldsymbol{T}B_i}\ (\mathit{body}_P(p)=\{B_1,\dots,B_m\})$$

- Examples

$$\frac{\boldsymbol{T}a \quad \boldsymbol{F}\{b,\sim c\}}{\boldsymbol{T}\{d,\sim e\}}\ (*) \qquad \frac{\boldsymbol{T}a \quad \boldsymbol{F}\{d,\sim e\}}{\boldsymbol{T}\{b,\sim c\}}\ (*)$$

$$(*) \quad \mathit{body}_P(a)=\{\{b,\sim c\},\{d,\sim e\}\}$$

Potassco

Backward True Atom (BTA)

- Prerequisites An atom is *true*, and the bodies of all rules with the atom as head except for one are *false*
- Consequence The residual body is *true*
- Tableau Rule BTA

$$\frac{\begin{array}{c}\boldsymbol{T}p\\ \boldsymbol{F}B_1,\dots,\boldsymbol{F}B_{i-1},\boldsymbol{F}B_{i+1},\dots,\boldsymbol{F}B_m\end{array}}{\boldsymbol{T}B_i}\quad(\mathit{body}_P(p)=\{B_1,\dots,B_m\})$$

- Examples

$$\frac{\begin{array}{c}\boldsymbol{T}a\\ \boldsymbol{F}\{b,\sim c\}\end{array}}{\boldsymbol{T}\{d,\sim e\}}\ (*)\qquad\qquad\frac{\begin{array}{c}\boldsymbol{T}a\\ \boldsymbol{F}\{d,\sim e\}\end{array}}{\boldsymbol{T}\{b,\sim c\}}\ (*)$$

$$(*)\quad \mathit{body}_P(a)=\{\{b,\sim c\},\{d,\sim e\}\}$$

Tableau rules for atoms

Consider an atom p such that $body_P(p) = \{B_1, \dots, B_m\}$

- Rules FTA and BFA amount to implication

$$B_1 \vee \dots \vee B_m \rightarrow p$$

- Rules FFA and BTA amount to implication

$$p \rightarrow B_1 \vee \dots \vee B_m$$

- Together they yield

$$p \equiv B_1 \vee \dots \vee B_m$$

Tableau rules for atoms

Consider an atom p such that $body_P(p) = \{B_1, \ldots, B_m\}$

- Rules FTA and BFA amount to implication

  $$B_1 \vee \cdots \vee B_m \rightarrow p$$

- Rules FFA and BTA amount to implication

  $$p \rightarrow B_1 \vee \cdots \vee B_m$$

- Together they yield

  $$p \equiv B_1 \vee \cdots \vee B_m$$

Tableau rules for atoms

Consider an atom p such that $body_P(p) = \{B_1, \ldots, B_m\}$

- Rules FTA and BFA amount to implication

  $$B_1 \vee \cdots \vee B_m \rightarrow p$$

- Rules FFA and BTA amount to implication

  $$p \rightarrow B_1 \vee \cdots \vee B_m$$

- Together they yield

  $$p \equiv B_1 \vee \cdots \vee B_m$$

Relationship with program completion

Let P be a normal logic program

- The eight tableau rules introduced so far essentially provide (straightforward) inferences from $CF(P)$

Preliminaries for unfounded sets

Let P be a normal logic program

- For $P' \subseteq P$, define the greatest unfounded set of P wrt P' as

$$\mathbf{U}_P(P') = atom(P) \setminus Cn((P')^{\emptyset})$$

- For a loop $L \in loop(P)$, define the external bodies of L as

$$EB_P(L) = \{body(r) \mid r \in P, head(r) \in L, body(r)^+ \cap L = \emptyset\}$$

Potassco

Preliminaries for unfounded sets

Let P be a normal logic program

- For $P' \subseteq P$, define the greatest unfounded set of P wrt P' as

$$\mathbf{U}_P(P') = atom(P) \setminus Cn((P')^{\emptyset})$$

- For a loop $L \in loop(P)$, define the external bodies of L as

$$EB_P(L) = \{body(r) \mid r \in P, head(r) \in L, body(r)^+ \cap L = \emptyset\}$$

Well-Founded Negation (WFN)

- Prerequisites An atom is in the greatest unfounded set wrt rules whose bodies are *false*
- Consequence The atom is *false*
- Tableau Rule WFN

$$\frac{\boldsymbol{F}B_1,\dots,\boldsymbol{F}B_m}{\boldsymbol{F}p} \quad (p \in \mathbf{U}_P(\{r \in P \mid body(r) \notin \{B_1,\dots,B_m\}\}))$$

- Examples

$$\frac{\begin{array}{c} a \leftarrow \sim b \\ \boldsymbol{F}\{\sim b\} \end{array}}{\boldsymbol{F}a} \ (*) \qquad \frac{\begin{array}{c} a \leftarrow a \\ a \leftarrow \sim b \\ \boldsymbol{F}\{\sim b\} \end{array}}{\boldsymbol{F}a} \ (*)$$

$$(*) \quad a \in \mathbf{U}_P(P \setminus \{a \leftarrow \sim b\})$$

Well-Founded Negation (WFN)

- Prerequisites An atom is in the greatest unfounded set wrt rules whose bodies are *false*
- Consequence The atom is *false*
- Tableau Rule WFN

$$\frac{\boldsymbol{F}B_1,\ldots,\boldsymbol{F}B_m}{\boldsymbol{F}p}\quad (p \in \mathbf{U}_P(\{r \in P \mid body(r) \notin \{B_1,\ldots,B_m\}\}))$$

- Examples

$$\frac{\begin{array}{c}a \leftarrow \sim b\\ \boldsymbol{F}\{\sim b\}\end{array}}{\boldsymbol{F}a}\ (*) \qquad \frac{\begin{array}{c}a \leftarrow a\\ a \leftarrow \sim b\\ \boldsymbol{F}\{\sim b\}\end{array}}{\boldsymbol{F}a}\ (*)$$

$$(*) \quad a \in \mathbf{U}_P(P \setminus \{a \leftarrow \sim b\})$$

Well-Founded Justification (WFJ)

- Prerequisites A *true* atom is in the greatest unfounded set wrt rules whose bodies are *false*, if a particular body is made *false*
- Consequence The respective body is *true*
- Tableau Rule WFJ

$$\frac{\begin{array}{c}\boldsymbol{T}p\\ \boldsymbol{F}B_1,\dots,\boldsymbol{F}B_{i-1},\boldsymbol{F}B_{i+1},\dots,\boldsymbol{F}B_m\end{array}}{\boldsymbol{T}B_i}\quad(p\in\mathbf{U}_P(\{r\in P\mid body(r)\notin\{B_1,\dots,B_m\}\}))$$

- Examples

$$\frac{\begin{array}{c}a\leftarrow\sim b\\ \boldsymbol{T}a\end{array}}{\boldsymbol{T}\{\sim b\}}\ (*)\qquad\qquad\frac{\begin{array}{c}a\leftarrow a\\ a\leftarrow\sim b\\ \boldsymbol{T}a\end{array}}{\boldsymbol{T}\{\sim b\}}\ (*)$$

$$(*)\quad a\in\mathbf{U}_P(P\setminus\{a\leftarrow\sim b\})$$

Well-Founded Justification (WFJ)

- Prerequisites A *true* atom is in the greatest unfounded set wrt rules whose bodies are *false*, if a particular body is made *false*
- Consequence The respective body is *true*
- Tableau Rule WFJ

$$\frac{\begin{array}{c}\boldsymbol{T}p\\ \boldsymbol{F}B_1,\dots,\boldsymbol{F}B_{i-1},\boldsymbol{F}B_{i+1},\dots,\boldsymbol{F}B_m\end{array}}{\boldsymbol{T}B_i}\quad (p \in \mathbf{U}_P(\{r \in P \mid body(r) \notin \{B_1,\dots,B_m\}\}))$$

- Examples

$$\frac{\begin{array}{c}a \leftarrow \sim b\\ \boldsymbol{T}a\end{array}}{\boldsymbol{T}\{\sim b\}}\ (*) \qquad \frac{\begin{array}{c}a \leftarrow a\\ a \leftarrow \sim b\\ \boldsymbol{T}a\end{array}}{\boldsymbol{T}\{\sim b\}}\ (*)$$

$$(*) \quad a \in \mathbf{U}_P(P \setminus \{a \leftarrow \sim b\})$$

Well-founded tableau rules

- Tableau rules WFN and WFJ ensure non-circular support for *true* atoms
- Note
  1. WFN subsumes falsifying atoms via FFA,
  2. WFJ can be viewed as "backward propagation" for unfounded sets,
  3. WFJ subsumes backward propagation of *true* atoms via BTA

Well-founded tableau rules

- Tableau rules WFN and WFJ ensure non-circular support for *true* atoms
- Note
  1. WFN subsumes falsifying atoms via FFA,
  2. WFJ can be viewed as "backward propagation" for unfounded sets,
  3. WFJ subsumes backward propagation of *true* atoms via BTA

Relationship with well-founded operator

Let P be a normal logic program, $\langle T, F \rangle$ a partial interpretation, and $P' = \{r \in P \mid body(r)^+ \cap F = \emptyset \text{ and } body(r)^- \cap T = \emptyset\}$.

- The following conditions are equivalent
  1. $p \in \mathbf{U}_P\langle T, F \rangle$
  2. $p \in \mathbf{U}_P(P')$
- Hence, the well-founded operator $\mathbf{\Omega}$ and WFN coincide
- Note In contrast to $\mathbf{\Omega}$, WFN does not necessarily require a rule body to contain a *false* literal for the rule being inapplicable

Potassco

Relationship with well-founded operator

Let P be a normal logic program, $\langle T, F\rangle$ a partial interpretation, and $P' = \{r \in P \mid body(r)^+ \cap F = \emptyset \text{ and } body(r)^- \cap T = \emptyset\}$.

- The following conditions are equivalent
  1. $p \in \mathbf{U}_P\langle T, F\rangle$
  2. $p \in \mathbf{U}_P(P')$
- Hence, the well-founded operator $\mathbf{\Omega}$ and WFN coincide
- Note In contrast to $\mathbf{\Omega}$, WFN does not necessarily require a rule body to contain a *false* literal for the rule being inapplicable

Relationship with well-founded operator

Let P be a normal logic program, $\langle T, F\rangle$ a partial interpretation, and $P' = \{r \in P \mid body(r)^+ \cap F = \emptyset \text{ and } body(r)^- \cap T = \emptyset\}$.

- The following conditions are equivalent
  1. $p \in \mathbf{U}_P\langle T, F\rangle$
  2. $p \in \mathbf{U}_P(P')$
- Hence, the well-founded operator $\mathbf{\Omega}$ and WFN coincide
- Note In contrast to $\mathbf{\Omega}$, WFN does not necessarily require a rule body to contain a *false* literal for the rule being inapplicable

Relationship with well-founded operator

Let P be a normal logic program, $\langle T, F\rangle$ a partial interpretation, and $P' = \{r \in P \mid body(r)^+ \cap F = \emptyset \text{ and } body(r)^- \cap T = \emptyset\}$.

- The following conditions are equivalent
  1. $p \in \mathbf{U}_P\langle T, F\rangle$
  2. $p \in \mathbf{U}_P(P')$
- Hence, the well-founded operator $\mathbf{\Omega}$ and WFN coincide
- Note In contrast to $\mathbf{\Omega}$, WFN does not necessarily require a rule body to contain a *false* literal for the rule being inapplicable

Forward Loop (FL)

- Prerequisites The external bodies of a loop are *false*
- Consequence The atoms in the loop are *false*
- Tableau Rule FL

$$\frac{\boldsymbol{F}B_1, \dots, \boldsymbol{F}B_m}{\boldsymbol{F}p} \; (p \in L, L \in loop(P), EB_P(L) = \{B_1, \dots, B_m\})$$

- Example

$$a \leftarrow a$$
$$a \leftarrow \sim b$$
$$\frac{\boldsymbol{F}\{\sim b\}}{\boldsymbol{F}a} \; (EB_P(\{a\}) = \{\{\sim b\}\})$$

Forward Loop (FL)

- Prerequisites The external bodies of a loop are *false*
- Consequence The atoms in the loop are *false*
- Tableau Rule FL

$$\frac{\boldsymbol{F}B_1,\ldots,\boldsymbol{F}B_m}{\boldsymbol{F}p}\ (p\in L, L\in loop(P), EB_P(L)=\{B_1,\ldots,B_m\})$$

- Example

$$\begin{array}{c} a\leftarrow a \\ a\leftarrow \sim b \end{array}$$

$$\frac{\boldsymbol{F}\{\sim b\}}{\boldsymbol{F}a}\ (EB_P(\{a\})=\{\{\sim b\}\})$$

Backward Loop (BL)

- Prerequisites An atom of a loop is *true*, and all external bodies except for one are *false*
- Consequence The residual external body is *true*
- Tableau Rule BL

$$\frac{\begin{array}{c}\boldsymbol{T}p \\ \boldsymbol{F}B_1,\dots,\boldsymbol{F}B_{i-1},\boldsymbol{F}B_{i+1},\dots,\boldsymbol{F}B_m\end{array}}{\boldsymbol{T}B_i}\ (p \in L, L \in loop(P), EB_P(L) = \{B_1,\dots,B_m\})$$

- Example

$$a \leftarrow a$$
$$a \leftarrow {\sim}b$$
$$\frac{\boldsymbol{T}a}{\boldsymbol{T}\{{\sim}b\}}\ (EB_P(\{a\}) = \{\{{\sim}b\}\})$$

Backward Loop (BL)

- Prerequisites An atom of a loop is *true*, and all external bodies except for one are *false*
- Consequence The residual external body is *true*
- Tableau Rule BL

$$\frac{\begin{array}{c}\boldsymbol{T}p \\ \boldsymbol{F}B_1,\dots,\boldsymbol{F}B_{i-1},\boldsymbol{F}B_{i+1},\dots,\boldsymbol{F}B_m\end{array}}{\boldsymbol{T}B_i}\ (p \in L, L \in loop(P), EB_P(L) = \{B_1,\dots,B_m\})$$

- Example

$$\begin{array}{c} a \leftarrow a \\ a \leftarrow \sim b \\ \hline \end{array}$$

$$\frac{\boldsymbol{T}a}{\boldsymbol{T}\{\sim b\}}\ (EB_P(\{a\}) = \{\{\sim b\}\})$$

Tableau rules for loops

- Tableau rules FL and BL ensure non-circular support for *true* atoms
- For a loop L such that $EB_P(L) = \{B_1, \dots, B_m\}$, they amount to implications of form

$$\bigvee_{p \in L} p \rightarrow B_1 \vee \dots \vee B_m$$

- Comparison to well-founded tableau rules yields
  - FL (plus FFA and FFB) is equivalent to WFN (plus FFB),
  - BL cannot simulate inferences via WFJ

Tableau rules for loops

- Tableau rules FL and BL ensure non-circular support for *true* atoms
- For a loop L such that $EB_P(L) = \{B_1, \dots, B_m\}$,
  they amount to implications of form

$$\bigvee_{p \in L} p \rightarrow B_1 \vee \cdots \vee B_m$$

- Comparison to well-founded tableau rules yields
  - FL (plus FFA and FFB) is equivalent to WFN (plus FFB),
  - BL cannot simulate inferences via WFJ

Potassco

Tableau rules for loops

- Tableau rules FL and BL ensure non-circular support for *true* atoms
- For a loop L such that $EB_P(L) = \{B_1, \ldots, B_m\}$, they amount to implications of form

$$\bigvee_{p \in L} p \rightarrow B_1 \vee \cdots \vee B_m$$

- Comparison to well-founded tableau rules yields
  - FL (plus FFA and FFB) is equivalent to WFN (plus FFB),
  - BL cannot simulate inferences via WFJ

Relationship with loop formulas

- Tableau rules FL and BL essentially provide (straightforward) inferences from loop formulas
  - Impractical to precompute exponentially many loop formulas
- In practice, ASP solvers such as *smodels* and *clasp*
  - exploit strongly connected components of positive atom dependency graphs
    - can be viewed as an interpolation of FL
  - do not directly implement BL (and neither WFJ)
    - probably difficult to do efficiently
  - could simulate BL via FL/WFN by means of failed-literal detection (lookahead)

Relationship with loop formulas

- Tableau rules FL and BL essentially provide (straightforward) inferences from loop formulas
  - Impractical to precompute exponentially many loop formulas
- In practice, ASP solvers such as *smodels* and *clasp*
  - exploit strongly connected components of positive atom dependency graphs
    - can be viewed as an interpolation of FL
  - do not directly implement BL (and neither WFJ)
    - probably difficult to do efficiently
  - could simulate BL via FL/WFN by means of failed-literal detection (lookahead)

Relationship with loop formulas

- Tableau rules FL and BL essentially provide (straightforward) inferences from loop formulas
  - Impractical to precompute exponentially many loop formulas
- In practice, ASP solvers such as *smodels* and *clasp*
  - exploit strongly connected components of positive atom dependency graphs
    - can be viewed as an interpolation of FL
  - do not directly implement BL (and neither WFJ)
    - probably difficult to do efficiently
  - could simulate BL via FL/WFN by means of failed-literal detection (lookahead)

Relationship with loop formulas

- Tableau rules FL and BL essentially provide (straightforward) inferences from loop formulas
  - Impractical to precompute exponentially many loop formulas
- In practice, ASP solvers such as *smodels* and *clasp*
  - exploit strongly connected components of positive atom dependency graphs
    - can be viewed as an interpolation of FL
  - do not directly implement BL (and neither WFJ)
    - probably difficult to do efficiently
  - could simulate BL via FL/WFN by means of failed-literal detection (lookahead)

Case analysis by *Cut*

- Up to now, all tableau rules are deterministic

  That is, rules extend a single branch but cannot create sub-branches
- In general, closing a branch leads to a partial assignment
- Case analysis is done by $Cut[\mathcal{C}]$ where $\mathcal{C} \subseteq atom(P) \cup body(P)$

Case analysis by *Cut*

- Up to now, all tableau rules are deterministic

  That is, rules extend a single branch but cannot create sub-branches
- In general, closing a branch leads to a partial assignment
- Case analysis is done by $Cut[\mathcal{C}]$ where $\mathcal{C} \subseteq atom(P) \cup body(P)$

Case analysis by *Cut*

- Up to now, all tableau rules are deterministic
  That is, rules extend a single branch but cannot create sub-branches
- In general, closing a branch leads to a partial assignment
- Case analysis is done by $Cut[\mathcal{C}]$ where $\mathcal{C} \subseteq atom(P) \cup body(P)$

Case analysis by *Cut*

- Prerequisites None
- Consequence Two alternative (complementary) entries
- Tableau Rule $Cut[\mathcal{C}]$

$$\frac{}{\boldsymbol{T}v \mid \boldsymbol{F}v}\ (v \in \mathcal{C})$$

- Examples

$$\frac{\begin{array}{c} a \leftarrow \sim b \\ b \leftarrow \sim a \end{array}}{\boldsymbol{T}a \mid \boldsymbol{F}a}\ (\mathcal{C} = atom(P))$$

$$\frac{\begin{array}{c} a \leftarrow \sim b \\ b \leftarrow \sim a \end{array}}{\boldsymbol{T}\{\sim b\} \mid \boldsymbol{F}\{\sim b\}}\ (\mathcal{C} = body(P))$$

Potassco

Case analysis by *Cut*

- Prerequisites None
- Consequence Two alternative (complementary) entries
- Tableau Rule $\mathit{Cut}[\mathcal{C}]$

$$\frac{}{\boldsymbol{T}v \quad \mid \quad \boldsymbol{F}v} \; (v \in \mathcal{C})$$

- Examples

$$\frac{\begin{array}{c} a \leftarrow {\sim}b \\ b \leftarrow {\sim}a \end{array}}{\boldsymbol{T}a \quad \mid \quad \boldsymbol{F}a} \; (\mathcal{C} = \mathit{atom}(P))$$

$$\frac{\begin{array}{c} a \leftarrow {\sim}b \\ b \leftarrow {\sim}a \end{array}}{\boldsymbol{T}\{{\sim}b\} \quad \mid \quad \boldsymbol{F}\{{\sim}b\}} \; (\mathcal{C} = \mathit{body}(P))$$

Well-known tableau calculi

- Fitting's operator $\mathbf{\Phi}$ applies forward propagation without sophisticated unfounded set checks

$$\mathcal{T}_{\mathbf{\Phi}} = \{FTB, FTA, FFB, FFA\}$$

- Well-founded operator $\mathbf{\Omega}$ replaces negation of single atoms with negation of unfounded sets

$$\mathcal{T}_{\mathbf{\Omega}} = \{FTB, FTA, FFB, WFN\}$$

- "Local" propagation via a program's completion can be determined by elementary inferences on atoms and rule bodies

$$\mathcal{T}_{completion} = \{FTB, FTA, FFB, FFA, BTB, BTA, BFB, BFA\}$$

Well-known tableau calculi

- Fitting's operator $\mathbf{\Phi}$ applies forward propagation without sophisticated unfounded set checks

$$\mathcal{T}_{\mathbf{\Phi}} = \{FTB, FTA, FFB, FFA\}$$

- Well-founded operator $\mathbf{\Omega}$ replaces negation of single atoms with negation of unfounded sets

$$\mathcal{T}_{\mathbf{\Omega}} = \{FTB, FTA, FFB, WFN\}$$

- "Local" propagation via a program's completion can be determined by elementary inferences on atoms and rule bodies

$$\mathcal{T}_{completion} = \{FTB, FTA, FFB, FFA, BTB, BTA, BFB, BFA\}$$

Well-known tableau calculi

- Fitting's operator $\mathbf{\Phi}$ applies forward propagation without sophisticated unfounded set checks

$$\mathcal{T}_{\mathbf{\Phi}} = \{FTB, FTA, FFB, FFA\}$$

- Well-founded operator $\mathbf{\Omega}$ replaces negation of single atoms with negation of unfounded sets

$$\mathcal{T}_{\mathbf{\Omega}} = \{FTB, FTA, FFB, WFN\}$$

- "Local" propagation via a program's completion can be determined by elementary inferences on atoms and rule bodies

$$\mathcal{T}_{completion} = \{FTB, FTA, FFB, FFA, BTB, BTA, BFB, BFA\}$$

Outline

8 Tableau Calculi

9 Tableau Calculi for ASP

10 Tableau Calculi characterizing ASP solvers

11 Proof complexity

Tableau calculi characterizing ASP solvers

- ASP solvers combine propagation with case analysis
- We obtain the following tableau calculi characterizing

$$
\begin{aligned}
\mathcal{T}_{\mathit{cmodels\text{-}1}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{Cut}[\mathit{atom}(P) \cup \mathit{body}(P)]\} \\
\mathcal{T}_{\mathit{assat}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{FL}\} \cup \{\mathit{Cut}[\mathit{atom}(P) \cup \mathit{body}(P)]\} \\
\mathcal{T}_{\mathit{smodels}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{WFN}\} \cup \{\mathit{Cut}[\mathit{atom}(P)]\} \\
\mathcal{T}_{\mathit{noMoRe}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{WFN}\} \cup \{\mathit{Cut}[\mathit{body}(P)]\} \\
\mathcal{T}_{\mathit{nomore}^{++}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{WFN}\} \cup \{\mathit{Cut}[\mathit{atom}(P) \cup \mathit{body}(P)]\}
\end{aligned}
$$

- SAT-based ASP solvers, *assat* and *cmodels*, incrementally add loop formulas to a program's completion
- Native ASP solvers, *smodels*, *dlv*, *noMoRe*, and *nomore++*, essentially differ only in their *Cut* rules

Tableau calculi characterizing ASP solvers

- ASP solvers combine propagation with case analysis
- We obtain the following tableau calculi characterizing

$$
\begin{aligned}
\mathcal{T}_{\mathit{cmodels\text{-}1}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{Cut}[\mathit{atom}(P) \cup \mathit{body}(P)]\} \\
\mathcal{T}_{\mathit{assat}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{FL}\} \cup \{\mathit{Cut}[\mathit{atom}(P) \cup \mathit{body}(P)]\} \\
\mathcal{T}_{\mathit{smodels}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{WFN}\} \cup \{\mathit{Cut}[\mathit{atom}(P)]\} \\
\mathcal{T}_{\mathit{noMoRe}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{WFN}\} \cup \{\mathit{Cut}[\mathit{body}(P)]\} \\
\mathcal{T}_{\mathit{nomore}^{++}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{WFN}\} \cup \{\mathit{Cut}[\mathit{atom}(P) \cup \mathit{body}(P)]\}
\end{aligned}
$$

- SAT-based ASP solvers, *assat* and *cmodels*, incrementally add loop formulas to a program's completion
- Native ASP solvers, *smodels*, *dlv*, *noMoRe*, and *nomore++*, essentially differ only in their *Cut* rules

Tableau calculi characterizing ASP solvers

- ASP solvers combine propagation with case analysis
- We obtain the following tableau calculi characterizing

$$\begin{aligned}
\mathcal{T}_{\mathit{cmodels\text{-}1}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{Cut}[\mathit{atom}(P) \cup \mathit{body}(P)]\} \\
\mathcal{T}_{\mathit{assat}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{FL}\} \cup \{\mathit{Cut}[\mathit{atom}(P) \cup \mathit{body}(P)]\} \\
\mathcal{T}_{\mathit{smodels}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{WFN}\} \cup \{\mathit{Cut}[\mathit{atom}(P)]\} \\
\mathcal{T}_{\mathit{noMoRe}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{WFN}\} \cup \{\mathit{Cut}[\mathit{body}(P)]\} \\
\mathcal{T}_{\mathit{nomore}^{++}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{WFN}\} \cup \{\mathit{Cut}[\mathit{atom}(P) \cup \mathit{body}(P)]\}
\end{aligned}$$

- SAT-based ASP solvers, *assat* and *cmodels*, incrementally add loop formulas to a program's completion
- Native ASP solvers, *smodels*, *dlv*, *noMoRe*, and *nomore++*, essentially differ only in their *Cut* rules

Tableau calculi characterizing ASP solvers

- ASP solvers combine propagation with case analysis
- We obtain the following tableau calculi characterizing

$$
\begin{aligned}
\mathcal{T}_{\mathit{cmodels\text{-}1}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{Cut}[\mathit{atom}(P) \cup \mathit{body}(P)]\} \\
\mathcal{T}_{\mathit{assat}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{FL}\} \cup \{\mathit{Cut}[\mathit{atom}(P) \cup \mathit{body}(P)]\} \\
\mathcal{T}_{\mathit{smodels}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{WFN}\} \cup \{\mathit{Cut}[\mathit{atom}(P)]\} \\
\mathcal{T}_{\mathit{noMoRe}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{WFN}\} \cup \{\mathit{Cut}[\mathit{body}(P)]\} \\
\mathcal{T}_{\mathit{nomore}^{++}} &= \mathcal{T}_{\mathit{completion}} \cup \{\mathit{WFN}\} \cup \{\mathit{Cut}[\mathit{atom}(P) \cup \mathit{body}(P)]\}
\end{aligned}
$$

- SAT-based ASP solvers, *assat* and *cmodels*, incrementally add loop formulas to a program's completion
- Native ASP solvers, *smodels*, *dlv*, *noMoRe*, and *nomore++*, essentially differ only in their *Cut* rules

Outline

8 Tableau Calculi

9 Tableau Calculi for ASP

10 Tableau Calculi characterizing ASP solvers

11 Proof complexity

Potassco

Proof complexity

- Proof complexity is used for describing the relative efficiency of different proof systems
  - It compares proof systems based on minimal refutations
  - It is independent of heuristics
- A proof system $\mathcal{T}$ polynomially simulates a proof system $\mathcal{T}'$, if every refutation of $\mathcal{T}'$ can be polynomially mapped to a refutation of $\mathcal{T}$
  - Otherwise, $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$
- For showing that proof system $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$, we have to provide an infinite witnessing family of programs such that minimal refutations of $\mathcal{T}$ asymptotically are exponentially larger than minimal refutations of $\mathcal{T}'$
  - The size of tableaux is simply the number of their entries
- We do not need to know the precise number of entries: Counting required *Cut* applications is sufficient !

Proof complexity

- Proof complexity is used for describing the relative efficiency of different proof systems

  It compares proof systems based on minimal refutations

  It is independent of heuristics
- A proof system $\mathcal{T}$ polynomially simulates a proof system $\mathcal{T}'$, if every refutation of $\mathcal{T}'$ can be polynomially mapped to a refutation of $\mathcal{T}$

  Otherwise, $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$
- For showing that proof system $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$, we have to provide an infinite witnessing family of programs such that minimal refutations of $\mathcal{T}$ asymptotically are exponentially larger than minimal refutations of $\mathcal{T}'$

  The size of tableaux is simply the number of their entries
- We do not need to know the precise number of entries: Counting required *Cut* applications is sufficient !

Proof complexity

- Proof complexity is used for describing the relative efficiency of different proof systems
  - It compares proof systems based on minimal refutations
  - It is independent of heuristics
- A proof system $\mathcal{T}$ polynomially simulates a proof system $\mathcal{T}'$, if every refutation of $\mathcal{T}'$ can be polynomially mapped to a refutation of $\mathcal{T}$
  - Otherwise, $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$
- For showing that proof system $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$, we have to provide an infinite witnessing family of programs such that minimal refutations of $\mathcal{T}$ asymptotically are exponentially larger than minimal refutations of $\mathcal{T}'$
  - The size of tableaux is simply the number of their entries
- We do not need to know the precise number of entries: Counting required *Cut* applications is sufficient !

Proof complexity

- Proof complexity is used for describing the relative efficiency of different proof systems

  It compares proof systems based on minimal refutations

  It is independent of heuristics
- A proof system $\mathcal{T}$ polynomially simulates a proof system $\mathcal{T}'$, if every refutation of $\mathcal{T}'$ can be polynomially mapped to a refutation of $\mathcal{T}$

  Otherwise, $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$
- For showing that proof system $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$, we have to provide an infinite witnessing family of programs such that minimal refutations of $\mathcal{T}$ asymptotically are exponentially larger than minimal refutations of $\mathcal{T}'$

  The size of tableaux is simply the number of their entries
- We do not need to know the precise number of entries: Counting required *Cut* applications is sufficient !

Potassco

Proof complexity

- Proof complexity is used for describing the relative efficiency of different proof systems

  It compares proof systems based on minimal refutations

  It is independent of heuristics
- A proof system $\mathcal{T}$ polynomially simulates a proof system $\mathcal{T}'$, if every refutation of $\mathcal{T}'$ can be polynomially mapped to a refutation of $\mathcal{T}$

  Otherwise, $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$
- For showing that proof system $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$, we have to provide an infinite witnessing family of programs such that minimal refutations of $\mathcal{T}$ asymptotically are exponentially larger than minimal refutations of $\mathcal{T}'$

  The size of tableaux is simply the number of their entries
- We do not need to know the precise number of entries: Counting required *Cut* applications is sufficient !

Proof complexity

- Proof complexity is used for describing the relative efficiency of different proof systems
  - It compares proof systems based on minimal refutations
  - It is independent of heuristics
- A proof system $\mathcal{T}$ polynomially simulates a proof system $\mathcal{T}'$, if every refutation of $\mathcal{T}'$ can be polynomially mapped to a refutation of $\mathcal{T}$
  - Otherwise, $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$
- For showing that proof system $\mathcal{T}$ does not polynomially simulate $\mathcal{T}'$, we have to provide an infinite witnessing family of programs such that minimal refutations of $\mathcal{T}$ asymptotically are exponentially larger than minimal refutations of $\mathcal{T}'$
  - The size of tableaux is simply the number of their entries
- We do not need to know the precise number of entries: Counting required *Cut* applications is sufficient !

$\mathcal{T}_{smodels}$ versus $\mathcal{T}_{noMoRe}$

- $\mathcal{T}_{smodels}$ restricts Cut to $atom(P)$ and $\mathcal{T}_{noMoRe}$ to $body(P)$
  Are both approaches similar or is one of them superior to the other?
- Let $\{P_a^n\}$, $\{P_b^n\}$, and $\{P_c^n\}$ be infinite families of programs where

$$P_a^n = \left\{ \begin{array}{l} x \leftarrow \sim x \\ x \leftarrow a_1, b_1 \\ \vdots \\ x \leftarrow a_n, b_n \end{array} \right\} \quad P_b^n = \left\{ \begin{array}{ll} x \leftarrow c_1, \dots, c_n, \sim x & \\ c_1 \leftarrow a_1 & c_1 \leftarrow b_1 \\ \vdots & \vdots \\ c_n \leftarrow a_n & c_n \leftarrow b_n \end{array} \right\} \quad P_c^n = \left\{ \begin{array}{l} a_1 \leftarrow \sim b_1 \\ b_1 \leftarrow \sim a_1 \\ \vdots \\ a_n \leftarrow \sim b_n \\ b_n \leftarrow \sim a_n \end{array} \right\}$$

- In minimal refutations for $P_a^n \cup P_c^n$, the number of applications of $Cut[body(P_a^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$ is linear in n, whereas $\mathcal{T}_{smodels}$ requires exponentially many applications of $Cut[atom(P_a^n \cup P_c^n)]$
  Vice versa, minimal refutations for $P_b^n \cup P_c^n$ require linearly many applications of $Cut[atom(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{smodels}$ and exponentially many applications of $Cut[body(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$

Potassco

$\mathcal{T}_{smodels}$ versus $\mathcal{T}_{noMoRe}$

- $\mathcal{T}_{smodels}$ restricts *Cut* to $atom(P)$ and $\mathcal{T}_{noMoRe}$ to $body(P)$
  Are both approaches similar or is one of them superior to the other?
- Let $\{P_a^n\}$, $\{P_b^n\}$, and $\{P_c^n\}$ be infinite families of programs where

$$P_a^n = \left\{ \begin{array}{l} x \leftarrow \sim x \\ x \leftarrow a_1, b_1 \\ \quad\vdots \\ x \leftarrow a_n, b_n \end{array} \right\} \quad P_b^n = \left\{ \begin{array}{ll} x \leftarrow c_1, \dots, c_n, \sim x & \\ c_1 \leftarrow a_1 & c_1 \leftarrow b_1 \\ \quad\vdots & \quad\vdots \\ c_n \leftarrow a_n & c_n \leftarrow b_n \end{array} \right\} \quad P_c^n = \left\{ \begin{array}{l} a_1 \leftarrow \sim b_1 \\ b_1 \leftarrow \sim a_1 \\ \quad\vdots \\ a_n \leftarrow \sim b_n \\ b_n \leftarrow \sim a_n \end{array} \right\}$$

- In minimal refutations for $P_a^n \cup P_c^n$, the number of applications of $Cut[body(P_a^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$ is linear in n, whereas $\mathcal{T}_{smodels}$ requires exponentially many applications of $Cut[atom(P_a^n \cup P_c^n)]$
  Vice versa, minimal refutations for $P_b^n \cup P_c^n$ require linearly many applications of $Cut[atom(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{smodels}$ and exponentially many applications of $Cut[body(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$

Potassco

$\mathcal{T}_{smodels}$ versus $\mathcal{T}_{noMoRe}$

- $\mathcal{T}_{smodels}$ restricts *Cut* to $atom(P)$ and $\mathcal{T}_{noMoRe}$ to $body(P)$
  Are both approaches similar or is one of them superior to the other?

- Let $\{P_a^n\}$, $\{P_b^n\}$, and $\{P_c^n\}$ be infinite families of programs where

$$P_a^n = \left\{ \begin{array}{l} x \leftarrow \sim x \\ x \leftarrow a_1, b_1 \\ \vdots \\ x \leftarrow a_n, b_n \end{array} \right\} \quad P_b^n = \left\{ \begin{array}{ll} x \leftarrow c_1, \dots, c_n, \sim x & \\ c_1 \leftarrow a_1 & c_1 \leftarrow b_1 \\ \vdots & \vdots \\ c_n \leftarrow a_n & c_n \leftarrow b_n \end{array} \right\} \quad P_c^n = \left\{ \begin{array}{l} a_1 \leftarrow \sim b_1 \\ b_1 \leftarrow \sim a_1 \\ \vdots \\ a_n \leftarrow \sim b_n \\ b_n \leftarrow \sim a_n \end{array} \right\}$$

- In minimal refutations for $P_a^n \cup P_c^n$, the number of applications of $Cut[body(P_a^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$ is linear in n, whereas $\mathcal{T}_{smodels}$ requires exponentially many applications of $Cut[atom(P_a^n \cup P_c^n)]$
  Vice versa, minimal refutations for $P_b^n \cup P_c^n$ require linearly many applications of $Cut[atom(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{smodels}$ and exponentially many applications of $Cut[body(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$

$\mathcal{T}_{smodels}$ versus $\mathcal{T}_{noMoRe}$

- $\mathcal{T}_{smodels}$ restricts *Cut* to $atom(P)$ and $\mathcal{T}_{noMoRe}$ to $body(P)$
  Are both approaches similar or is one of them superior to the other?
- Let $\{P_a^n\}$, $\{P_b^n\}$, and $\{P_c^n\}$ be infinite families of programs where

$$P_a^n = \left\{ \begin{array}{c} x \leftarrow {\sim}x \\ x \leftarrow a_1, b_1 \\ \vdots \\ x \leftarrow a_n, b_n \end{array} \right\} \quad P_b^n = \left\{ \begin{array}{cc} \multicolumn{2}{c}{x \leftarrow c_1, \dots, c_n, {\sim}x} \\ c_1 \leftarrow a_1 & c_1 \leftarrow b_1 \\ \vdots & \vdots \\ c_n \leftarrow a_n & c_n \leftarrow b_n \end{array} \right\} \quad P_c^n = \left\{ \begin{array}{c} a_1 \leftarrow {\sim}b_1 \\ b_1 \leftarrow {\sim}a_1 \\ \vdots \\ a_n \leftarrow {\sim}b_n \\ b_n \leftarrow {\sim}a_n \end{array} \right\}$$

- In minimal refutations for $P_a^n \cup P_c^n$, the number of applications of $Cut[body(P_a^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$ is linear in n, whereas $\mathcal{T}_{smodels}$ requires exponentially many applications of $Cut[atom(P_a^n \cup P_c^n)]$
- Vice versa, minimal refutations for $P_b^n \cup P_c^n$ require linearly many applications of $Cut[atom(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{smodels}$ and exponentially many applications of $Cut[body(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$

Potassco

$\mathcal{T}_{smodels}$ versus $\mathcal{T}_{noMoRe}$

- $\mathcal{T}_{smodels}$ restricts *Cut* to $atom(P)$ and $\mathcal{T}_{noMoRe}$ to $body(P)$
  Are both approaches similar or is one of them superior to the other?
- Let $\{P_a^n\}$, $\{P_b^n\}$, and $\{P_c^n\}$ be infinite families of programs where

$$P_a^n = \left\{ \begin{array}{c} x \leftarrow \sim x \\ x \leftarrow a_1, b_1 \\ \vdots \\ x \leftarrow a_n, b_n \end{array} \right\} \quad P_b^n = \left\{ \begin{array}{cc} \multicolumn{2}{c}{x \leftarrow c_1, \dots, c_n, \sim x} \\ c_1 \leftarrow a_1 & c_1 \leftarrow b_1 \\ \vdots & \vdots \\ c_n \leftarrow a_n & c_n \leftarrow b_n \end{array} \right\} \quad P_c^n = \left\{ \begin{array}{c} a_1 \leftarrow \sim b_1 \\ b_1 \leftarrow \sim a_1 \\ \vdots \\ a_n \leftarrow \sim b_n \\ b_n \leftarrow \sim a_n \end{array} \right\}$$

- In minimal refutations for $P_a^n \cup P_c^n$, the number of applications of $Cut[body(P_a^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$ is linear in n, whereas $\mathcal{T}_{smodels}$ requires exponentially many applications of $Cut[atom(P_a^n \cup P_c^n)]$
- Vice versa, minimal refutations for $P_b^n \cup P_c^n$ require linearly many applications of $Cut[atom(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{smodels}$ and exponentially many applications of $Cut[body(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$

$\mathcal{T}_{smodels}$ versus $\mathcal{T}_{noMoRe}$

- $\mathcal{T}_{smodels}$ restricts *Cut* to $atom(P)$ and $\mathcal{T}_{noMoRe}$ to $body(P)$
  Are both approaches similar or is one of them superior to the other?
- Let $\{P_a^n\}$, $\{P_b^n\}$, and $\{P_c^n\}$ be infinite families of programs where

$$P_a^n = \left\{ \begin{array}{l} x \leftarrow \sim x \\ x \leftarrow a_1, b_1 \\ \vdots \\ x \leftarrow a_n, b_n \end{array} \right\} \quad P_b^n = \left\{ \begin{array}{ll} \multicolumn{2}{l}{x \leftarrow c_1, \dots, c_n, \sim x} \\ c_1 \leftarrow a_1 & c_1 \leftarrow b_1 \\ \vdots & \vdots \\ c_n \leftarrow a_n & c_n \leftarrow b_n \end{array} \right\} \quad P_c^n = \left\{ \begin{array}{l} a_1 \leftarrow \sim b_1 \\ b_1 \leftarrow \sim a_1 \\ \vdots \\ a_n \leftarrow \sim b_n \\ b_n \leftarrow \sim a_n \end{array} \right\}$$

- In minimal refutations for $P_a^n \cup P_c^n$, the number of applications of $Cut[body(P_a^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$ is linear in n, whereas $\mathcal{T}_{smodels}$ requires exponentially many applications of $Cut[atom(P_a^n \cup P_c^n)]$
- Vice versa, minimal refutations for $P_b^n \cup P_c^n$ require linearly many applications of $Cut[atom(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{smodels}$ and exponentially many applications of $Cut[body(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$

$\mathcal{T}_{smodels}$ versus $\mathcal{T}_{noMoRe}$

- $\mathcal{T}_{smodels}$ restricts *Cut* to $atom(P)$ and $\mathcal{T}_{noMoRe}$ to $body(P)$
  Are both approaches similar or is one of them superior to the other?
- Let $\{P^n_a\}$, $\{P^n_b\}$, and $\{P^n_c\}$ be infinite families of programs where

$$P^n_a = \left\{ \begin{array}{c} x \leftarrow {\sim}x \\ x \leftarrow a_1, b_1 \\ \vdots \\ x \leftarrow a_n, b_n \end{array} \right\} \quad P^n_b = \left\{ \begin{array}{cc} \multicolumn{2}{c}{x \leftarrow c_1, \dots, c_n, {\sim}x} \\ c_1 \leftarrow a_1 & c_1 \leftarrow b_1 \\ \vdots & \vdots \\ c_n \leftarrow a_n & c_n \leftarrow b_n \end{array} \right\} \quad P^n_c = \left\{ \begin{array}{c} a_1 \leftarrow {\sim}b_1 \\ b_1 \leftarrow {\sim}a_1 \\ \vdots \\ a_n \leftarrow {\sim}b_n \\ b_n \leftarrow {\sim}a_n \end{array} \right\}$$

- In minimal refutations for $P^n_a \cup P^n_c$, the number of applications of $Cut[body(P^n_a \cup P^n_c)]$ with $\mathcal{T}_{noMoRe}$ is linear in n, whereas $\mathcal{T}_{smodels}$ requires exponentially many applications of $Cut[atom(P^n_a \cup P^n_c)]$
- Vice versa, minimal refutations for $P^n_b \cup P^n_c$ require linearly many applications of $Cut[atom(P^n_b \cup P^n_c)]$ with $\mathcal{T}_{smodels}$ and exponentially many applications of $Cut[body(P^n_b \cup P^n_c)]$ with $\mathcal{T}_{noMoRe}$

$\mathcal{T}_{smodels}$ versus $\mathcal{T}_{noMoRe}$

- $\mathcal{T}_{smodels}$ restricts *Cut* to $atom(P)$ and $\mathcal{T}_{noMoRe}$ to $body(P)$
  Are both approaches similar or is one of them superior to the other?
- Let $\{P_a^n\}$, $\{P_b^n\}$, and $\{P_c^n\}$ be infinite families of programs where

$$P_a^n = \left\{ \begin{array}{c} x \leftarrow \sim x \\ x \leftarrow a_1, b_1 \\ \vdots \\ x \leftarrow a_n, b_n \end{array} \right\} \quad P_b^n = \left\{ \begin{array}{cc} \multicolumn{2}{c}{x \leftarrow c_1, \dots, c_n, \sim x} \\ c_1 \leftarrow a_1 & c_1 \leftarrow b_1 \\ \vdots & \vdots \\ c_n \leftarrow a_n & c_n \leftarrow b_n \end{array} \right\} \quad P_c^n = \left\{ \begin{array}{c} a_1 \leftarrow \sim b_1 \\ b_1 \leftarrow \sim a_1 \\ \vdots \\ a_n \leftarrow \sim b_n \\ b_n \leftarrow \sim a_n \end{array} \right\}$$

- In minimal refutations for $P_a^n \cup P_c^n$, the number of applications of $Cut[body(P_a^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$ is linear in n, whereas $\mathcal{T}_{smodels}$ requires exponentially many applications of $Cut[atom(P_a^n \cup P_c^n)]$
- Vice versa, minimal refutations for $P_b^n \cup P_c^n$ require linearly many applications of $Cut[atom(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{smodels}$ and exponentially many applications of $Cut[body(P_b^n \cup P_c^n)]$ with $\mathcal{T}_{noMoRe}$

Relative efficiency

- As witnessed by $\{P_a^n \cup P_c^n\}$ and $\{P_b^n \cup P_c^n\}$, respectively, $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ do not polynomially simulate one another
- Any refutation of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{noMoRe}$ is a refutation of $\mathcal{T}_{nomore^{++}}$ (but not vice versa)
- Hence
  - both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are polynomially simulated by $\mathcal{T}_{nomore^{++}}$ and
  - $\mathcal{T}_{nomore^{++}}$ is polynomially simulated by neither $\mathcal{T}_{smodels}$ nor $\mathcal{T}_{noMoRe}$
- More generally, the proof system obtained with $Cut[atom(P) \cup body(P)]$ is exponentially stronger than the ones with either $Cut[atom(P)]$ or $Cut[body(P)]$
- Case analyses (at least) on atoms and bodies are mandatory in powerful ASP solvers

Relative efficiency

- As witnessed by $\{P_a^n \cup P_c^n\}$ and $\{P_b^n \cup P_c^n\}$, respectively, $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ do not polynomially simulate one another
- Any refutation of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{noMoRe}$ is a refutation of $\mathcal{T}_{nomore^{++}}$ (but not vice versa)
- Hence
  - both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are polynomially simulated by $\mathcal{T}_{nomore^{++}}$ and
  - $\mathcal{T}_{nomore^{++}}$ is polynomially simulated by neither $\mathcal{T}_{smodels}$ nor $\mathcal{T}_{noMoRe}$
- More generally, the proof system obtained with $Cut[atom(P) \cup body(P)]$ is exponentially stronger than the ones with either $Cut[atom(P)]$ or $Cut[body(P)]$
- Case analyses (at least) on atoms and bodies are mandatory in powerful ASP solvers

Relative efficiency

- As witnessed by $\{P_a^n \cup P_c^n\}$ and $\{P_b^n \cup P_c^n\}$, respectively, $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ do not polynomially simulate one another
- Any refutation of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{noMoRe}$ is a refutation of $\mathcal{T}_{nomore^{++}}$ (but not vice versa)
- Hence
  - both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are polynomially simulated by $\mathcal{T}_{nomore^{++}}$ and
  - $\mathcal{T}_{nomore^{++}}$ is polynomially simulated by neither $\mathcal{T}_{smodels}$ nor $\mathcal{T}_{noMoRe}$

- More generally, the proof system obtained with $Cut[atom(P) \cup body(P)]$ is exponentially stronger than the ones with either $Cut[atom(P)]$ or $Cut[body(P)]$
- Case analyses (at least) on atoms and bodies are mandatory in powerful ASP solvers

Relative efficiency

- As witnessed by $\{P_a^n \cup P_c^n\}$ and $\{P_b^n \cup P_c^n\}$, respectively, $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ do not polynomially simulate one another
- Any refutation of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{noMoRe}$ is a refutation of $\mathcal{T}_{nomore^{++}}$ (but not vice versa)
- Hence
  - both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are polynomially simulated by $\mathcal{T}_{nomore^{++}}$ and
  - $\mathcal{T}_{nomore^{++}}$ is polynomially simulated by neither $\mathcal{T}_{smodels}$ nor $\mathcal{T}_{noMoRe}$

- More generally, the proof system obtained with $Cut[atom(P) \cup body(P)]$ is exponentially stronger than the ones with either $Cut[atom(P)]$ or $Cut[body(P)]$
- Case analyses (at least) on atoms and bodies are mandatory in powerful ASP solvers

Relative efficiency

- As witnessed by $\{P_a^n \cup P_c^n\}$ and $\{P_b^n \cup P_c^n\}$, respectively, $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ do not polynomially simulate one another
- Any refutation of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{noMoRe}$ is a refutation of $\mathcal{T}_{nomore^{++}}$ (but not vice versa)
- Hence
  - both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{noMoRe}$ are polynomially simulated by $\mathcal{T}_{nomore^{++}}$ and
  - $\mathcal{T}_{nomore^{++}}$ is polynomially simulated by neither $\mathcal{T}_{smodels}$ nor $\mathcal{T}_{noMoRe}$

- More generally, the proof system obtained with $Cut[atom(P) \cup body(P)]$ is exponentially stronger than the ones with either $Cut[atom(P)]$ or $Cut[body(P)]$
- Case analyses (at least) on atoms and bodies are mandatory in powerful ASP solvers

$\mathcal{T}_{smodels}$: Example tableau

(r_1)	$a \leftarrow \sim b$		(r_2)	$b \leftarrow d, \sim a$		(r_3)	$c \leftarrow b, d$
(r_4)	$c \leftarrow g$		(r_5)	$d \leftarrow c$		(r_6)	$d \leftarrow g$
(r_7)	$e \leftarrow f, \sim c$		(r_8)	$f \leftarrow \sim g$		(r_9)	$g \leftarrow \sim a, \sim f$

(1)	$\boldsymbol{T}a$	[*Cut*]
(2)	$\boldsymbol{T}\{\sim b\}$	[*BTA*: r_1, 1]
(3)	$\boldsymbol{F}b$	[*BTB*: 2]
(4)	$\boldsymbol{F}\{d, \sim a\}$	[*BFA*: r_2, 3]
(5)	$\boldsymbol{F}\{\sim a, \sim f\}$	[*FFB*: r_9, 1]
(6)	$\boldsymbol{F}g$	[*FFA*: r_9, 5]
(7)	$\boldsymbol{T}\{\sim g\}$	[*FTB*: r_8, 6]
(8)	$\boldsymbol{T}f$	[*FTA*: r_8, 7]
(9)	$\boldsymbol{F}\{b, d\}$	[*FFB*: r_3, 3]
(10)	$\boldsymbol{F}\{g\}$	[*FFB*: r_4, r_6, 6]
(11)	$\boldsymbol{F}c$	[*FFA*: r_3, r_4, 9, 10]
(12)	$\boldsymbol{F}\{c\}$	[*FFB*: r_5, 11]
(13)	$\boldsymbol{F}d$	[*FFA*: r_5, r_6, 10, 12]
(14)	$\boldsymbol{T}\{f, \sim c\}$	[*FTB*: r_7, 8, 11]
(15)	$\boldsymbol{T}e$	[*FTA*: r_7, 14]

(16)	$\boldsymbol{F}a$	[*Cut*]
(17)	$\boldsymbol{F}\{\sim b\}$	[*BFA*: r_1, 16]
(18)	$\boldsymbol{T}b$	[*BFB*: 17]
(19)	$\boldsymbol{T}\{d, \sim a\}$	[*BTA*: r_2, 18]
(20)	$\boldsymbol{T}d$	[*BTB*: 19]
(21)	$\boldsymbol{T}\{b, d\}$	[*FTB*: r_3, 18, 20]
(22)	$\boldsymbol{T}c$	[*FTA*: r_3, 21]
(23)	$\boldsymbol{F}\{f, \sim c\}$	[*FFB*: r_7, 22]
(24)	$\boldsymbol{F}e$	[*FFA*: r_7, 23]
(25)	$\boldsymbol{T}\{c\}$	[*FTB*: r_5, 22]

(26)	$\boldsymbol{T}f$	[*Cut*]
(27)	$\boldsymbol{F}\{\sim a, \sim f\}$	[*FFB*: r_9, 26]
(28)	$\boldsymbol{F}c$	[*WFN*: 27]

(29)	$\boldsymbol{F}f$	[*Cut*]
(30)	$\boldsymbol{T}\{\sim a, \sim f\}$	[*FTB*: r_9, 16, 29]
(31)	$\boldsymbol{T}g$	[*FTA*: r_9, 30]
(32)	$\boldsymbol{T}\{g\}$	[*FTB*: r_4, r_6, 31]
(33)	$\boldsymbol{F}\{\sim g\}$	[*FFB*: r_8, 31]

$\mathcal{T}_{noMoRe}$: Example tableau

(r_1)	$a \leftarrow \sim b$		(r_2)	$b \leftarrow d, \sim a$		(r_3)	$c \leftarrow b, d$	
(r_4)	$c \leftarrow g$		(r_5)	$d \leftarrow c$		(r_6)	$d \leftarrow g$	
(r_7)	$e \leftarrow f, \sim c$		(r_8)	$f \leftarrow \sim g$		(r_9)	$g \leftarrow \sim a, \sim f$	

(1)	$\boldsymbol{T}\{\sim b\}$	$[Cut]$
(2)	$\boldsymbol{T}a$	$[FTA\text{: } r_1, 1]$
(3)	$\boldsymbol{F}b$	$[BTB\text{: } 1]$
(4)	$\boldsymbol{F}\{d, \sim a\}$	$[BFA\text{: } r_2, 3]$
(5)	$\boldsymbol{F}\{\sim a, \sim f\}$	$[FFB\text{: } r_9, 2]$
(6)	$\boldsymbol{F}g$	$[FFA\text{: } r_9, 5]$
(7)	$\boldsymbol{T}\{\sim g\}$	$[FTB\text{: } r_8, 6]$
(8)	$\boldsymbol{T}f$	$[FTA\text{: } r_8, 7]$
(9)	$\boldsymbol{F}\{b, d\}$	$[FFB\text{: } r_3, 3]$
(10)	$\boldsymbol{F}\{g\}$	$[FFB\text{: } r_4, r_6, 6]$
(11)	$\boldsymbol{F}c$	$[FFA\text{: } r_3, r_4, 9, 10]$
(12)	$\boldsymbol{F}\{c\}$	$[FFB\text{: } r_5, 11]$
(13)	$\boldsymbol{F}d$	$[FFA\text{: } r_5, r_6, 10, 12]$
(14)	$\boldsymbol{T}\{f, \sim c\}$	$[FTB\text{: } r_7, 8, 11]$
(15)	$\boldsymbol{T}e$	$[FTA\text{: } r_7, 14]$

(16)	$\boldsymbol{F}\{\sim b\}$	$[Cut]$
(17)	$\boldsymbol{F}a$	$[FFA\text{: } r_1, 16]$
(18)	$\boldsymbol{T}b$	$[BFB\text{: } 16]$
(19)	$\boldsymbol{T}\{d, \sim a\}$	$[BTA\text{: } r_2, 18]$
(20)	$\boldsymbol{T}d$	$[BTB\text{: } 19]$
(21)	$\boldsymbol{T}\{b, d\}$	$[FTB\text{: } r_3, 18, 20]$
(22)	$\boldsymbol{T}c$	$[FTA\text{: } r_3, 21]$
(23)	$\boldsymbol{F}\{f, \sim c\}$	$[FFB\text{: } r_7, 22]$
(24)	$\boldsymbol{F}e$	$[FFA\text{: } r_7, 23]$
(25)	$\boldsymbol{T}\{c\}$	$[FTB\text{: } r_5, 22]$

(26)	$\boldsymbol{T}\{\sim g\}$	$[Cut]$
(27)	$\boldsymbol{F}g$	$[BTB\text{: } 26]$
(28)	$\boldsymbol{F}\{g\}$	$[FFB\text{: } r_4, r_6, 27]$
(29)	$\boldsymbol{F}c$	$[WFN\text{: } 28]$

(30)	$\boldsymbol{F}\{\sim g\}$	$[Cut]$
(31)	$\boldsymbol{T}g$	$[BFB\text{: } 30]$
(32)	$\boldsymbol{T}\{g\}$	$[FTB\text{: } r_4, r_6, 31]$
(33)	$\boldsymbol{F}f$	$[FFA\text{: } r_8, 30]$
(34)	$\boldsymbol{T}\{\sim a, \sim f\}$	$[FTB\text{: } r_9, 17, 33]$

$\mathcal{T}_{nomore^{++}}$: Example tableau

$$
\begin{array}{llllll}
(r_1) & a \leftarrow \sim b & (r_2) & b \leftarrow d, \sim a & (r_3) & c \leftarrow b, d \\
(r_4) & c \leftarrow g & (r_5) & d \leftarrow c & (r_6) & d \leftarrow g \\
(r_7) & e \leftarrow f, \sim c & (r_8) & f \leftarrow \sim g & (r_9) & g \leftarrow \sim a, \sim f
\end{array}
$$

(1)	$\boldsymbol{T}a$	[*Cut*]
(2)	$\boldsymbol{T}\{\sim b\}$	[*BTA*: r_1, 1]
(3)	$\boldsymbol{F}b$	[*BTB*: 2]
(4)	$\boldsymbol{F}\{d, \sim a\}$	[*BFA*: r_2, 3]
(5)	$\boldsymbol{F}\{\sim a, \sim f\}$	[*FFB*: r_9, 1]
(6)	$\boldsymbol{F}g$	[*FFA*: r_9, 5]
(7)	$\boldsymbol{T}\{\sim g\}$	[*FTB*: r_8, 6]
(8)	$\boldsymbol{T}f$	[*FTA*: r_8, 7]
(9)	$\boldsymbol{F}\{b, d\}$	[*FFB*: r_3, 3]
(10)	$\boldsymbol{F}\{g\}$	[*FFB*: r_4, r_6, 6]
(11)	$\boldsymbol{F}c$	[*FFA*: r_3, r_4, 9, 10]
(12)	$\boldsymbol{F}\{c\}$	[*FFB*: r_5, 11]
(13)	$\boldsymbol{F}d$	[*FFA*: r_5, r_6, 10, 12]
(14)	$\boldsymbol{T}\{f, \sim c\}$	[*FTB*: r_7, 8, 11]
(15)	$\boldsymbol{T}e$	[*FTA*: r_7, 14]

(16)	$\boldsymbol{F}a$	[*Cut*]
(17)	$\boldsymbol{F}\{\sim b\}$	[*BFA*: r_1, 16]
(18)	$\boldsymbol{T}b$	[*BFB*: 17]
(19)	$\boldsymbol{T}\{d, \sim a\}$	[*BTA*: r_2, 18]
(20)	$\boldsymbol{T}d$	[*BTB*: 19]
(21)	$\boldsymbol{T}\{b, d\}$	[*FTB*: r_3, 18, 20]
(22)	$\boldsymbol{T}c$	[*FTA*: r_3, 21]
(23)	$\boldsymbol{F}\{f, \sim c\}$	[*FFB*: r_7, 22]
(24)	$\boldsymbol{F}e$	[*FFA*: r_7, 23]
(25)	$\boldsymbol{T}\{c\}$	[*FTB*: r_5, 22]

(26)	$\boldsymbol{T}\{\sim g\}$	[*Cut*]
(27)	$\boldsymbol{F}g$	[*BTB*: 26]
(28)	$\boldsymbol{F}\{g\}$	[*FFB*: r_4, r_6, 27]
(29)	$\boldsymbol{F}c$	[*WFN*: 28]

(30)	$\boldsymbol{F}\{\sim g\}$	[*Cut*]
(31)	$\boldsymbol{T}g$	[*BFB*: 30]
(32)	$\boldsymbol{T}\{g\}$	[*FTB*: r_4, r_6, 31]
(33)	$\boldsymbol{F}f$	[*FFA*: r_8, 30]
(34)	$\boldsymbol{T}\{\sim a, \sim f\}$	[*FTB*: r_9, 16, 33]

[1] C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub.
The `nomore++` approach to answer set solving.
In G. Sutcliffe and A. Voronkov, editors, *Proceedings of the Twelfth International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, volume 3835 of *Lecture Notes in Artificial Intelligence*, pages 95–109. Springer-Verlag, 2005.

[2] C. Anger, K. Konczak, T. Linke, and T. Schaub.
A glimpse of answer set programming.
Künstliche Intelligenz, 19(1):12–17, 2005.

[3] Y. Babovich and V. Lifschitz.
Computing answer sets using program completion.
Unpublished draft, 2003.

[4] C. Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2003.

[5] C. Baral, G. Brewka, and J. Schlipf, editors.
Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07), volume 4483 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2007.

[6] C. Baral and M. Gelfond.
Logic programming and knowledge representation.
Journal of Logic Programming, 12:1–80, 1994.

[7] S. Baselice, P. Bonatti, and M. Gelfond.
Towards an integration of answer set and constraint solving.
In M. Gabbrielli and G. Gupta, editors, *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66. Springer-Verlag, 2005.

[8] A. Biere.
Adaptive restart strategies for conflict driven SAT solvers.

In H. Kleine Büning and X. Zhao, editors, *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer-Verlag, 2008.

[9] A. Biere.
PicoSAT essentials.
Journal on Satisfiability, Boolean Modeling and Computation, 4:75–97, 2008.

[10] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors.
Handbook of Satisfiability, volume 185 of *Frontiers in Artificial Intelligence and Applications*.
IOS Press, 2009.

[11] G. Brewka, T. Eiter, and M. Truszczyński.
Answer set programming at a glance.
Communications of the ACM, 54(12):92–103, 2011.

[12] K. Clark.

Negation as failure.
In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[13] M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors.
Handbook of Tableau Methods.
Kluwer Academic Publishers, 1999.

[14] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov.
Complexity and expressive power of logic programming.
In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity (CCC'97)*, pages 82–101. IEEE Computer Society Press, 1997.

[15] M. Davis, G. Logemann, and D. Loveland.
A machine program for theorem-proving.
Communications of the ACM, 5:394–397, 1962.

[16] M. Davis and H. Putnam.
A computing procedure for quantification theory.

Journal of the ACM, 7:201–215, 1960.

[17] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub.
Conflict-driven disjunctive answer set solving.
In G. Brewka and J. Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 422–432. AAAI Press, 2008.

[18] C. Drescher, M. Gebser, B. Kaufmann, and T. Schaub.
Heuristics in conflict resolution.
In M. Pagnucco and M. Thielscher, editors, *Proceedings of the Twelfth International Workshop on Nonmonotonic Reasoning (NMR'08)*, number UNSW-CSE-TR-0819 in School of Computer Science and Engineering, The University of New South Wales, Technical Report Series, pages 141–149, 2008.

[19] N. Eén and N. Sörensson.
An extensible SAT-solver.

In E. Giunchiglia and A. Tacchella, editors, *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer-Verlag, 2004.

[20] T. Eiter and G. Gottlob.
On the computational cost of disjunctive logic programming: Propositional case.
Annals of Mathematics and Artificial Intelligence, 15(3-4):289–323, 1995.

[21] T. Eiter, G. Ianni, and T. Krennwallner.
Answer Set Programming: A Primer.
In S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M. Rousset, and R. Schmidt, editors, *Fifth International Reasoning Web Summer School (RW'09)*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer-Verlag, 2009.

[22] F. Fages.
Consistency of Clark's completion and the existence of stable models.

Journal of Methods of Logic in Computer Science, 1:51–60, 1994.

[23] P. Ferraris.
Answer sets for propositional theories.
In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, pages 119–131. Springer-Verlag, 2005.

[24] P. Ferraris and V. Lifschitz.
Mathematical foundations of answer set programming.
In S. Artëmov, H. Barringer, A. d'Avila Garcez, L. Lamb, and J. Woods, editors, *We Will Show Them! Essays in Honour of Dov Gabbay*, volume 1, pages 615–664. College Publications, 2005.

[25] M. Fitting.
A Kripke-Kleene semantics for logic programs.
Journal of Logic Programming, 2(4):295–312, 1985.

[26] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`.

[27] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
Engineering an incremental ASP solver.
In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer-Verlag, 2008.

[28] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
On the implementation of weight constraint rules in conflict-driven ASP solvers.
In Hill and Warren [44], pages 250–264.

[29] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
Answer Set Solving in Practice.

Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

[30] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
clasp: A conflict-driven answer set solver.
In Baral et al. [5], pages 260–265.

[31] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set enumeration.
In Baral et al. [5], pages 136–148.

[32] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set solving.
In Veloso [68], pages 386–392.

[33] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Advanced preprocessing for answer set solving.
In M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, pages 15–19. IOS Press, 2008.

[34] M. Gebser, B. Kaufmann, and T. Schaub.
The conflict-driven answer set solver clasp: Progress report.
In E. Erdem, F. Lin, and T. Schaub, editors, *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Artificial Intelligence*, pages 509–514. Springer-Verlag, 2009.

[35] M. Gebser, B. Kaufmann, and T. Schaub.
Solution enumeration for projected Boolean search problems.
In W. van Hoeve and J. Hooker, editors, *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2009.

[36] M. Gebser, M. Ostrowski, and T. Schaub.
Constraint answer set solving.
In Hill and Warren [44], pages 235–249.

[37] M. Gebser and T. Schaub.
Tableau calculi for answer set programming.
In S. Etalle and M. Truszczyński, editors, *Proceedings of the Twenty-second International Conference on Logic Programming (ICLP'06)*, volume 4079 of *Lecture Notes in Computer Science*, pages 11–25. Springer-Verlag, 2006.

[38] M. Gebser and T. Schaub.
Generic tableaux for answer set programming.
In V. Dahl and I. Niemelä, editors, *Proceedings of the Twenty-third International Conference on Logic Programming (ICLP'07)*, volume 4670 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 2007.

[39] M. Gelfond.
Answer sets.
In V. Lifschitz, F. van Harmelen, and B. Porter, editors, *Handbook of Knowledge Representation*, chapter 7, pages 285–316. Elsevier Science, 2008.

[40] M. Gelfond and N. Leone.
Logic programming and knowledge representation — the A-Prolog perspective.
Artificial Intelligence, 138(1-2):3–38, 2002.

[41] M. Gelfond and V. Lifschitz.
The stable model semantics for logic programming.
In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.

[42] M. Gelfond and V. Lifschitz.
Logic programs with classical negation.
In D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming (ICLP'90)*, pages 579–597. MIT Press, 1990.

[43] E. Giunchiglia, Y. Lierler, and M. Maratea.
Answer set programming based on propositional satisfiability.

Journal of Automated Reasoning, 36(4):345–377, 2006.

[44] P. Hill and D. Warren, editors.
Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09), volume 5649 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[45] J. Huang.
The effect of restarts on the efficiency of clause learning.
In Veloso [68], pages 2318–2323.

[46] K. Konczak, T. Linke, and T. Schaub.
Graphs and colorings for answer set programming.
Theory and Practice of Logic Programming, 6(1-2):61–106, 2006.

[47] J. Lee.
A model-theoretic counterpart of loop formulas.
In L. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 503–508. Professional Book Center, 2005.

[48] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello.
The DLV system for knowledge representation and reasoning.
ACM Transactions on Computational Logic, 7(3):499–562, 2006.

[49] V. Lifschitz.
Answer set programming and plan generation.
Artificial Intelligence, 138(1-2):39–54, 2002.

[50] V. Lifschitz.
Introduction to answer set programming.
Unpublished draft, 2004.

[51] V. Lifschitz and A. Razborov.
Why are there so many loop formulas?
ACM Transactions on Computational Logic, 7(2):261–268, 2006.

[52] F. Lin and Y. Zhao.
ASSAT: computing answer sets of a logic program by SAT solvers.
Artificial Intelligence, 157(1-2):115–137, 2004.

[53] V. Marek and M. Truszczyński.
Nonmonotonic logic: context-dependent reasoning.
Artifical Intelligence. Springer-Verlag, 1993.

[54] V. Marek and M. Truszczyński.
Stable models and an alternative logic programming paradigm.
In K. Apt, V. Marek, M. Truszczyński, and D. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[55] J. Marques-Silva, I. Lynce, and S. Malik.
Conflict-driven clause learning SAT solvers.
In Biere et al. [10], chapter 4, pages 131–153.

[56] J. Marques-Silva and K. Sakallah.
GRASP: A search algorithm for propositional satisfiability.
IEEE Transactions on Computers, 48(5):506–521, 1999.

[57] V. Mellarkod and M. Gelfond.
Integrating answer set reasoning with constraint solving techniques.

In J. Garrigue and M. Hermenegildo, editors, *Proceedings of the Ninth International Symposium on Functional and Logic Programming (FLOPS'08)*, volume 4989 of *Lecture Notes in Computer Science*, pages 15–31. Springer-Verlag, 2008.

[58] V. Mellarkod, M. Gelfond, and Y. Zhang.
Integrating answer set programming and constraint logic programming.
Annals of Mathematics and Artificial Intelligence, 53(1-4):251–287, 2008.

[59] D. Mitchell.
A SAT solver primer.
Bulletin of the European Association for Theoretical Computer Science, 85:112–133, 2005.

[60] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik.
Chaff: Engineering an efficient SAT solver.
In *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*, pages 530–535. ACM Press, 2001.

[61] I. Niemelä.
Logic programs with stable model semantics as a constraint programming paradigm.
Annals of Mathematics and Artificial Intelligence, 25(3-4):241–273, 1999.

[62] R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T).
Journal of the ACM, 53(6):937–977, 2006.

[63] K. Pipatsrisawat and A. Darwiche.
A lightweight component caching scheme for satisfiability solvers.
In J. Marques-Silva and K. Sakallah, editors, *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer-Verlag, 2007.

[64] L. Ryan.

Efficient algorithms for clause-learning SAT solvers.
Master's thesis, Simon Fraser University, 2004.

[65] P. Simons, I. Niemelä, and T. Soininen.
Extending and implementing the stable model semantics.
Artificial Intelligence, 138(1-2):181–234, 2002.

[66] T. Syrjänen.
Lparse 1.0 user's manual.

[67] A. Van Gelder, K. Ross, and J. Schlipf.
The well-founded semantics for general logic programs.
Journal of the ACM, 38(3):620–650, 1991.

[68] M. Veloso, editor.
Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07). AAAI/MIT Press, 2007.

[69] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik.
Efficient conflict driven learning in a Boolean satisfiability solver.

In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285. ACM Press, 2001.