TECHNISCHE UNIVERSITÄT DRESDEN

DEDUCTION SYSTEMS

Answer Set Programming: Basics

Markus Krötzsch

Chair for Knowledge-Based Systems

TU Dresden, 18 June 2018

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

ASP Basics: Overview

Outline

Answer Set Programming

in a Nutshell

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is supported by several fast solvers, such as clasp, DLV, and smodels

Outline

Normal logic programs

- A logic program, P, over a set $\mathcal{A}$ of atoms is a finite set of rules
- A (normal) rule, r, is of the form

$$a_0 \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n$$

where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$

Normal logic programs

- A logic program, P, over a set $\mathcal{A}$ of atoms is a finite set of rules
- A (normal) rule, r, is of the form

$$a_0 \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n$$

  where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$
- Notation

$$\begin{array}{rcl}
head(r) & = & a_0 \\
body(r) & = & \{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\} \\
body(r)^+ & = & \{a_1, \ldots, a_m\} \\
body(r)^- & = & \{a_{m+1}, \ldots, a_n\} \\
atom(P) & = & \bigcup_{r \in P} \left(\{head(r)\} \cup body(r)^+ \cup body(r)^-\right) \\
body(P) & = & \{body(r) \mid r \in P\}
\end{array}$$

Normal logic programs

- A logic program, P, over a set $\mathcal{A}$ of atoms is a finite set of rules
- A (normal) rule, r, is of the form

$$a_0 \leftarrow a_1, \ldots, a_m, {\sim}a_{m+1}, \ldots, {\sim}a_n$$

  where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$
- Notation

$$\begin{array}{rcl} head(r) & = & a_0 \\ body(r) & = & \{a_1, \ldots, a_m, {\sim}a_{m+1}, \ldots, {\sim}a_n\} \\ body(r)^+ & = & \{a_1, \ldots, a_m\} \\ body(r)^- & = & \{a_{m+1}, \ldots, a_n\} \\ atom(P) & = & \bigcup_{r \in P} \left(\{head(r)\} \cup body(r)^+ \cup body(r)^-\right) \\ body(P) & = & \{body(r) \mid r \in P\} \end{array}$$

- A program P is positive if $body(r)^- = \emptyset$ for all $r \in P$

Outline

Formal Definition

Stable models of positive programs

Formal Definition

Stable models of positive programs

- A set of atoms X is closed under a positive program P iff for any $r \in P$, $head(r) \in X$ whenever $body(r)^{+} \subseteq X$
  - Then X (seen as an interpretation) corresponds to a model of P (seen as a propositional logic formula)

Formal Definition

Stable models of positive programs

- A set of atoms X is closed under a positive program P iff for any $r \in P$, $head(r) \in X$ whenever $body(r)^+ \subseteq X$
  - Then X (seen as an interpretation) corresponds to a model of P (seen as a propositional logic formula)
- The smallest set of atoms which is closed under a positive program P is denoted by $Cn(P)$
  - $Cn(P)$ corresponds to the $\subseteq$-smallest model of P

Formal Definition

Stable models of positive programs

- A set of atoms X is closed under a positive program P iff for any $r \in P$, $head(r) \in X$ whenever $body(r)^{+} \subseteq X$
  - Then X (seen as an interpretation) corresponds to a model of P (seen as a propositional logic formula)
- The smallest set of atoms which is closed under a positive program P is denoted by $Cn(P)$
  - $Cn(P)$ corresponds to the $\subseteq$-smallest model of P
- The set $Cn(P)$ of atoms is the stable model of a positive program P

Formal Definition

Stable models of normal programs

- The (Gelfond-Lifschitz) reduct P^X of a program P relative to a set X of atoms is defined by

$$P^X = \{head(r) \leftarrow body(r)^+ \mid r \in P \text{ and } body(r)^- \cap X = \emptyset\}$$

Formal Definition

Stable models of normal programs

- The (Gelfond-Lifschitz) reduct P^X of a program P relative to a set X of atoms is defined by

$$P^X = \{head(r) \leftarrow body(r)^+ \mid r \in P \text{ and } body(r)^- \cap X = \emptyset\}$$

- A set X of atoms is a stable model of a program P, if $Cn(P^X) = X$

Formal Definition

Stable models of normal programs

- The (Gelfond-Lifschitz) reduct P^X of a program P relative to a set X of atoms is defined by

$$P^X = \{head(r) \leftarrow body(r)^+ \mid r \in P \text{ and } body(r)^- \cap X = \emptyset\}$$

- A set X of atoms is a stable model of a program P, if $Cn(P^X) = X$

- Note: $Cn(P^X)$ is the $\subseteq$–smallest (classical) model of P^X
- Note: Every atom in X is justified by an “applying rule from P”

A closer look at P^X

- In other words, given a set X of atoms from P,

  P^X is obtained from P by deleting

  (1) each rule having $\sim a$ in its body with $a \in X$
  and then

  (2) all negative atoms of the form $\sim a$
  in the bodies of the remaining rules

A closer look at P^X

- In other words, given a set X of atoms from P,

  P^X is obtained from P by deleting

  (1) each rule having $\sim a$ in its body with $a \in X$
  and then

  (2) all negative atoms of the form $\sim a$
  in the bodies of the remaining rules

- Note: Only negative body literals are evaluated wrt X

Outline

A first example

$$P = \{p \leftarrow p,\ q \leftarrow {\sim}p\}$$

A first example

$$P = \{p \leftarrow p,\ q \leftarrow {\sim}p\}$$

X		$Cn(P^X)$
$\{\ \}$		
$\{p\}$		
$\{q\}$		
$\{p, q\}$		

A first example

$$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$
$\{\ \ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$
$\{p\ \}$	$p \leftarrow p$	$\emptyset$
$\{\ q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$
$\{p, q\}$	$p \leftarrow p$	$\emptyset$

A first example

$$P = \{p \leftarrow p,\; q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$	
$\{\quad\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\quad\}$	$p \leftarrow p$	$\emptyset$	
$\{\quad q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

A first example

$$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\ \}$	$p \leftarrow p$	$\emptyset$	✘
$\{\ q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

A first example

$$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\ \}$	$p \leftarrow p$	$\emptyset$	✘
$\{\ q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	

TECHNISCHE UNIVERSITÄT DRESDEN

A first example

$$P = \{p \leftarrow p,\ q \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✘
$\{p\ \}$	$p \leftarrow p$	$\emptyset$	✘
$\{\ q\}$	$p \leftarrow p$ $q \leftarrow$	$\{q\}$	✔
$\{p, q\}$	$p \leftarrow p$	$\emptyset$	✘

A second example

$$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$$

A second example

$$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$
$\{\quad\}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$
$\{p\quad\}$	$p \leftarrow$	$\{p\}$
$\{\quad q\}$	$q \leftarrow$	$\{q\}$
$\{p, q\}$		$\emptyset$

A second example

$$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$
$\{\ \ \}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$ ✘
$\{p\ \}$	$p \leftarrow$	$\{p\}$
$\{\ q\}$	$q \leftarrow$	$\{q\}$
$\{p, q\}$		$\emptyset$

A second example

$$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\ \}$	$p \leftarrow$	$\{p\}$	✔
$\{\ q\}$	$q \leftarrow$	$\{q\}$	
$\{p, q\}$		$\emptyset$	

A second example

$$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$	
$\{\ \ \}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\ \}$	$p \leftarrow$	$\{p\}$	✔
$\{\ q\}$	$q \leftarrow$	$\{q\}$	✔
$\{p, q\}$		$\emptyset$	

A second example

$$P = \{p \leftarrow {\sim}q,\ q \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$	
$\{\quad\}$	$p \leftarrow$ $q \leftarrow$	$\{p, q\}$	✘
$\{p\quad\}$	$p \leftarrow$	$\{p\}$	✔
$\{\quad q\}$	$q \leftarrow$	$\{q\}$	✔
$\{p, q\}$		$\emptyset$	✘

A third example

$$P = \{p \leftarrow {\sim}p\}$$

A third example

$$P = \{p \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$
$\{\ \}$	$p \leftarrow$	$\{p\}$
$\{p\}$		$\emptyset$

A third example

$$P = \{p \leftarrow {\sim}p\}$$

X	P^X	$Cn(P^X)$	
$\{\ \}$	$p \leftarrow$	$\{p\}$	✘
$\{p\}$		$\emptyset$	

TECHNISCHE UNIVERSITÄT DRESDEN

A third example

$$P = \{p \leftarrow \sim p\}$$

X	P^X	$Cn(P^X)$	
$\{\ \}$	$p \leftarrow$	$\{p\}$	✘
$\{p\}$		$\emptyset$	✘

Some properties

- A logic program may have zero, one, or multiple stable models!

Some properties

- A logic program may have zero, one, or multiple stable models!
- If X is a stable model of a logic program P,
  then X is a model of P (seen as a propositional logic formula with negation instead of $\sim$)
- If X and Y are stable models of a normal program P,
  then $X \not\subset Y$

Outline

Motivation

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

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

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 (Clark) 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

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

An example

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

A closer look

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

$$
\begin{aligned}
\overleftarrow{CF}(P) &= \left\{ a \leftarrow \bigvee_{B \in body_P(a)} BF(B) \mid a \in atom(P) \right\} \\
\overrightarrow{CF}(P) &= \left\{ a \rightarrow \bigvee_{B \in body_P(a)} BF(B) \mid a \in atom(P) \right\} \\
body_P(a) &= \{ body(r) \mid r \in P \text{ and } head(r) = a \}
\end{aligned}
$$

A closer look

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

$$\begin{aligned}
\overleftarrow{CF}(P) &= \left\{a \leftarrow \bigvee\nolimits_{B \in body_P(a)} BF(B) \mid a \in atom(P)\right\} \\
\overrightarrow{CF}(P) &= \left\{a \rightarrow \bigvee\nolimits_{B \in body_P(a)} BF(B) \mid a \in atom(P)\right\} \\
body_P(a) &= \{body(r) \mid r \in P \text{ and } head(r) = a\}
\end{aligned}$$

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

A closer look

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

A closer look

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

A closer look

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

TECHNISCHE UNIVERSITÄT DRESDEN

A closer look

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

A closer look

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

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

A closer look

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

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

Supported models

- Every stable model of P is a model of $CF(P)$,

Supported models

- Every stable model of P is a model of $CF(P)$, but not vice versa

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

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

TECHNISCHE UNIVERSITÄT DRESDEN

An example

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

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

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 $\{a, c, e\}$

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

Motivation

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

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

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

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

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 $G(P)$

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 $(L, E \cap (L \times L))$

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 $loop(P)$

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 $G(P)$

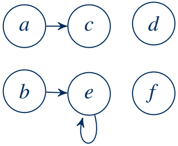
  That is, each pair of atoms in L is connected by a path of non-zero length in $(L, E \cap (L \times L))$

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

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

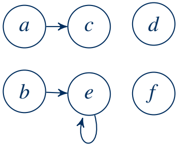
Example

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



Example

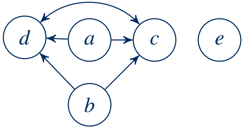
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$



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

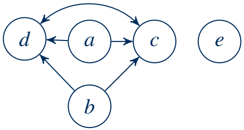
Another example

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



Another example

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



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

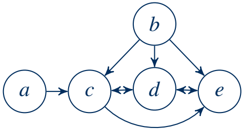
TECHNISCHE UNIVERSITÄT DRESDEN

Yet another example

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

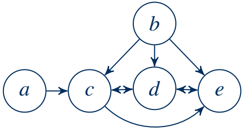
Yet another example

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



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

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

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 $EB_P(L) = body(ES_P(L))$

Loop formulas

Let P be a normal logic program

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

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

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

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

Loop formulas

Let P be a normal logic program

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

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

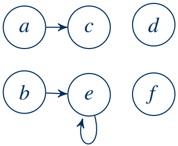
- Define the external bodies of L in P as $EB_P(L) = body(ES_P(L))$
- The (disjunctive) loop formula of L for P is

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

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

Example

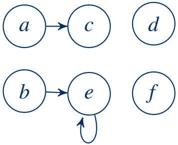
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \sim d & e \leftarrow b, \sim f \\ b \leftarrow \sim a & d \leftarrow \sim c, \sim e & e \leftarrow e \end{array} \right\}$



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

Example

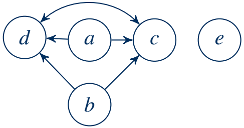
- $P = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, {\sim}d & e \leftarrow b, {\sim}f \\ b \leftarrow {\sim}a & d \leftarrow {\sim}c, {\sim}e & e \leftarrow e \end{array} \right\}$



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

Another example

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

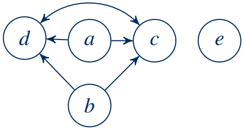


- $loop(P) = \{\{c, d\}\}$

TECHNISCHE UNIVERSITÄT DRESDEN

Another example

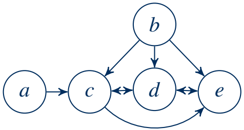
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a, b & d \leftarrow a & e \leftarrow {\sim}a, {\sim}b \\ b \leftarrow {\sim}a & c \leftarrow d & d \leftarrow b, c & \end{array} \right\}$



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

Yet another example

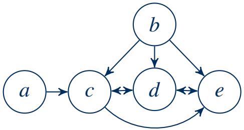
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, {\sim}a \\ b \leftarrow {\sim}a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



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

Yet another example

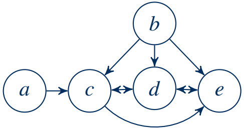
- $P = \left\{ \begin{array}{llll} a \leftarrow {\sim}b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, {\sim}a \\ b \leftarrow {\sim}a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



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

Yet another example

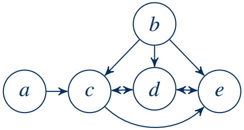
- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, \sim a \\ b \leftarrow \sim a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



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

Yet another example

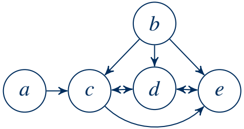
- $P = \left\{ \begin{array}{llll} a \leftarrow \sim b & c \leftarrow a & d \leftarrow b, c & e \leftarrow b, \sim a \\ b \leftarrow \sim a & c \leftarrow b, d & d \leftarrow e & e \leftarrow c, d \end{array} \right\}$



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

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

The following result is due to Fangzhen Lin and Yuting Zhao [2004], who used it to implement ASP using SAT solvers:

Theorem

Let P be a normal logic program and $X \subseteq atom(P)$
Then, X is a stable model of P iff $X \models CF(P) \cup LF(P)$

Note: There can be exponentially many loops in the worst case, so the reduction may incur a substantial blow-up. However, practical problems often include only a rather small number of loops.

Summary

Answer Set Programming is non-monotonic logic programming with a stable-model semantics

Main reasoning task: computing (all, zero or more) stable models (a.k.a. answer sets)

Reduction to SAT is possible by

- Clark completion (supported models) +
- Loop formulae (answer sets)