Answer Set Programming

Christoph Schwering

UNSW Sydney

COMP4418, Week 6

Contact

- Email: `c.schwering@unsw.edu.au`
- Research associate in the AI group
- Interests: Knowledge Representation and Reasoning
  - Formalisation knowledge, belief, actions, sensing
  - Tractable reasoning for highly expressive languages

ASP at a Glance

- ASP $=$ Answer Set Programming
  - ASP $\neq$ Microsoft's Active Server Pages

ASP at a Glance

- ASP $=$ Answer Set Programming
  - ASP $\neq$ Microsoft's Active Server Pages

- ASP belongs to logic programming
  - Like Prolog: $Head \leftarrow Body$ or $Head$ `:-` $Body$ `.`
  - Like Prolog: negation as failure
  - Unlike Prolog: $Head$ may be empty $\Rightarrow$ constraints

ASP at a Glance

- ASP $=$ Answer Set Programming
  - ASP $\neq$ Microsoft's Active Server Pages

- ASP belongs to logic programming
  - Like Prolog: $Head \leftarrow Body$ or $Head$ `:-` $Body$`.`
  - Like Prolog: negation as failure
  - Unlike Prolog: $Head$ may be empty $\Rightarrow$ constraints

- Declarative programming
  - Unlike Prolog: no procedural control
  - Order has no impact on semantics

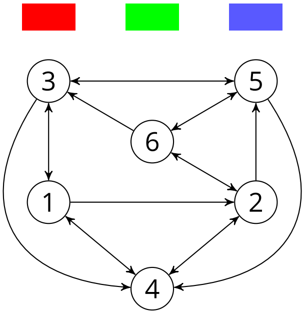
ASP at a Glance

- ASP = Answer Set Programming
  - ASP $\neq$ Microsoft's Active Server Pages
- ASP belongs to logic programming
  - Like Prolog: $Head \leftarrow Body$ or $Head$ `:-` $Body$ `.`
  - Like Prolog: negation as failure
  - Unlike Prolog: $Head$ may be empty $\Rightarrow$ constraints
- Declarative programming
  - Unlike Prolog: no procedural control
  - Order has no impact on semantics
- ASP programs compute *models*
  - Unlike Prolog: not query-oriented, no resolution
  - Unlike Prolog: not Turing-complete
  - Tool for problems in NP and NP^{NP} (common belief: $NP \subsetneq NP^{NP}$)

Example: Graph Colouring

Definition: graph colouring problem

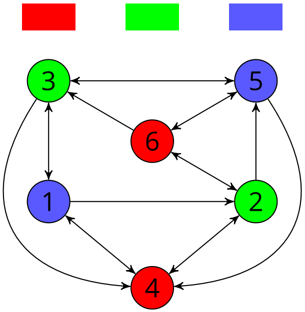
Input: graph with vertices V and edges $E \subseteq V \times V$, set of colors C.
Is there a mapping $m : V \to C$ with $m(x) \neq m(y)$ for all $(x, y) \in E$?



Example: Graph Colouring

Definition: graph colouring problem

Input: graph with vertices V and edges $E \subseteq V \times V$, set of colors C.
Is there a mapping $m : V \rightarrow C$ with $m(x) \neq m(y)$ for all $(x, y) \in E$?



Example: Graph Colouring

Definition: graph colouring problem

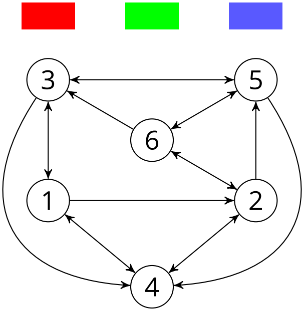
Input: graph with vertices V and edges $E \subseteq V \times V$, set of colors C.
Is there a mapping $m : V \to C$ with $m(x) \neq m(y)$ for all $(x, y) \in E$?

- Graph Coulouring is NP-complete
  - NP: guess solution, verify in polynomial time
  - NP-complete: among hardest in NP
- Many applications:
  - Mapping (neighbouring countries to different colors)
  - Scheduling (e.g., conflicting jobs to different time slots)
  - Allocation problems, Sudoku, ...

Example: Graph Colouring

Definition: graph colouring problem

Input: graph with vertices V and edges $E \subseteq V \times V$, set of colors C.
Is there a mapping $m : V \to C$ with $m(x) \neq m(y)$ for all $(x, y) \in E$?

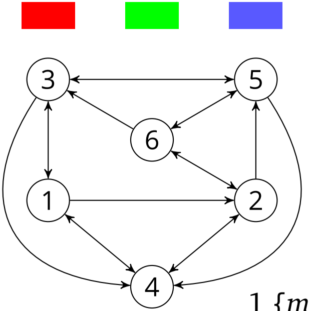


$c(r)$. $c(g)$. $c(b)$.
$v(1)$. ... $v(6)$.
$e(1,2)$. $e(1,3)$. $e(1,4)$.
$e(2,4)$. $e(2,5)$. $e(2,6)$.
$e(3,1)$. $e(3,4)$. $e(3,5)$.
$e(4,1)$. $e(4,2)$.
$e(5,3)$. $e(5,4)$. $e(5,6)$.
$e(6,2)$. $e(6,3)$. $e(6,5)$.

Example: Graph Colouring

Definition: graph colouring problem

Input: graph with vertices V and edges $E \subseteq V \times V$, set of colors C.
Is there a mapping $m : V \to C$ with $m(x) \neq m(y)$ for all $(x, y) \in E$?




```
c(r). c(g). c(b).
v(1). ... v(6).
e(1,2). e(1,3). e(1,4).
e(2,4). e(2,5). e(2,6).
e(3,1). e(3,4). e(3,5).
e(4,1). e(4,2).
e(5,3). e(5,4). e(5,6).
e(6,2). e(6,3). e(6,5).
```

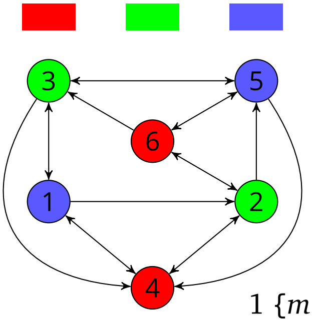
`1 {m(X,C) : c(C)} 1 :- v(X).` guess mapping m

`:- e(X,Y), m(X,C), m(Y,C).` verify $m(X) \neq m(Y)$

Example: Graph Colouring

Definition: graph colouring problem

Input: graph with vertices V and edges $E \subseteq V \times V$, set of colors C.
Is there a mapping $m : V \to C$ with $m(x) \neq m(y)$ for all $(x, y) \in E$?




```
c(r). c(g). c(b).
v(1). ... v(6).
e(1,2). e(1,3). e(1,4).
e(2,4). e(2,5). e(2,6).
e(3,1). e(3,4). e(3,5).
e(4,1). e(4,2).
e(5,3). e(5,4). e(5,6).
e(6,2). e(6,3). e(6,5).
```

```
1 {m(X,C) : c(C)} 1 :- v(X).        guess mapping m
:- e(X,Y), m(X,C), m(Y,C).          verify m(X) ≠ m(Y)
```

Applications of ASP

- Automated production configuration
- Decision-support system for space shuttle
- Bioinformatics (diagnosis, inconsistency detection)
- General game playing

- Several implementations are available
- For this lecture: **Clingo** `www.potassco.org`

Overview of the Lecture

- **Semantics of ASP programs**
- Extensions of ASP programs
- Handling of variables in ASP
- ASP as modelling language

Motivation

Consider the following logic program:

- $a.$
 $c \leftarrow a, b.$
 $d \leftarrow a, \text{not}\, b.$

```
a.
c :- a, b.
d :- a, not b.
```

Motivation

Consider the following logic program:

- $a.$
  $c \leftarrow a, b.$
  $d \leftarrow a, \mathrm{not}\, b.$
- Prolog proves by SLD resolution:

Motivation

Consider the following logic program:

- $a.$
  $c \leftarrow a, b.$
  $d \leftarrow a, \mathrm{not}\, b.$
- Prolog proves by SLD resolution:
  - Proves a (for a is a fact)

Motivation

Consider the following logic program:

- $a.$
  $c \leftarrow a, b.$
  $d \leftarrow a, \mathrm{not}\, b.$
- Prolog proves by SLD resolution:
  - Proves a (for a is a fact)
  - Cannot prove b (for b is in no head)

Motivation

Consider the following logic program:

- $a.$
  $c \leftarrow a, b.$
  $d \leftarrow a, \mathrm{not}\, b.$
- Prolog proves by SLD resolution:
  - Proves a (for a is a fact)
  - Cannot prove b (for b is in no head)
  - Cannot prove c (for cannot prove b)

Motivation

Consider the following logic program:

- $a.$
  $c \leftarrow a, b.$
  $d \leftarrow a, \mathrm{not}\, b.$
- Prolog proves by SLD resolution:
  - Proves a (for a is a fact)
  - Cannot prove b (for b is in no head)
  - Cannot prove c (for cannot prove b)
  - Proves d (for prove a but not b)

Motivation

Consider the following logic program:

- $a.$
  $c \leftarrow a, b.$
  $d \leftarrow a, \mathrm{not}\, b.$
- Prolog proves by SLD resolution:
  - Proves a (for a is a fact)
  - Cannot prove b (for b is in no head)
  - Cannot prove c (for cannot prove b)
  - Proves d (for prove a but not b)
- What is the *semantics* of this logic program?

Motivation

Consider the following logic program:

- $a.$ $\quad\quad$ a

  $c \leftarrow a, b.$ $\quad\quad$ $a \wedge b \to c$

  $d \leftarrow a, \mathrm{not}\, b.$ $\quad\quad$ $a \wedge \neg b \to d$
- Prolog proves by SLD resolution:
  - Proves a (for a is a fact)
  - Cannot prove b (for b is in no head)
  - Cannot prove c (for cannot prove b)
  - Proves d (for prove a but not b)
- What is the *semantics* of this logic program?
  - Models: $M_1 =$

a	b	c	d
1	0	0	1

$M_2 =$

a	b	c	d
1	1	1	0

$\ldots$

Motivation

Consider the following logic program:

- $a.$ $\quad$ a

  $c \leftarrow a, b.$ $\quad$ $a \wedge b \rightarrow c$

  $d \leftarrow a, \text{not}\, b.$ $\quad$ $a \wedge \neg b \rightarrow d$

- Prolog proves by SLD resolution:
  - Proves a (for a is a fact)
  - Cannot prove b (for b is in no head)
  - Cannot prove c (for cannot prove b)
  - Proves d (for prove a but not b)
- What is the *semantics* of this logic program?
  - Models: $M_1 =$

a	b	c	d
1	0	0	1

$M_2 =$

a	b	c	d
1	1	1	0

$\ldots$

  - M_1 corresponds to Prolog, what is special about M_1?

Motivation

Consider the following logic program:

- $a.$ $\quad\quad$ a
  $c \leftarrow a, b.$ $\quad\quad$ $a \wedge b \rightarrow c$
  $d \leftarrow a, \operatorname{not} b.$ $\quad\quad$ $a \wedge \neg b \rightarrow d$
- Prolog proves by SLD resolution:
  - Proves a (for a is a fact)
  - Cannot prove b (for b is in no head)
  - Cannot prove c (for cannot prove b)
  - Proves d (for prove a but not b)
- What is the *semantics* of this logic program?
  - Models: $M_1 =$

a	b	c	d
1	0	0	1

$M_2 =$

a	b	c	d
1	1	1	0

$\ldots$

  - M_1 corresponds to Prolog, what is special about M_1?
  - M_1 is a **stable model** a.k.a. **answer set**:

    M_1 only satisfies *justified* propositions

ASP gives **semantics** to **logic programming**

Intuition

The motivating guidelines behind stable model semantics are:

- A stable model satisfies all the rules of a logic program
- The reasoner shall not believe anything they are not forced to believe — the **rationality principle**

The rationality principle is related to *non-monotonic reasoning*:

- Closed-world assumption
- Autoepistemic logic
- Default logic

For now: only ground programs, i.e., no variables

Syntax

Definition: normal logic program (NLP)

A **normal logic program** P is a set of (normal) rules of the form

$$A \leftarrow B_1, \ldots, B_m, \mathrm{not}\, C_1, \ldots, \mathrm{not}\, C_n.$$

where A, B_i, C_j are atomic propositions.
When $m = n = 0$, we omit the "$\leftarrow$" and just write A.

Syntax

Definition: normal logic program (NLP)

A **normal logic program** P is a set of (normal) rules of the form

$$A \leftarrow B_1, \ldots, B_m, \mathrm{not}\, C_1, \ldots, \mathrm{not}\, C_n.$$

where A, B_i, C_j are atomic propositions.
When $m = n = 0$, we omit the "$\leftarrow$" and just write A.

For such a rule r, we define:

- $\mathrm{Head}(r) = \{A\}$
- $\mathrm{Body}(r) = \{B_1, \ldots, B_m, \mathrm{not}\, C_1, \ldots, \mathrm{not}\, C_n\}$

In code, r is written as A `:-` $B_1, \ldots, B_m$, `not` $C_1, \ldots,$ `not` C_n`.`

Semantics: Interpretation

Definition: partial interpretation, satisfaction

A **partial interpretation** S is a set of atomic propositions.

S **satisfies** $A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n$ iff
$A \in S$ or $B_i \notin S$ for some i or $C_j \in S$ for some j.

In English:

- S satisfies rule iff S satisfies the head or falsifies the body
- S falsifies body iff S falsifies some B_i or satisfies some C_j

Semantics: Interpretation

Definition: partial interpretation, satisfaction

A **partial interpretation** S is a set of atomic propositions.

S **satisfies** $A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n$ iff
$A \in S$ or $B_i \notin S$ for some i or $C_j \in S$ for some j.

In English:

- S satisfies rule iff S satisfies the head or falsifies the body
- S falsifies body iff S falsifies some B_i or satisfies some C_j

Ex.: Let $P = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \text{not}\, b.\}$

$S = \{a, b, c\}$ satisfies a, but it does not satisfy $(\text{not}\, b)$.

Semantics: Interpretation

Definition: partial interpretation, satisfaction

A **partial interpretation** S is a set of atomic propositions.

S **satisfies** $A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n$ iff
$A \in S$ or $B_i \notin S$ for some i or $C_j \in S$ for some j.

In English:

- S satisfies rule iff S satisfies the head or falsifies the body
- S falsifies body iff S falsifies some B_i or satisfies some C_j

Ex.: Let $P = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \text{not}\, b.\}$

$S = \{a, b, c\}$ satisfies a, but it does not satisfy $(\text{not}\, b)$.
$S = \{a, b, c\}$ satisfies $c \leftarrow a, b$ as well as $d \leftarrow a, \text{not}\, b$.

Semantics without Negation

Definition: stable model for programs without negation

For P without negated literals:
S is a **stable model** of P iff
S is a minimal set (w.r.t. $\subseteq$) that satisfies all $r \in P$.

Semantics without Negation

Definition: stable model for programs without negation

For P without negated literals:
S is a **stable model** of P iff
S is a minimal set (w.r.t. $\subseteq$) that satisfies all $r \in P$.

Ex.: $P = \{a. \quad c \leftarrow a, b.\}$

Semantics without Negation

Definition: stable model for programs without negation

For P without negated literals:
S is a **stable model** of P iff
S is a minimal set (w.r.t. $\subseteq$) that satisfies all $r \in P$.

Ex.: $P = \{a. \quad c \leftarrow a, b.\}$

$S_1 = \{a\}$ is a stable model of P

Semantics without Negation

Definition: stable model for programs without negation

For P without negated literals:
S is a **stable model** of P iff
S is a minimal set (w.r.t. $\subseteq$) that satisfies all $r \in P$.

Ex.: $P = \{a. \quad c \leftarrow a, b.\}$

$S_1 = \{a\}$ is a stable model of P
$S_2 = \{a, b\}$ is not a stable model of P

Semantics without Negation

Definition: stable model for programs without negation

For P without negated literals:
S is a **stable model** of P iff
S is a minimal set (w.r.t. $\subseteq$) that satisfies all $r \in P$.

Ex.: $P = \{a. \quad c \leftarrow a, b.\}$

$S_1 = \{a\}$ is a stable model of P
$S_2 = \{a, b\}$ is not a stable model of P
$S_3 = \{a, b, c\}$ is not a stable model of P

Semantics without Negation

Definition: stable model for programs without negation

For P without negated literals:
S is a **stable model** of P iff
S is a minimal set (w.r.t. $\subseteq$) that satisfies all $r \in P$.

Ex.: $P = \{a. \quad c \leftarrow a, b.\}$

$S_1 = \{a\}$ is a stable model of P
$S_2 = \{a, b\}$ is not a stable model of P
$S_3 = \{a, b, c\}$ is not a stable model of P

Theorem: unique-model property

If P is negation-free (i.e., contains no $(\mathrm{not}\, C)$), then there is exactly one stable model, which can be computed in linear time.

Semantics without Negation – Examples

The stable model of a negation-free program can be computed by *forward chaining*.

Semantics without Negation – Examples

The stable model of a negation-free program can be computed by *forward chaining*.

Ex.: $P_1 = \{a. \quad b \leftarrow a.\}$
$S^1 = \{a\}$
$S^2 = \{a, b\}$
Fixpoint

Semantics without Negation – Examples

The stable model of a negation-free program can be computed by *forward chaining*.

Ex.: $P_1 = \{a. \quad b \leftarrow a.\}$
$S^1 = \{a\}$
$S^2 = \{a, b\}$
Fixpoint

Ex.: $P_2 = \{a \leftarrow b. \quad b \leftarrow a.\}$
$S^1 = \{\}$
Fixpoint

Semantics without Negation – Examples

The stable model of a negation-free program can be computed by *forward chaining*.

Ex.: $P_1 = \{a. \quad b \leftarrow a.\}$
$S^1 = \{a\}$
$S^2 = \{a, b\}$
Fixpoint

Ex.: $P_2 = \{a \leftarrow b. \quad b \leftarrow a.\}$
$S^1 = \{\}$
Fixpoint

Ex.: $P_3 = \{a \leftarrow b. \quad b \leftarrow a. \quad a.\}$
$S^1 = \{a\}$
$S^2 = \{a, b\}$
Fixpoint

Semantics with Negation

Definition: reduct

The **reduct** P^S of P relative to S is the least set such that
if $A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n \in P$ and $C_1, \ldots, C_n \notin S$
then $A \leftarrow B_1, \ldots, B_m \in P^S$.

In English: for each rule r from P,

- if $(\text{not}\, C) \in \text{Body}(r)$ for some $C \in S$: drop it
- else: remove all negated literals and add to P^S

Semantics with Negation

Definition: reduct

The **reduct** P^S of P relative to S is the least set such that
if $A \leftarrow B_1, \ldots, B_m, \mathrm{not}\, C_1, \ldots, \mathrm{not}\, C_n \in P$ and $C_1, \ldots, C_n \notin S$
then $A \leftarrow B_1, \ldots, B_m \in P^S$.

In English: for each rule r from P,

- if $(\mathrm{not}\, C) \in \mathrm{Body}(r)$ for some $C \in S$: drop it
- else: remove all negated literals and add to P^S

Ex.: $P = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \mathrm{not}\, b.\}$

Semantics with Negation

Definition: reduct

The **reduct** P^S of P relative to S is the least set such that
if $A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n \in P$ and $C_1, \ldots, C_n \notin S$
then $A \leftarrow B_1, \ldots, B_m \in P^S$.

In English: for each rule r from P,

- if $(\text{not}\, C) \in \text{Body}(r)$ for some $C \in S$: drop it
- else: remove all negated literals and add to P^S

Ex.: $P = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \text{not}\, b.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \sout{\text{not}\, b}.\}$

Semantics with Negation

Definition: reduct

The **reduct** P^S of P relative to S is the least set such that
if $A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n \in P$ and $C_1, \ldots, C_n \notin S$
then $A \leftarrow B_1, \ldots, B_m \in P^S$.

In English: for each rule r from P,

- if $(\text{not}\, C) \in \text{Body}(r)$ for some $C \in S$: drop it
- else: remove all negated literals and add to P^S

Ex.: $P = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \text{not}\, b.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \cancel{\text{not}\, b}.\}$

$S_2 = \{a, b\} \quad \Rightarrow \quad P^{S_2} = \{a. \quad c \leftarrow a, b. \quad \cancel{d \leftarrow a, \text{not}\, b.}\}$

Semantics with Negation

Definition: reduct

The **reduct** P^S of P relative to S is the least set such that
if $A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n \in P$ and $C_1, \ldots, C_n \notin S$
then $A \leftarrow B_1, \ldots, B_m \in P^S$.

In English: for each rule r from P,

- if $(\text{not}\, C) \in \text{Body}(r)$ for some $C \in S$: drop it
- else: remove all negated literals and add to P^S

Ex.: $P = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \text{not}\, b.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \sout{\text{not}\, b}.\}$
$S_2 = \{a, b\} \quad \Rightarrow \quad P^{S_2} = \{a. \quad c \leftarrow a, b. \quad \sout{d \leftarrow a, \text{not}\, b.}\}$
$S_3 = \{a, d\} \quad \Rightarrow \quad P^{S_3} = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \sout{\text{not}\, b}.\}$

Semantics with Negation

Definition: reduct

The **reduct** P^S of P relative to S is the least set such that
if $A \leftarrow B_1, \ldots, B_m, \mathrm{not}\, C_1, \ldots, \mathrm{not}\, C_n \in P$ and $C_1, \ldots, C_n \notin S$
then $A \leftarrow B_1, \ldots, B_m \in P^S$.

In English: for each rule r from P,

- if $(\mathrm{not}\, C) \in \mathrm{Body}(r)$ for some $C \in S$: drop it
- else: remove all negated literals and add to P^S

Ex.: $P = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \mathrm{not}\, b.\}$

$$
\begin{aligned}
S_1 &= \{a\} & \Rightarrow \quad P^{S_1} &= \{a. \quad c \leftarrow a, b. \quad d \leftarrow a.\} \\
S_2 &= \{a, b\} & \Rightarrow \quad P^{S_2} &= \{a. \quad c \leftarrow a, b.\} \\
S_3 &= \{a, d\} & \Rightarrow \quad P^{S_3} &= \{a. \quad c \leftarrow a, b. \quad d \leftarrow a.\}
\end{aligned}
$$

Semantics with Negation

Definition: reduct

The **reduct** P^S of P relative to S is the least set such that
if $A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n \in P$ and $C_1, \ldots, C_n \notin S$
then $A \leftarrow B_1, \ldots, B_m \in P^S$.

In English: for each rule r from P,

- if $(\text{not}\, C) \in \text{Body}(r)$ for some $C \in S$: drop it
- else: remove all negated literals and add to P^S

Ex.: $P = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \text{not}\, b.\}$

$$
\begin{array}{lcl}
S_1 = \{a\} & \Rightarrow & P^{S_1} = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a.\} \\
S_2 = \{a, b\} & \Rightarrow & P^{S_2} = \{a. \quad c \leftarrow a, b.\} \\
S_3 = \{a, d\} & \Rightarrow & P^{S_3} = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a.\}
\end{array}
$$

Definition: stable model for programs with negation

For P with negated literals:
S is a **stable model** of P iff S is a stable model of P^S.

Semantics with Negation

Definition: reduct

The **reduct** P^S of P relative to S is the least set such that
if $A \leftarrow B_1, \ldots, B_m, \text{not } C_1, \ldots, \text{not } C_n \in P$ and $C_1, \ldots, C_n \notin S$
then $A \leftarrow B_1, \ldots, B_m \in P^S$.

In English: for each rule r from P,

- if $(\text{not } C) \in \text{Body}(r)$ for some $C \in S$: drop it
- else: remove all negated literals and add to P^S

Ex.: $P = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \text{not } b.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a.\}$ ✗
$S_2 = \{a, b\} \quad \Rightarrow \quad P^{S_2} = \{a. \quad c \leftarrow a, b.\}$
$S_3 = \{a, d\} \quad \Rightarrow \quad P^{S_3} = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a.\}$

Definition: stable model for programs with negation

For P with negated literals:
S is a **stable model** of P iff S is a stable model of P^S.

Semantics with Negation

Definition: reduct

The **reduct** P^S of P relative to S is the least set such that
if $A \leftarrow B_1, \ldots, B_m, \mathrm{not}\, C_1, \ldots, \mathrm{not}\, C_n \in P$ and $C_1, \ldots, C_n \notin S$
then $A \leftarrow B_1, \ldots, B_m \in P^S$.

In English: for each rule r from P,

- if $(\mathrm{not}\, C) \in \mathrm{Body}(r)$ for some $C \in S$: drop it
- else: remove all negated literals and add to P^S

Ex.: $P = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \mathrm{not}\, b.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a.\}$ ✗

$S_2 = \{a, b\} \quad \Rightarrow \quad P^{S_2} = \{a. \quad c \leftarrow a, b.\}$ ✗

$S_3 = \{a, d\} \quad \Rightarrow \quad P^{S_3} = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a.\}$

Definition: stable model for programs with negation

For P with negated literals:
S is a **stable model** of P iff S is a stable model of P^S.

Semantics with Negation

Definition: reduct

The **reduct** P^S of P relative to S is the least set such that
if $A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n \in P$ and $C_1, \ldots, C_n \notin S$
then $A \leftarrow B_1, \ldots, B_m \in P^S$.

In English: for each rule r from P,

- if $(\text{not}\, C) \in \text{Body}(r)$ for some $C \in S$: drop it
- else: remove all negated literals and add to P^S

Ex.: $P = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a, \text{not}\, b.\}$

$S_1 = \{a\} \Rightarrow P^{S_1} = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a.\}$ ✗

$S_2 = \{a, b\} \Rightarrow P^{S_2} = \{a. \quad c \leftarrow a, b.\}$ ✗

$S_3 = \{a, d\} \Rightarrow P^{S_3} = \{a. \quad c \leftarrow a, b. \quad d \leftarrow a.\}$ ✓

Definition: stable model for programs with negation

For P with negated literals:
S is a **stable model** of P iff S is a stable model of P^S.

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} =$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$S_1 = \{a\} \qquad \Rightarrow \quad P^{S_1} = \{a \leftarrow \sout{\text{not}\, b}. \quad \sout{b \leftarrow \text{not}\, a.}\}$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a \leftarrow \sout{\text{not}\, b}. \quad \sout{b \leftarrow \text{not}\, a.}\}$

$S_2 = \{b\} \quad \Rightarrow \quad P^{S_2} =$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a \leftarrow \sout{\text{not}\, b}. \quad \sout{b \leftarrow \text{not}\, a.}\}$$
$$S_2 = \{b\} \quad \Rightarrow \quad P^{S_2} = \{\sout{a \leftarrow \text{not}\, b.} \quad b \leftarrow \sout{\text{not}\, a}.\}$$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$$
\begin{aligned}
S_1 = \{a\} &\Rightarrow P^{S_1} = \{a \leftarrow \sout{\text{not}\, b}. \quad \sout{b \leftarrow \text{not}\, a.}\} \\
S_2 = \{b\} &\Rightarrow P^{S_2} = \{\sout{a \leftarrow \text{not}\, b.} \quad b \leftarrow \sout{\text{not}\, a}.\} \\
S_3 = \{a, b\} &\Rightarrow P^{S_3} =
\end{aligned}
$$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$$
\begin{array}{lcl}
S_1 = \{a\} & \Rightarrow & P^{S_1} = \{a \leftarrow \cancel{\text{not}\, b}. \quad \cancel{b \leftarrow \text{not}\, a.}\} \\
S_2 = \{b\} & \Rightarrow & P^{S_2} = \{\cancel{a \leftarrow \text{not}\, b.} \quad b \leftarrow \cancel{\text{not}\, a}.\} \\
S_3 = \{a, b\} & \Rightarrow & P^{S_3} = \{\cancel{a \leftarrow \text{not}\, b.} \quad \cancel{b \leftarrow \text{not}\, a.}\}
\end{array}
$$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$$
\begin{array}{lcl}
S_1 = \{a\} & \Rightarrow & P^{S_1} = \{a.\} \\
S_2 = \{b\} & \Rightarrow & P^{S_2} = \{b.\} \\
S_3 = \{a, b\} & \Rightarrow & P^{S_3} = \{\}
\end{array}
$$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$$
\begin{array}{llll}
S_1 = \{a\} & \Rightarrow & P^{S_1} = \{a.\} & \checkmark \\
S_2 = \{b\} & \Rightarrow & P^{S_2} = \{b.\} & \\
S_3 = \{a, b\} & \Rightarrow & P^{S_3} = \{\} &
\end{array}
$$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$$S_1 = \{a\} \Rightarrow P^{S_1} = \{a.\} \quad ✓$$
$$S_2 = \{b\} \Rightarrow P^{S_2} = \{b.\} \quad ✓$$
$$S_3 = \{a, b\} \Rightarrow P^{S_3} = \{\}$$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a.\}$ ✓

$S_2 = \{b\} \quad \Rightarrow \quad P^{S_2} = \{b.\}$ ✓

$S_3 = \{a, b\} \quad \Rightarrow \quad P^{S_3} = \{\}$ ✗

Two stable models!

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a.\}$ ✓

$S_2 = \{b\} \quad \Rightarrow \quad P^{S_2} = \{b.\}$ ✓

$S_3 = \{a, b\} \quad \Rightarrow \quad P^{S_3} = \{\}$ ✗

Two stable models!

Ex.: $P = \{a \leftarrow \text{not}\, a.\}$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a.\}$ ✓

$S_2 = \{b\} \quad \Rightarrow \quad P^{S_2} = \{b.\}$ ✓

$S_3 = \{a, b\} \quad \Rightarrow \quad P^{S_3} = \{\}$ ✗

Two stable models!

Ex.: $P = \{a \leftarrow \text{not}\, a.\}$

$S_1 = \{\} \quad \Rightarrow \quad P^{S_1} =$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a.\}$ ✓

$S_2 = \{b\} \quad \Rightarrow \quad P^{S_2} = \{b.\}$ ✓

$S_3 = \{a, b\} \quad \Rightarrow \quad P^{S_3} = \{\}$ ✗

Two stable models!

Ex.: $P = \{a \leftarrow \text{not}\, a.\}$

$S_1 = \{\} \quad \Rightarrow \quad P^{S_1} = \{a \leftarrow \sout{\text{not}\, a}.\}$

Semantics with Negation – Examples

<u>Ex.</u>: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a.\}$ ✓

$S_2 = \{b\} \quad \Rightarrow \quad P^{S_2} = \{b.\}$ ✓

$S_3 = \{a, b\} \quad \Rightarrow \quad P^{S_3} = \{\}$ ✗

Two stable models!

<u>Ex.</u>: $P = \{a \leftarrow \text{not}\, a.\}$

$S_1 = \{\} \quad \Rightarrow \quad P^{S_1} = \{a \leftarrow \cancel{\text{not}\, a}.\}$

$S_2 = \{a\} \quad \Rightarrow \quad P^{S_2} =$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a.\}$ ✓

$S_2 = \{b\} \quad \Rightarrow \quad P^{S_2} = \{b.\}$ ✓

$S_3 = \{a, b\} \quad \Rightarrow \quad P^{S_3} = \{\}$ ✗

Two stable models!

Ex.: $P = \{a \leftarrow \text{not}\, a.\}$

$S_1 = \{\} \quad \Rightarrow \quad P^{S_1} = \{a \leftarrow \sout{\text{not}\, a}.\}$

$S_2 = \{a\} \quad \Rightarrow \quad P^{S_2} = \{\sout{a \leftarrow \text{not}\, a.}\}$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$S_1 = \{a\} \Rightarrow P^{S_1} = \{a.\}$ ✓

$S_2 = \{b\} \Rightarrow P^{S_2} = \{b.\}$ ✓

$S_3 = \{a, b\} \Rightarrow P^{S_3} = \{\}$ ✗

Two stable models!

Ex.: $P = \{a \leftarrow \text{not}\, a.\}$

$S_1 = \{\} \Rightarrow P^{S_1} = \{a.\}$

$S_2 = \{a\} \Rightarrow P^{S_2} = \{\}$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a.\}$ ✓

$S_2 = \{b\} \quad \Rightarrow \quad P^{S_2} = \{b.\}$ ✓

$S_3 = \{a, b\} \quad \Rightarrow \quad P^{S_3} = \{\}$ ✗

Two stable models!

Ex.: $P = \{a \leftarrow \text{not}\, a.\}$

$S_1 = \{\} \quad \Rightarrow \quad P^{S_1} = \{a.\}$ ✗

$S_2 = \{a\} \quad \Rightarrow \quad P^{S_2} = \{\}$

Semantics with Negation – Examples

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, a.\}$

$S_1 = \{a\} \quad \Rightarrow \quad P^{S_1} = \{a.\}$ ✓

$S_2 = \{b\} \quad \Rightarrow \quad P^{S_2} = \{b.\}$ ✓

$S_3 = \{a, b\} \quad \Rightarrow \quad P^{S_3} = \{\}$ ✗

Two stable models!

Ex.: $P = \{a \leftarrow \text{not}\, a.\}$

$S_1 = \{\} \quad \Rightarrow \quad P^{S_1} = \{a.\}$ ✗

$S_2 = \{a\} \quad \Rightarrow \quad P^{S_2} = \{\}$ ✗

No stable model!

Semantics: Overview

Definition: reduct

The **reduct** P^S of P relative to S is the least set such that
if $A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n \in P$ and $C_1, \ldots, C_n \notin S$
then $A \leftarrow B_1, \ldots, B_m \in P^S$.

Definition: stable model

If P contains no $(\text{not}\, C)$:
S is a **stable model** of P iff
S is a minimal set (w.r.t. $\subseteq$) that satisfies all $r \in P$.

If P contains $(\text{not}\, C)$:
S is a **stable model** of P iff S is a stable model of P^S.

Theorem: necessary satisfaction condition

If S is a stable model and $A \in S$, then S satisfies some $r \in P$ with $A \in \text{Head}(r)$.

Semantics – Examples

Ex.: $P = \{a \leftarrow a. \quad b \leftarrow \text{not}\, a.\}$

S $\qquad$ P^S $\qquad$ Stable model?

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow \text{not}\, c.\}$

S $\qquad$ P^S $\qquad$ Stable model?

Example on paper

Entailment

Definition: entailment, cautious monotonicity

P **entails** a rule r iff every stable model of P satisfies r.

P is **cautiously monotonic** iff
for all rules r_1, r_2, if P entails r_1 and r_2, then $P \cup \{r_1\}$ entails r_2.

Entailment

Definition: entailment, cautious monotonicity

P **entails** a rule r iff every stable model of P satisfies r.
P is **cautiously monotonic** iff
for all rules r_1, r_2, if P entails r_1 and r_2, then $P \cup \{r_1\}$ entails r_2.

If P is cautiously monotonic, a solver can iteratively augment it with already proved lemmas.

Entailment

Definition: entailment, cautious monotonicity

P **entails** a rule r iff every stable model of P satisfies r.
P is **cautiously monotonic** iff
for all rules r_1, r_2, if P entails r_1 and r_2, then $P \cup \{r_1\}$ entails r_2.

If P is cautiously monotonic, a solver can iteratively augment it with already proved lemmas. *Bad news*: it does **not** hold in general.

Entailment

Definition: entailment, cautious monotonicity

P **entails** a rule r iff every stable model of P satisfies r.
P is **cautiously monotonic** iff
for all rules r_1, r_2, if P entails r_1 and r_2, then $P \cup \{r_1\}$ entails r_2.

If P is cautiously monotonic, a solver can iteratively augment it with already proved lemmas. *Bad news*: it does **not** hold in general.

Ex.: $P = \{a \leftarrow \mathrm{not}\, b. \quad b \leftarrow c, \mathrm{not}\, a. \quad c \leftarrow a.\}$

$S_1 = \{a, c\} \;\Rightarrow\; P^{S_1} = \{a. \quad c \leftarrow a.\}$ ✓

(no other stable model S: $b \notin S \Rightarrow a \in S \Rightarrow c \in S$ and $b \in S \Rightarrow c \in S \Rightarrow a \in S \Rightarrow b \notin S$)

Entailment

Definition: entailment, cautious monotonicity

P **entails** a rule r iff every stable model of P satisfies r.
P is **cautiously monotonic** iff
for all rules r_1, r_2, if P entails r_1 and r_2, then $P \cup \{r_1\}$ entails r_2.

If P is cautiously monotonic, a solver can iteratively augment it with already proved lemmas. *Bad news*: it does **not** hold in general.

Ex.: $P = \{a \leftarrow \mathrm{not}\, b. \quad b \leftarrow c, \mathrm{not}\, a. \quad c \leftarrow a.\}$

$S_1 = \{a, c\} \;\Rightarrow\; P^{S_1} = \{a. \quad c \leftarrow a.\}$ ✓

(no other stable model S: $b \notin S \Rightarrow a \in S \Rightarrow c \in S$ and $b \in S \Rightarrow c \in S \Rightarrow a \in S \Rightarrow b \notin S$)

$S_1 = \{a, c\} \;\Rightarrow\; (P \cup \{c.\})^{S_1} = \{a. \quad c \leftarrow a. \quad c.\}$ ✓
$S_2 = \{b, c\} \;\Rightarrow\; (P \cup \{c.\})^{S_2} = \{b \leftarrow c. \quad c \leftarrow a. \quad c.\}$ ✓

(no other stable model S: $c \in S$ and $a \notin S \Rightarrow b \in S$ and $b \notin S \Rightarrow a \in S$)

Entailment

Definition: entailment, cautious monotonicity

P **entails** a rule r iff every stable model of P satisfies r.
P is **cautiously monotonic** iff
for all rules r_1, r_2, if P entails r_1 and r_2, then $P \cup \{r_1\}$ entails r_2.

If P is cautiously monotonic, a solver can iteratively augment it with already proved lemmas. *Bad news*: it does **not** hold in general.

Ex.: $P = \{a \leftarrow \text{not}\, b. \quad b \leftarrow c, \text{not}\, a. \quad c \leftarrow a.\}$

$S_1 = \{a, c\} \;\Rightarrow\; P^{S_1} = \{a. \quad c \leftarrow a.\}$ ✓

(no other stable model S: $b \notin S \Rightarrow a \in S \Rightarrow c \in S$ and $b \in S \Rightarrow c \in S \Rightarrow a \in S \Rightarrow b \notin S$)

$S_1 = \{a, c\} \;\Rightarrow\; (P \cup \{c.\})^{S_1} = \{a. \quad c \leftarrow a. \quad c.\}$ ✓
$S_2 = \{b, c\} \;\Rightarrow\; (P \cup \{c.\})^{S_2} = \{b \leftarrow c. \quad c \leftarrow a. \quad c.\}$ ✓

(no other stable model S: $c \in S$ and $a \notin S \Rightarrow b \in S$ and $b \notin S \Rightarrow a \in S$)

Good news: some classes of programs are cautiously monotonic.

Overview of the Lecture

- Semantics of ASP programs
- **Extensions of ASP programs**
- Handling of variables in ASP
- ASP as modelling language

Integrity Constraints

Definition: integrity constraint

An **integrity constraint** is a rule r of the form

$$\leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n$$

S **satisfies** r iff $B_i \notin S$ for some i or $C_j \in S$ for some j.
P^S contains $\leftarrow B_1, \ldots, B_m$ iff P contains r and $C_1, \ldots, C_n \notin S$.

Integrity Constraints

Definition: integrity constraint

An **integrity constraint** is a rule r of the form

$$\leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n$$

S **satisfies** r iff $B_i \notin S$ for some i or $C_j \in S$ for some j.
P^S contains $\leftarrow B_1, \ldots, B_m$ iff P contains r and $C_1, \ldots, C_n \notin S$.

Theorem: reduction to normal rules

Let P' be like P except that every integrity constraint

$$\leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n$$

is replaced with

$$dummy \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n, \text{not}\, dummy$$

for some new atom $dummy$.
Then P and P' have the same stable models.

Proof on paper

Choice Rules

Definition: choice rule

A **choice rule** is a rule the form

$$\{A_1, \ldots, A_k\} \leftarrow B_1, \ \ldots, B_m, \text{not}\, C_1, \ \ldots, \text{not}\, C_n$$

which allows any subset of $\{A_1, \ldots, A_k\}$ in a stable model.

Choice Rules

Definition: choice rule

A **choice rule** is a rule the form

$$\{A_1, \ldots, A_k\} \leftarrow B_1,\ \ldots, B_m, \text{not}\, C_1,\ \ldots, \text{not}\, C_n$$

which allows any subset of $\{A_1, \ldots, A_k\}$ in a stable model.

Theorem: reduction to normal rules

A choice rule can be encoded by $2k + 1$ normal rules using $2k + 1$ new atoms.

Choice Rules

Definition: choice rule

A **choice rule** is a rule the form

$$\{A_1, \ldots, A_k\} \leftarrow B_1, \ \ldots, B_m, \text{not}\, C_1, \ \ldots, \text{not}\, C_n$$

which allows any subset of $\{A_1, \ldots, A_k\}$ in a stable model.

Theorem: reduction to normal rules

A choice rule can be encoded by $2k + 1$ normal rules using $2k + 1$ new atoms.

Further extensions:

- Conditional literals: $\{A : B\}$
  Ex.: $\{m(v, C) : c(C)\}$ expands to $\{m(v, r), m(v, g), m(v, b)\}$
- Cardinality constraints: $min\ \{A_1, \ldots, A_k\}\ max$
  Ex.: $1\ \{m(v, r), m(v, g), m(v, b)\}\ 1$

Negation in the Rule Head

Definition: rules with negated head

A rule with **negated head** is of the form

$$\text{not}\, A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n$$

Negation in the Rule Head

Definition: rules with negated head

A rule with **negated head** is of the form

$$\text{not}\, A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n$$

Theorem: reduction to normal rules

Let P' be like P except that every rule with negated head

$$\text{not}\, A \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n$$

is replaced with

$$\leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n, \text{not}\, \mathit{dummy}$$

and

$$\mathit{dummy} \leftarrow \text{not}\, A$$

for some new atom *dummy*.

Then P and P' have the same stable models (modulo dummy propositions).

Complexity

Theorem: complexity of NLPs without negations

Is S a stable model of a negation-free P? – **Linear time**
Does a negation-free P have a stable model? – **Yes**

Theorem: complexity of NLPs with negations

Is S a stable model of P? – **Linear time**
Does P have a stable model? – **NP-complete**

Note: integrity constraints, choice rules, conditional literals, cardinality constraints, negation in heads **preserve complexity**

Disjunctive Logic Programs

Definition: disjunctive rule

A **disjunctive rule** is of the form

$$A_1; \ldots; A_k \leftarrow B_1, \ldots, B_m, \mathrm{not}\, C_1, \ldots, \mathrm{not}\, C_n$$

and means that A_1 or A_2 or $\ldots$ or A_k is true if the body is true.

Disjunctive Logic Programs

Definition: disjunctive rule

A **disjunctive rule** is of the form

$$A_1; \ldots; A_k \leftarrow B_1, \ldots, B_m, \text{not}\, C_1, \ldots, \text{not}\, C_n$$

and means that A_1 or A_2 or $\ldots$ or A_k is true if the body is true.

Theorem: complexity of disjunctive logic programs

Is S a stable model of P? – **co-NP-complete**
Does P have a stable model? – **$\mathbf{NP^{NP}}$-complete**

Reason: P^S may have multiple minimal models!

We won't consider disjunctive logic problems any further

Overview of the Lecture

- Semantics of ASP programs
- Extensions of ASP programs
- **Handling of variables in ASP**
- ASP as modelling language

Programs with Variables

- Atomic propositions may now contain variables, e.g.,
  $p(X, Z) \leftarrow e(X, Y), p(Y, Z)$.

Programs with Variables

- Atomic propositions may now contain variables, e.g., $p(X, Z) \leftarrow e(X, Y), p(Y, Z)$.
- Herbrand universe
  - U contains all constants from P
  - U contains all $f(t_1, \ldots, t_k)$ from P if f is a k-ary function in P and U contains $t_1, \ldots, t_k$

Programs with Variables

- Atomic propositions may now contain variables, e.g., $p(X,Z) \leftarrow e(X,Y), p(Y,Z)$.
- Herbrand universe
  - U contains all constants from P
  - U contains all $f(t_1, \ldots, t_k)$ from P if f is a k-ary function in P and U contains $t_1, \ldots, t_k$
- ASP **grounds** variables with Herbrand universe
  - Unlike Prolog: instantiation instead of unification
  - Caution: the ground program may grow exponentially
  - Caution: function symbols make grounding Turing-complete
  - If P is finite and mentions only constants, grounding is finite

Programs with Variables

- $f(X) \leftarrow b(X), \text{not}\, a(X).$
  $a(X) \leftarrow p(X).$
  $b(\text{sam}).$
  $b(\text{tweety.})$
  $p(\text{tweety}).$

- $f(\text{sam}) \leftarrow b(\text{sam}), \text{not}\, a(\text{sam}).$
  $f(\text{tweety}) \leftarrow b(\text{tweety}), \text{not}\, a(\text{tweety}).$
  $a(\text{sam}) \leftarrow p(\text{sam}).$
  $a(\text{tweety}) \leftarrow p(\text{tweety}).$
  $b(\text{sam}).$
  $b(\text{tweety.})$
  $p(\text{tweety.})$

Overview of the Lecture

- Semantics of ASP programs
- Extensions of ASP programs
- Handling of variables in ASP
- **ASP as modelling language**

ASP Modeling

$c(r).\ c(g).\ c(b).$
$v(1).\ \ldots\ v(6).$
$e(1,2).\ e(1,3).\ e(1,4).$
$e(2,4).\ e(2,5).\ e(2,6).$
$e(3,1).\ e(3,4).\ e(3,5).$
$e(4,1).\ e(4,2).$
$e(5,3).\ e(5,4).\ e(5,6).$
$e(6,2).\ e(6,3).\ e(6,5).$

Typical ASP structure:

- Problem **instance**: a set of facts
- Problem **class**: a set of rules
  - Generator rules: often choice rules
  - Test rules: often integrity constraints

$1\ \{m(X,C) : c(C)\}\ 1 \text{ :- } v(X).$

$\text{:- } e(X,Y), m(X,C), m(Y,C).$

Ideal modeling is **uniform**: problem class encoding fits all instances

Semantically equivalent encodings may differ immensely in performance!

Example: Non-monotonic Reasoning

Tweety the penguin:

- Normal birds fly.
- Penguins are abnormal.
- Tweety is a bird. So Tweety flies.
- Tweety is a penguin. So Tweety doesn't fly.

Example: Non-monotonic Reasoning

Tweety the penguin:

- Normal birds fly.
- Penguins are abnormal.
- Tweety is a bird. So Tweety flies.
- Tweety is a penguin. So Tweety doesn't fly.

Ex.: $U_1 = \{f(X) \leftarrow b(X), \mathrm{not}\, a(X). \quad a(X) \leftarrow p(X). \quad b(t).\}$
$P_1 = \{f(t) \leftarrow b(t), \mathrm{not}\, a(t). \quad a(t) \leftarrow p(t). \quad b(t).\}$
$S_1 = \{b(t), f(t)\} \;\Rightarrow\; P_1^{S_1} = \{f(t) \leftarrow b(t). \quad a(t) \leftarrow p(t). \quad b(t).\}$ ✓

Example: Non-monotonic Reasoning

Tweety the penguin:

- Normal birds fly.
- Penguins are abnormal.
- Tweety is a bird. So Tweety flies.
- Tweety is a penguin. So Tweety doesn't fly.

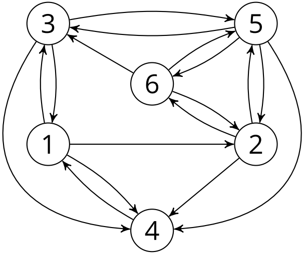
<u>Ex.</u>: $U_1 = \{f(X) \leftarrow b(X), \text{not}\, a(X). \quad a(X) \leftarrow p(X). \quad b(t).\}$
$P_1 = \{f(t) \leftarrow b(t), \text{not}\, a(t). \quad a(t) \leftarrow p(t). \quad b(t).\}$
$S_1 = \{b(t), f(t)\} \;\Rightarrow\; P_1^{S_1} = \{f(t) \leftarrow b(t). \quad a(t) \leftarrow p(t). \quad b(t).\}$ ✓

<u>Ex.</u>: $U_2 = U_1 \cup \{p(t).\}$
$P_2 = P_1 \cup \{p(t).\}$
$S_2 = \{a(t), b(t), p(t)\} \;\Rightarrow\; P_2^{S_2} = \{a(t) \leftarrow p(t). \quad b(t). \quad p(t).\}$ ✓

Example: Hamilton Cycle

Definition: Hamilton cycle problem

Input: graph with vertex set V and edges $E \subseteq V \times V$.
Is there a cycle that visits every vertex exactly once?

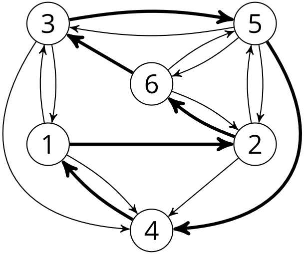


$$\{p(X,Y)\} \leftarrow e(X,Y).$$
$$r(X) \leftarrow p(1,X).$$
$$r(Y) \leftarrow r(X), p(X,Y).$$
$$\leftarrow 2\ \{p(X,Y)\}\ , v(X).$$
$$\leftarrow 2\ \{p(X,Y)\}\ , v(Y).$$
$$\leftarrow \text{not}\, r(X), v(X).$$

Example: Hamilton Cycle

Definition: Hamilton cycle problem

Input: graph with vertex set V and edges $E \subseteq V \times V$.
Is there a cycle that visits every vertex exactly once?

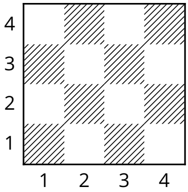


$$\{p(X,Y)\} \leftarrow e(X,Y).$$
$$r(X) \leftarrow p(1,X).$$
$$r(Y) \leftarrow r(X), p(X,Y).$$
$$\leftarrow 2\ \{p(X,Y)\}\ , v(X).$$
$$\leftarrow 2\ \{p(X,Y)\}\ , v(Y).$$
$$\leftarrow \text{not}\, r(X), v(X).$$

Example: N-Queens

Definition: N-queens problem

Place N queens on a $N \times N$ chessboard so that they do not attack each other, i.e., share no row, column, or diagonal.

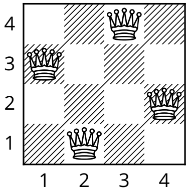


Program on paper

Example: N-Queens

Definition: N-queens problem

Place N queens on a $N \times N$ chessboard so that they do not attack each other, i.e., share no row, column, or diagonal.



Program on paper