Answer Set Solving in Practice

Torsten Schaub[1]
University of Potsdam
torsten@cs.uni-potsdam.de

[1]Standing on the shoulders of a great research group and community!

Organization: Overview

Potassco

Outline

1. Roadmap
2. Resources
3. Literature
4. Systems

What's on the menu?

Outline

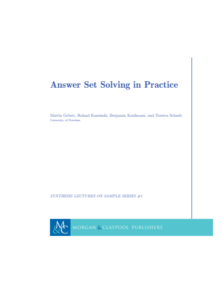
Resources

- Course material
  - https://github.com/potassco-asp-course
  - https://potassco.org/teaching
- Videos
  - https://youtube.com/c/potassco-live
- Mailing lists
  - https://sourceforge.net/projects/potassco/lists/potassco-users
  - https://sourceforge.net/projects/potassco/lists/potassco-announce
- Contact
  - asp@lists.cs.uni-potsdam.de

Outline

The Potassco Book

1. Motivation
2. Introduction
3. Basic modeling
4. Grounding
5. Characterizations
6. Solving
7. Systems
8. Advanced modeling
9. Conclusions



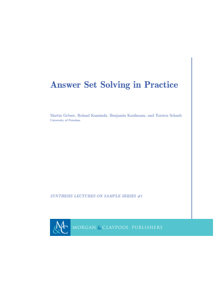
Resources

- http://potassco.org/book
- http://potassco.org/teaching

Potassco

The Potassco Book

1. Motivation
2. Introduction
3. Basic modeling
4. Grounding
5. Characterizations
6. Solving
7. Systems
8. Advanced modeling
9. Conclusions

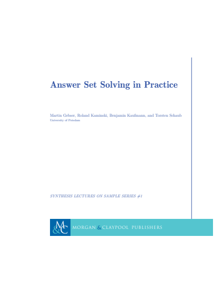


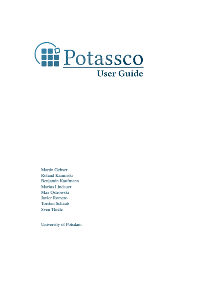
Resources

- `http://potassco.org/book`
- `http://potassco.org/teaching`

Potassco

The Potassco Book and Guide

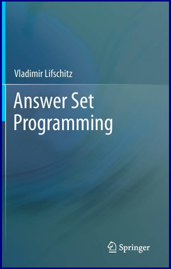


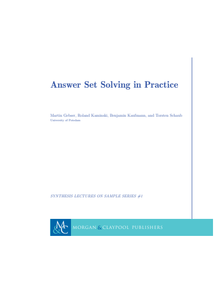


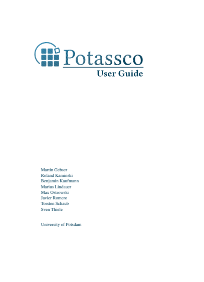
Resources

- http://potassco.org/book
- http://potassco.org/teaching

The ASP and Potassco Book, and Guide







Resources

- http://potassco.org/book
- http://potassco.org/teaching

Literature

- Books [5], [25], [32], [38], [43]
- Surveys [40], [31], [19], [10], [36], [47]
- Magazines [9], [48]
- Articles [35], [34], [7], [45], [44], [39], [33], [23], etc.
- Guide [29]

- See also our bibliography on github

Literature

- Books [5], [25], [32], [38], [43]
- Surveys [40], [31], [19], [10], [36], [47]
- Magazines [9], [48]
- Articles [35], [34], [7], [45], [44], [39], [33], [23], etc.
- Guide [29]

- See also our bibliography on github

Outline

Potassco

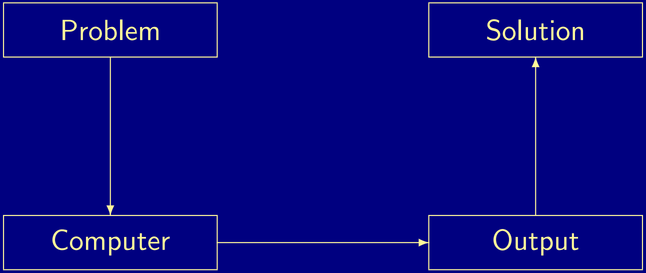
Systems

- Systems
  - *clingo* [27] https://potassco.org
  - *dlv* [37, 2] http://www.dlvsystem.com
- Grounders
  - *lparse* [50]
  - *gringo* [22, 28]+[24, 12] https://potassco.org
  - *idlv* [14]+[12] http://www.dlvsystem.com
- Solvers
  - *smodels* [46, 49]
  - *clasp* [26, 21] https://potassco.org
  - *wasp* [4] https://www.mat.unical.it/ricca/wasp
- Encodings
  - *asparagus* [8] https://asparagus.cs.uni-potsdam.de
  - competitions [30, 18, 16, 3, 15]

Motivation: Overview

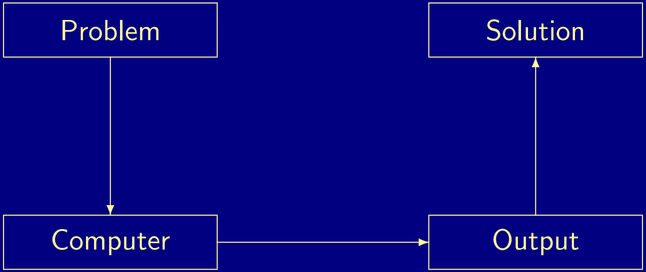
Outline

Informatics



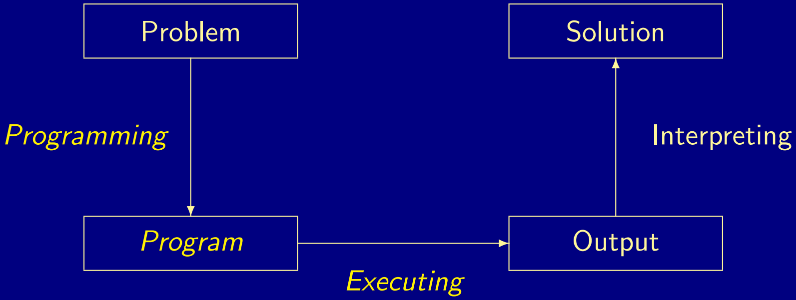
Traditional programming

"How to solve the problem?"



Traditional programming

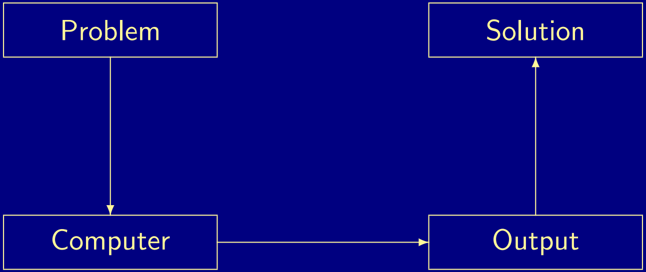
"How to solve the problem?"



Potassco

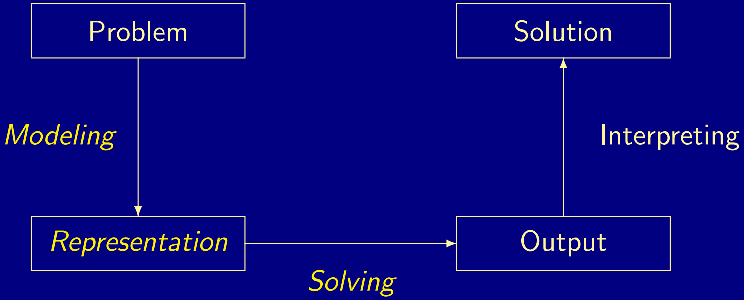
Declarative programming

"What is the problem?"



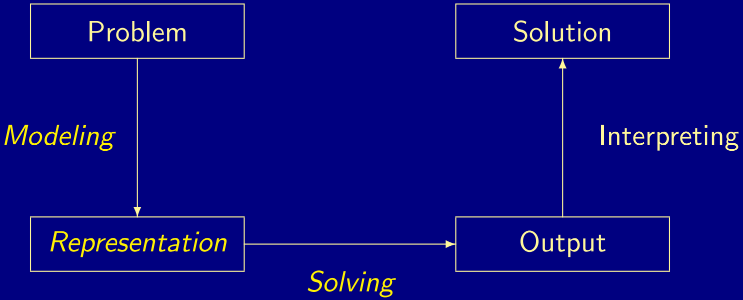
Declarative programming

"What is the problem?"

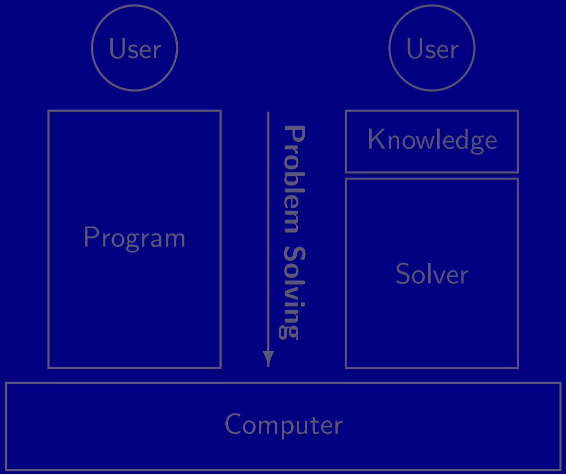


Declarative problem solving

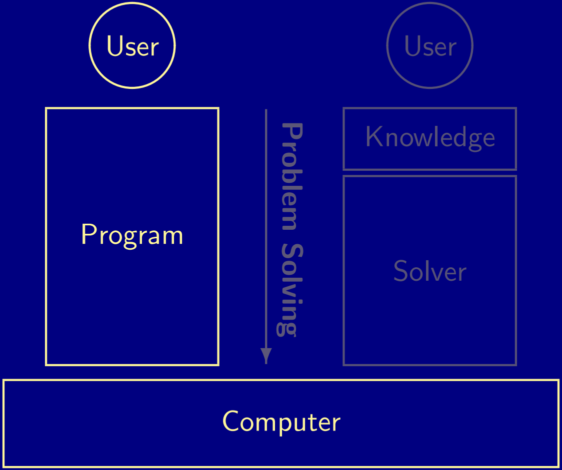
"What is the problem?"



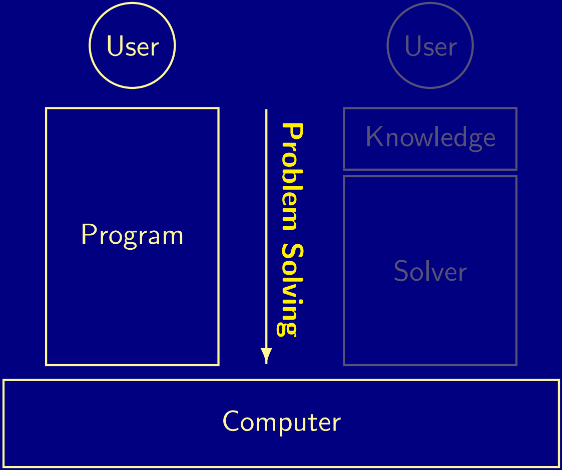
Traditional Software



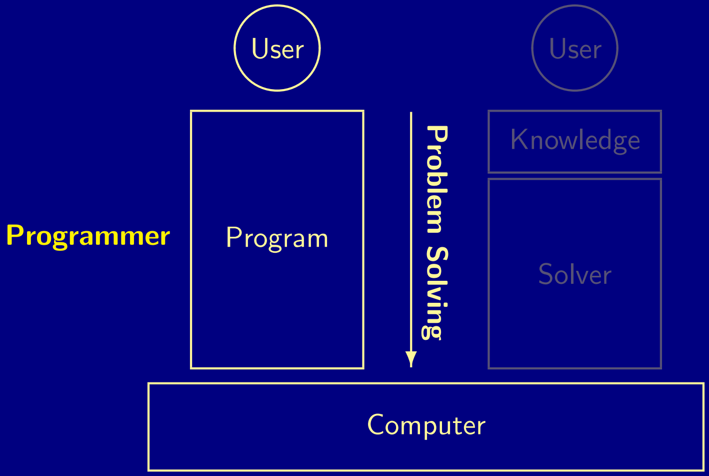
Traditional Software



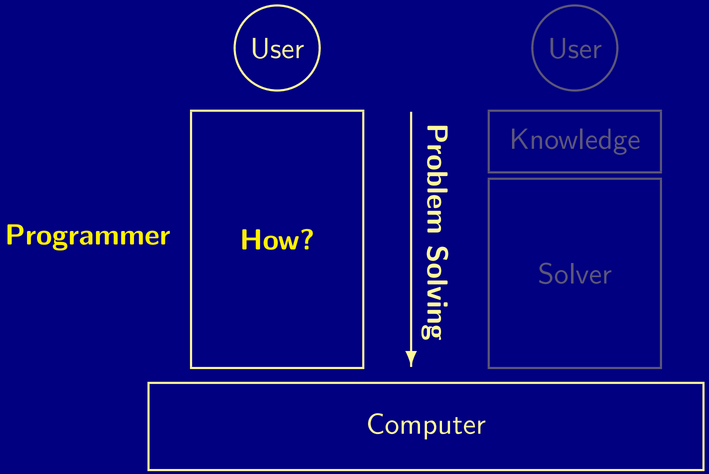
Traditional Software



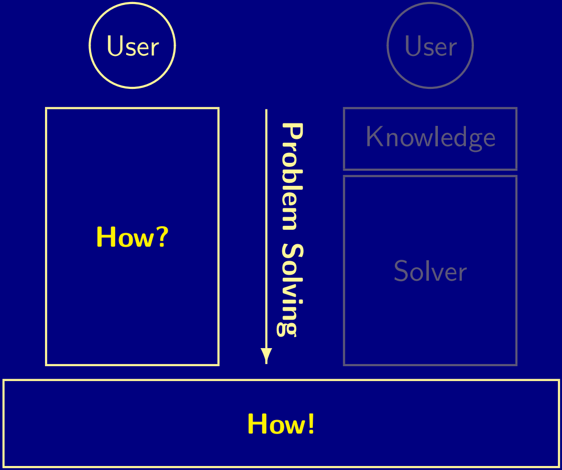
Traditional Software



Traditional Software

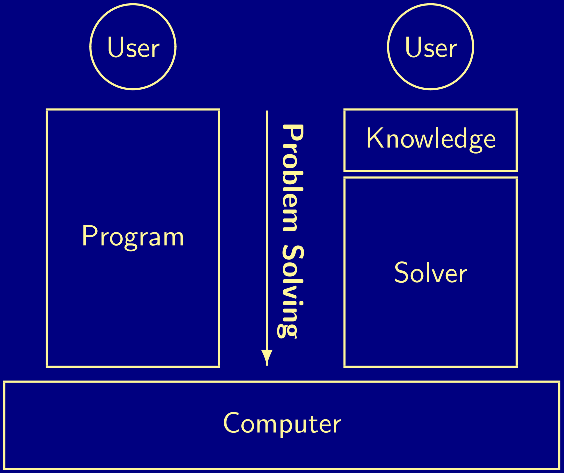


Traditional Software

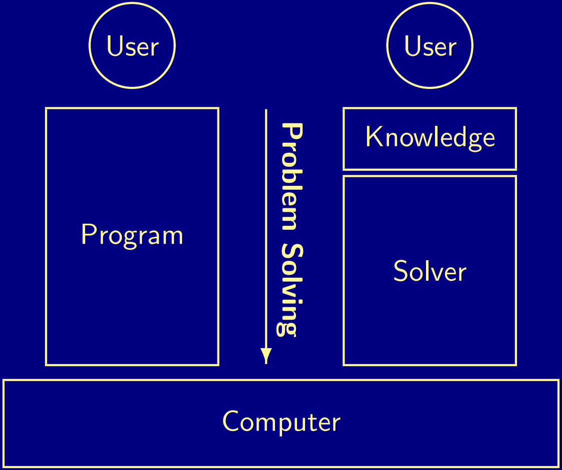


Potassco

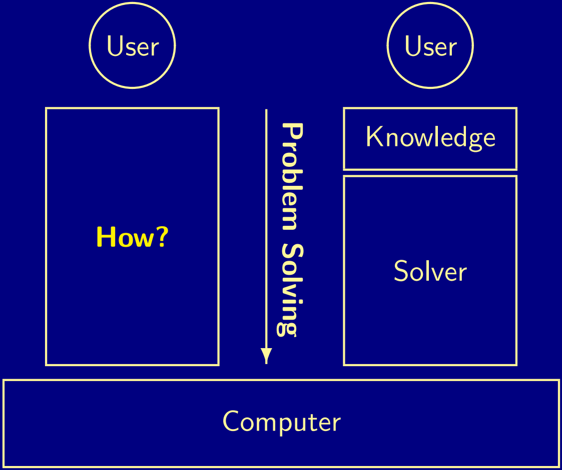
Declarative Software



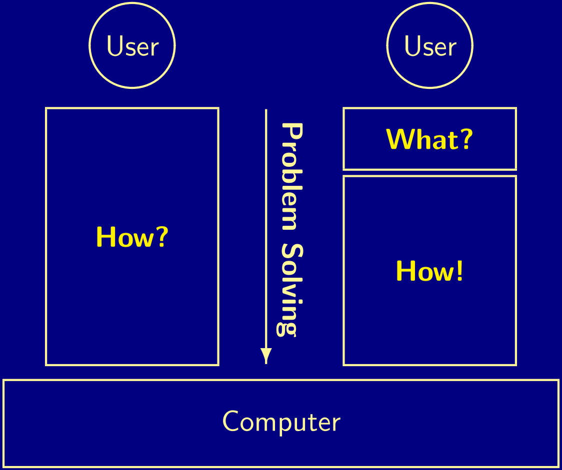
Knowledge-driven Software



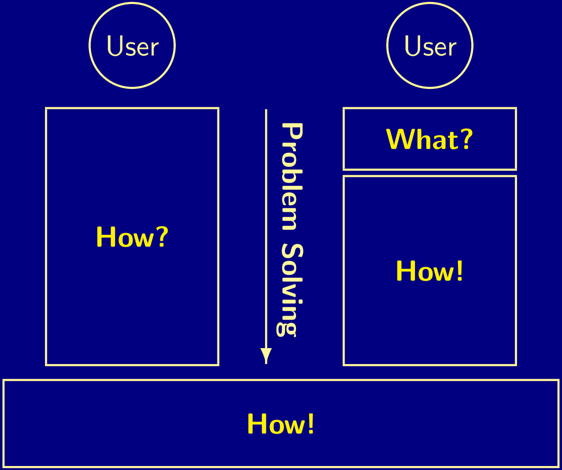
Knowledge-driven Software



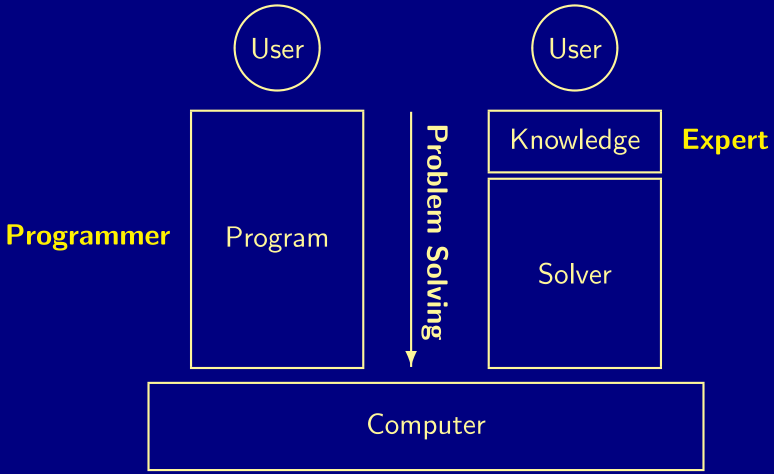
Knowledge-driven Software



Knowledge-driven Software



Knowledge-driven Software



What is the benefit?

+ Transparency
+ Flexibility
+ Maintainability
+ Reliability

+ Generality
+ Efficiency
+ Optimality
+ Availability

Knowledge **Expert**

Solver

Potassco

What is the benefit?

+ Transparency
+ Flexibility
+ Maintainability
+ Reliability

+ Generality
+ Efficiency
+ Optimality
+ Availability

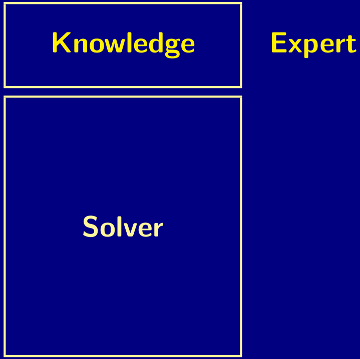
Knowledge	**Expert**
Solver	

Potassco

What is the benefit?

- \+ Transparency
- \+ Flexibility
- \+ Maintainability
- \+ Reliability

- \+ Generality
- \+ Efficiency
- \+ Optimality
- \+ Availability



What is the benefit?

- + Transparency
- + Flexibility
- + Maintainability
- + Reliability

- + Generality
- + Efficiency
- + Optimality
- + Availability

Natural Language	**Expert**
Solver	

Potassco

What is the benefit?

- + Transparency
- + Flexibility
- + Maintainability
- + Reliability

- + Generality
- + Efficiency
- + Optimality
- + Availability

Natural Language | **Layperson**

Solver

What is the benefit?

- \+ Transparency
- \+ Flexibility
- \+ Maintainability
- \+ Reliability

- \+ Generality
- \+ Efficiency
- \+ Optimality
- \+ Availability

Knowledge **Expert**

Solver

What is the benefit?

+ Transparency
+ Flexibility
+ Maintainability
+ Reliability

+ Generality
+ Efficiency
+ Optimality
+ Availability

Knowledge **Expert**

Solver

What is the benefit?

- \+ Transparency
- \+ Flexibility
- \+ Maintainability
- \+ Reliability

- \+ Generality
- \+ Efficiency
- \+ Optimality
- \+ Availability

Knowledge **Expert**

Solver

Outline

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- Where is ASP from?
  - Databases
  - Logic programming
  - Knowledge representation and reasoning
  - Satisfiability solving

Answer Set Programming (ASP)

- What is ASP? **ASP = DB+LP+KR+SAT !**
  ASP is an approach for declarative problem solving
- Where is ASP from?
  - Databases
  - Logic programming
  - Knowledge representation and reasoning
  - Satisfiability solving

Potassco

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this?
  Problems consisting of (many) decisions and constraints

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this?
  Problems consisting of (many) decisions and constraints

  Examples Sudoku, Configuration, Diagnosis, Music composition, Planning, System design, Time tabling, etc.

Potassco

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this? — **And industrial ones ?**
  Problems consisting of (many) decisions and constraints

  Examples Sudoku, Configuration, Diagnosis, Music composition, Planning, System design, Time tabling, etc.

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this? — **And industrial ones ?**
  - Debian, Ubuntu: Linux package configuration
  - Exeura: Call routing
  - Fcc: Radio frequency auction
  - Gioia Tauro: Workforce management
  - Nasa: Decision support for Space Shuttle
  - Siemens: Partner units configuration
  - Variantum: Product configuration
  - US Navy: risk assessment

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this? — **And industrial ones ?**
  - Debian, Ubuntu: Linux package configuration
  - Exeura: Call routing
  - **Fcc: Radio frequency auction**
  - Gioia Tauro: Workforce management
  - Nasa: Decision support for Space Shuttle
  - Siemens: Partner units configuration
  - Variantum: Product configuration
  - US Navy: risk assessment

Potassco

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative pr
- What is ASP good for?
  Solving knowledge-intense combinato
- What problems are this? — **And ind**
  - Debian, Ubuntu: Linux package cor
  - Exeura: Call routing
  - **Fcc: Radio frequency auction**
  - Gioia Tauro: Workforce management
  - Nasa: Decision support for Space Shuttle
  - Siemens: Partner units configuration
  - Variantum: Product configuration
  - US Navy: risk assessment

Over 13 months in 2016–17 the US Federal Communications Commission conducted an "incentive auction" to repurpose radio spectrum from broadcast television to wireless internet. In the end, the auction yielded $19.8 billion, $10.05 billion of which was paid to 175 broadcasters for voluntarily relinquishing their licenses across 14 UHF channels. Stations that continued broadcasting were assigned potentially new channels to fit as densely as possible into the channels that remained. The government netted more than $7 billion (used to pay down the national debt) after covering costs. A crucial element of the auction design was the construction of a solver, dubbed SATFC, that determined whether sets of stations could be "repacked" in this way; it needed to run every time a station was given a price quote. This

Potassco

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this?
  Problems consisting of (many) decisions and constraints
- What are ASP's distinguishing features?
  - High level, versatile modeling language
  - High performance solvers
  - Qualitative and quantitative optimization

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this?
  Problems consisting of (many) decisions and constraints
- What are ASP's distinguishing features?
  - High level, versatile modeling language
  - High performance solvers
  - Qualitative and quantitative optimization
- Any industrial impact?
  - ASP Tech companies: DLV Systems and **Potassco Solutions**
  - Increasing interest in (large) companies

Outline

Some (biased) moments in time

- '80 Capturing incomplete information
- '90 Amalgamation and computation
- '00 Applications and semantic rediscoveries
- '10 Customization and integration

Some (biased) moments in time

- '80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
- '00 Applications and semantic rediscoveries
- '10 Customization and integration

Some (biased) moments in time

- '80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
- '10 Customization and integration

Some (biased) moments in time

- '80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
  - Growing dissemination — see last slides —
  - Constructive logics Equilibrium Logic
- '10 Customization and integration

Potassco

Some (biased) moments in time

- '80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
  - Growing dissemination — see last slides —
  - Constructive logics Equilibrium Logic
- '10 Customization and integration
  - Complex reasoning modes APIs, multi-shot solving
  - Hybridization Constraint ASP, theory solving

Paradigm shift '90

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

Automated planning, Kautz and Selman (ECAI'92)

Represent planning problems as propositional theories so that models not proofs describe solutions

Paradigm shift '90

Theorem Proving based approach (eg. Prolog)

1. Provide a representation of the problem
2. A solution is given by a derivation of a query

Model Generation based approach (eg. SATisfiability testing)

1. Provide a representation of the problem
2. A solution is given by a model of the representation

Automated planning, Kautz and Selman (ECAI'92)

Represent planning problems as propositional theories so that models not proofs describe solutions

Potassco

Outline

Logic programs

- A logic program is a set of rules of the form

$$\underbrace{a}_{\text{head}} \leftarrow \underbrace{b_1, \dots, b_m, \neg c_1, \dots, \neg c_n}_{\text{body}}$$

  where
  - a and all b_i, c_j are atoms (propositional variables)
  - $\leftarrow$, $,$, $\neg$ denote if, and, and negation
  - intuitive reading: head must be true if body holds

- Semantics given by stable models, informally,
  1. (classical) models of the logic program
  2. requiring that each true atom is provable

Closed world assumption

Potassco

Logic programs

- A logic program is a set of rules of the form

$$\underbrace{a}_{\text{head}} \leftarrow \underbrace{b_1, \dots, b_m, \neg c_1, \dots, \neg c_n}_{\text{body}}$$

where

  - a and all b_i, c_j are atoms (propositional variables)
  - $\leftarrow$, $,$, $\neg$ denote if, and, and negation
  - intuitive reading: head must be true if body holds

- Semantics given by stable models, informally,
  1. (classical) models of the logic program
  2. requiring that each true atom is provable

Closed world assumption

Potassco

Logic programs

- A logic program is a set of rules of the form

$$\underbrace{a}_{\text{head}} \leftarrow \underbrace{b_1, \dots, b_m, \neg c_1, \dots, \neg c_n}_{\text{body}}$$

where
  - a and all b_i, c_j are atoms (propositional variables)
  - $\leftarrow$, $,$, $\neg$ denote if, and, and negation
  - intuitive reading: head must be true if body holds

- Semantics given by stable models, informally,
  1. (classical) models of the logic program
  2. requiring that each true atom is provable

Closed world assumption

Logic programs

- A logic program is a set of rules of the form

$$\underbrace{a}_{\text{head}} \leftarrow \underbrace{b_1, \dots, b_m, \neg c_1, \dots, \neg c_n}_{\text{body}}$$

  where
  - a and all b_i, c_j are atoms (propositional variables)
  - $\leftarrow$, $,$, $\neg$ denote if, and, and negation
  - intuitive reading: head must be true if body holds

- Semantics given by stable models, informally,
  1. (classical) models of the logic program
  2. requiring that each true atom is provable

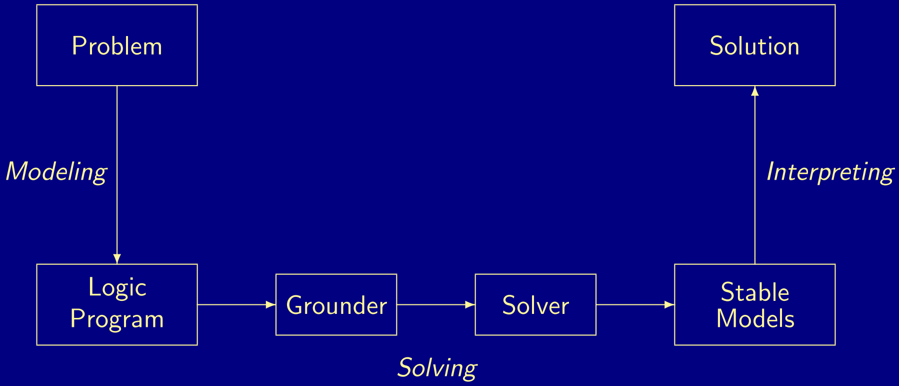
Closed world assumption

Potassco

Outline

Potassco

Modeling, grounding, and solving

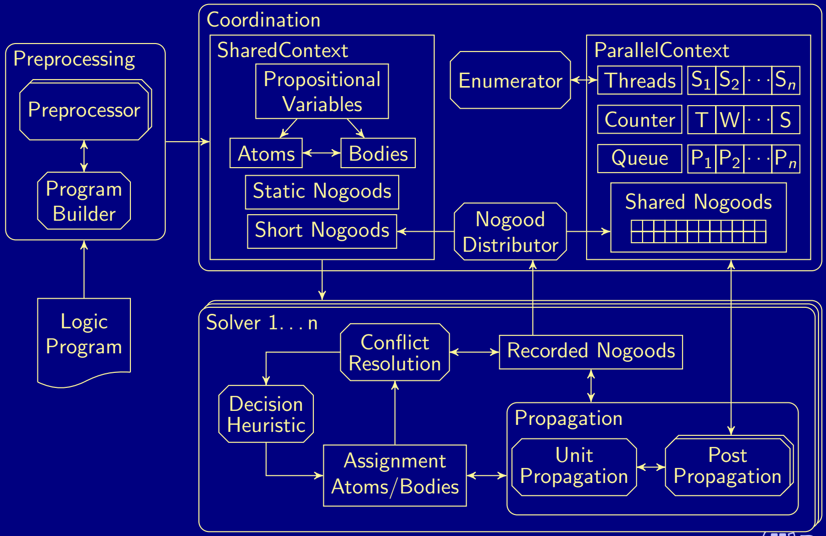


Potassco

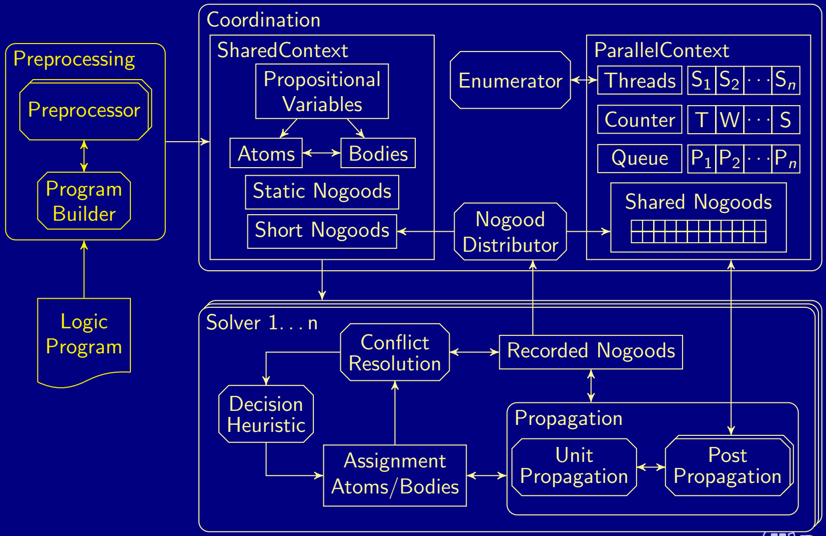
Outline

Potassco

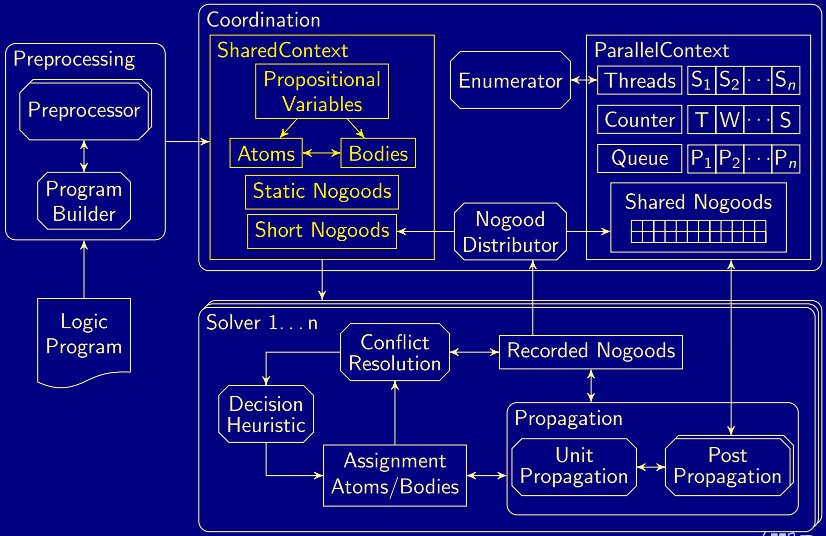
Multi-threaded architecture of *clasp*



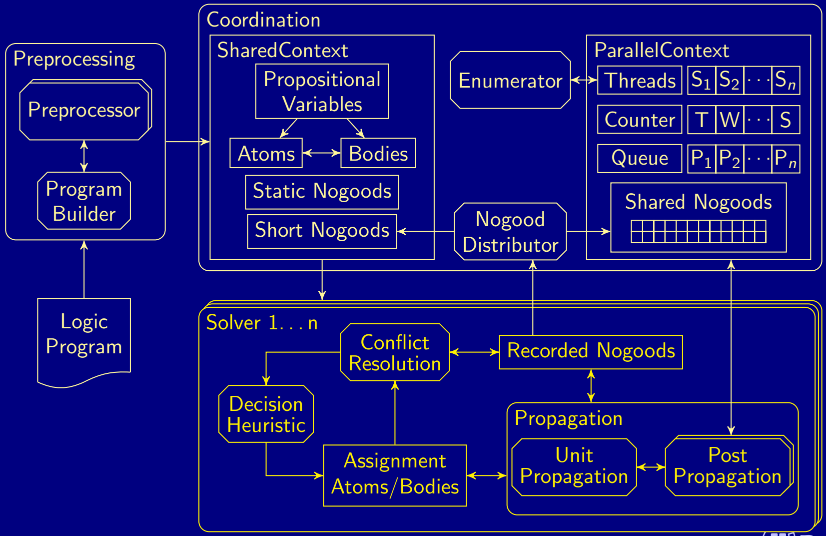
Multi-threaded architecture of *clasp*



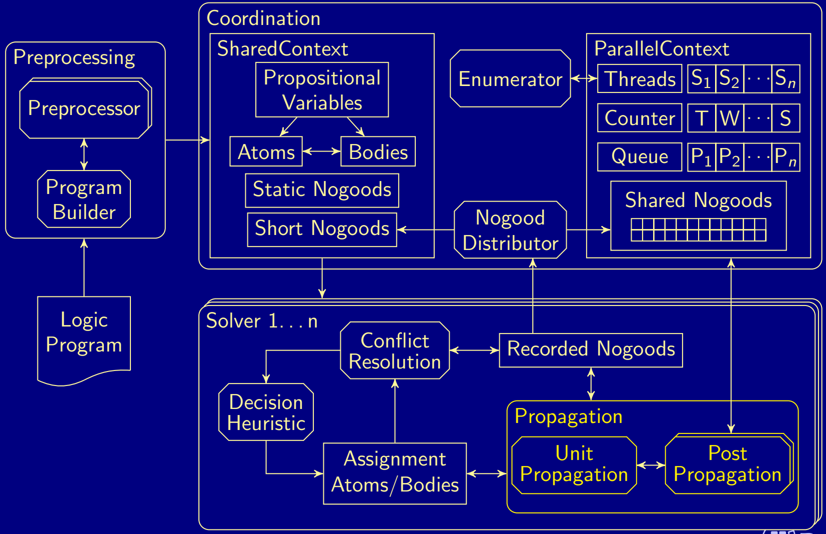
Multi-threaded architecture of *clasp*



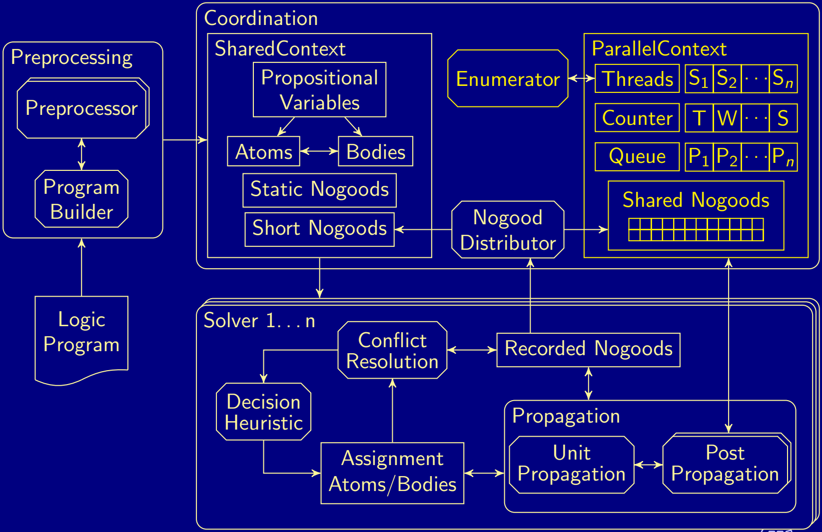
Multi-threaded architecture of *clasp*



Multi-threaded architecture of *clasp*



Multi-threaded architecture of *clasp*



Outline

Two sides of a coin

- ASP as High-level Language
  - Express problem instance as sets of facts
  - Encode problem class as a set of rules
  - Read off solutions from stable models of facts and rules
- ASP as Low-level Language
  - Compile a problem into a set of facts and rules
  - Solve the original problem by solving its compilation
- ASP and Imperative language
  - Control continuously changing logic programs

Two sides of a coin

- ASP as High-level Language
  - Express problem instance as sets of facts
  - Encode problem class as a set of rules
  - Read off solutions from stable models of facts and rules
- ASP as Low-level Language
  - Compile a problem into a set of facts and rules
  - Solve the original problem by solving its compilation
- ASP and Imperative language
  - Control continuously changing logic programs

Potassco

Two sides of a coin

- ASP as High-level Language
  - Express problem instance as sets of facts
  - Encode problem class as a set of rules
  - Read off solutions from stable models of facts and rules

- ASP as Low-level Language
  - Compile a problem into a set of facts and rules
  - Solve the original problem by solving its compilation

- ASP and Imperative language
  - Control continuously changing logic programs

Two sides of a coin

- ASP as High-level Language
  - Express problem instance as sets of facts
  - Encode problem class as a set of rules
  - Read off solutions from stable models of facts and rules
- ASP as "Low-level" Language
  - Compile a problem instance into a set of facts
  - Encode problem class as a set of rules
  - Solve the original problem by solving its compilation
- ASP and Imperative language
  - Control continuously changing logic programs

Two and a half sides of a coin

- ASP as High-level Language
  - Express problem instance as sets of facts
  - Encode problem class as a set of rules
  - Read off solutions from stable models of facts and rules
- ASP as "Low-level" Language
  - Compile a problem instance into a set of facts
  - Encode problem class as a set of rules
  - Solve the original problem by solving its compilation
- ASP and Imperative language
  - Control continuously changing logic programs

Outline

Upcoming experience

- ASP is a viable tool for Knowledge Representation and Reasoning
  - Integration of DB, LP, KR, and SAT techniques
  - Combinatorial search problems in the realm of NP and NP^{NP}
  - Succinct, elaboration-tolerant problem representations
    - rapid application development tool
  - Easy handling of knowledge-intensive applications
    - data, defaults, exceptions, frame axioms, reachability etc
- ASP offers efficient and versatile off-the-shelf solving technology
  - `https://potassco.org`
  - winning ASP, CASC, MISC, PB, and SAT competitions
- ASP has a growing range of applications, and its's good fun!

Upcoming experience

- ASP is a viable tool for Knowledge Representation and Reasoning
  - Integration of DB, LP, KR, and SAT techniques
  - Combinatorial search problems in the realm of NP and NP^{NP}
  - Succinct, elaboration-tolerant problem representations
    - rapid application development tool
  - Easy handling of knowledge-intensive applications
    - data, defaults, exceptions, frame axioms, reachability etc
- ASP offers efficient and versatile off-the-shelf solving technology
  - `https://potassco.org`
  - winning ASP, CASC, MISC, PB, and SAT competitions
- ASP has a growing range of applications, and its's good fun!

$$\mathrm{ASP} = \mathrm{DB}+\mathrm{LP}+\mathrm{KR}+\mathrm{SAT}$$

Upcoming experience

- ASP is a viable tool for Knowledge Representation and Reasoning
  - Integration of DB, LP, KR, and SAT techniques
  - Combinatorial search problems in the realm of NP and NP^{NP}
  - Succinct, elaboration-tolerant problem representations
    - rapid application development tool
  - Easy handling of knowledge-intensive applications
    - data, defaults, exceptions, frame axioms, reachability etc
- ASP offers efficient and versatile off-the-shelf solving technology
  - `https://potassco.org`
  - winning ASP, CASC, MISC, PB, and SAT competitions
- ASP has a growing range of applications, and its's good fun!

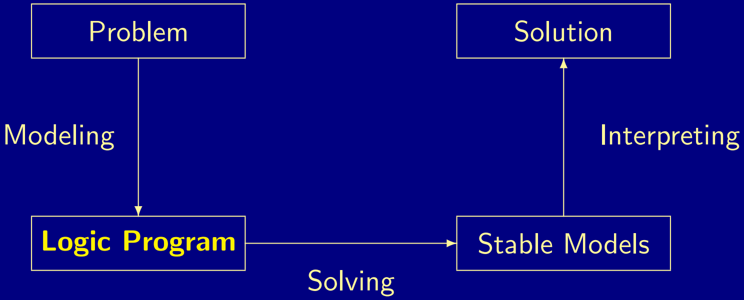
$$\text{ASP} = \text{DB}+\text{LP}+\text{KR}+\text{SMT}^n$$

Potassco

Introduction: Overview

Outline

Syntax


Problem
Solution
Modeling
Interpreting
Logic Program
Stable Models
Solving

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

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

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
  - $\mathcal{A}$ is also called the alphabet (or signature) of P
- A (normal) rule, r, is of the form

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

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
  - $\mathcal{A}$ is also called the alphabet (or signature) of P
- A (normal) rule, r, is of the form

$$\underbrace{a_0}_{\text{head}} \leftarrow \underbrace{a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n}_{\text{body}}$$

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

$$a_0 \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

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

$$a_0 \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

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

Potassco

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

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

  where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$
- A literal is an atom or a negated atom
- Note A body is a (finite) set of literals

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

$$\mathtt{a_0\ :\text{-}\ a_1,\ldots,a_m,\ not\ a_{m+1},\ldots,\ not\ a_n.}$$

  where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$
- A literal is an atom or a negated atom
- Note A body is a (finite) set of literals

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

$$a_0 \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

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

$$a_0 \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

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

$$\begin{array}{rcl} h(r) & = & a_0 \\ B(r) & = & \{a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n\} \end{array}$$

Potassco

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

$$\underbrace{a_0}_{h(r)} \leftarrow \underbrace{a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n}_{B(r)}$$

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

$$\begin{aligned} h(r) &= a_0 \\ B(r) &= \{a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n\} \end{aligned}$$

Potassco

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

$$a_0 \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

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

$$\begin{aligned} h(r) &= a_0 \\ B(r) &= \{a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n\} \end{aligned}$$

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

$$a_0 \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

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

$$\begin{aligned} H(r) &= \{a_0\} \\ B(r) &= \{a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n\} \end{aligned}$$

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

$$a_0 \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

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

$$\begin{aligned} h(r) &= a_0 \\ B(r) &= \{a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n\} \\ H(P) &= \{h(r) \mid r \in P\} \\ B(P) &= \{B(r) \mid r \in P\} \end{aligned}$$

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

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

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

$$\begin{aligned} h(r) &= a_0 \\ B(r)^+ &= \{a_1, \ldots, a_m\} \\ B(r)^- &= \{a_{m+1}, \ldots, a_n\} \end{aligned}$$

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

$$a_0 \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

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

$$\begin{aligned}
H(r) &= \{a_0\} \\
B(r)^+ &= \{a_1, \dots, a_m\} \\
B(r)^- &= \{a_{m+1}, \dots, a_n\} \\
A(P) &= \bigcup_{r \in P} \left(H(r) \cup B(r)^+ \cup B(r)^-\right)
\end{aligned}$$

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

$$a_0 \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

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

$$\begin{aligned} H(r) &= \{a_0\} \\ B(r)^+ &= \{a_1, \dots, a_m\} \\ B(r)^- &= \{a_{m+1}, \dots, a_n\} \\ A(P) &= \bigcup_{r \in P} \left(H(r) \cup B(r)^+ \cup B(r)^-\right) \end{aligned}$$

- Note We often assume that $\mathcal{A} = A(P)$

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

$$a_0 \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

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

$$\begin{aligned} h(r) &= a_0 \\ B(r)^+ &= \{a_1, \dots, a_m\} \\ B(r)^- &= \{a_{m+1}, \dots, a_n\} \end{aligned}$$

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

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

$$a_0 \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

  where $0 \leq m \leq n$ and each $a_i \in \mathcal{A}$ is an atom for $0 \leq i \leq n$
- A program P is positive if $B(r)^- = \emptyset$ for all $r \in P$

Examples

- Example rules
  - $a \leftarrow b, \neg c$
  - $a \leftarrow \neg c, b$
  - $a \leftarrow$
  - $a \leftarrow b$
  - $a \leftarrow \neg c$
  - $bachelor(joe) \leftarrow male(joe), \neg married(joe)$
- Example literals

  $a, b, c, bachelor(joe), male(joe), married(joe)$

  $\neg c, \neg married(joe)$

Examples

- Example rules
  - $a \leftarrow b, \neg c$
  - $a \leftarrow \neg c, b$
  - $a \leftarrow$
  - $a \leftarrow b$
  - $a \leftarrow \neg c$
  - $bachelor(joe) \leftarrow male(joe), \neg married(joe)$
- Example literals
  - $a, b, c, bachelor(joe), male(joe), married(joe)$
  - $\neg c, \neg married(joe)$

Potassco

Examples

- Example rules
  - $a \leftarrow b, \neg c$
  - $a \leftarrow \neg c, b$
  - $a \leftarrow$
  - $a \leftarrow b$
  - $a \leftarrow \neg c$
  - $bachelor(joe) \leftarrow male(joe), \neg married(joe)$
- Example literals
  - $a, b, c, bachelor(joe), male(joe), married(joe)$
  - $\neg c, \neg married(joe)$

Examples

- Example rules
  - $a \leftarrow b, \neg c$
  - $a \leftarrow \neg c, b$
  - $a \leftarrow$
  - $a \leftarrow b$
  - $a \leftarrow \neg c$
  - $bachelor(joe) \leftarrow male(joe), \neg married(joe)$
- Example literals
  - $a, b, c, bachelor(joe), male(joe), married(joe)$
  - $\neg c, \neg married(joe)$

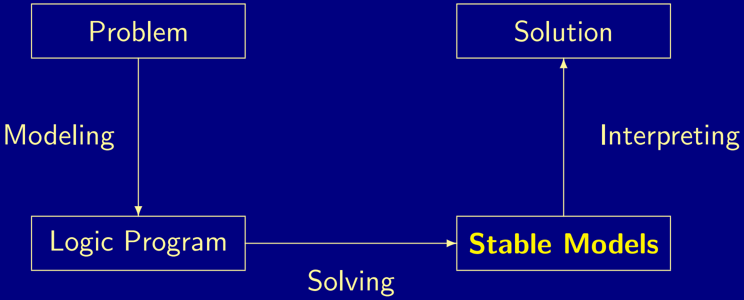
Notational conventions

	false, true	if	and	or	iff	(default) negation	strong negation
source code		`:-`	`,`	`;`		`not`	`-`
logic program		$\leftarrow$	$,$	$;$		$\neg$	$\sim$
formula	$\bot, \top$	$\rightarrow$	$\wedge$	$\vee$	$\leftrightarrow$	$\neg$	$\sim$

Potassco

Outline

Semantics


Problem
Solution
Modeling
Interpreting
Logic Program
Stable Models
Solving

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

What is a model?

- Assignment A function mapping variables to values
  - Example $A : \{x, y, z\} \rightarrow \mathbb{N}$ such that $A = \{x \mapsto 3, y \mapsto 1, z \mapsto 7\}$
- Solution An assignment satisfying a set of constraints

Potassco

What is a model?

- Assignment A function mapping variables to values
  - Example $A : \{x, y, z\} \to \mathbb{N}$ such that $A = \{x \mapsto 3, y \mapsto 1, z \mapsto 7\}$
- Solution An assignment satisfying a set of constraints
  - Example A is a solution of $\{2x < z, x + y < 2z\}$

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

- Interpretation An assignment mapping variables to truth values

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

- Interpretation An assignment mapping variables to truth values
  - Truth values $\boldsymbol{T}$ and $\boldsymbol{F}$ stand for true and false (but there may be others)

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

- Interpretation An assignment mapping variables to truth values
  - Truth values $\boldsymbol{T}$ and $\boldsymbol{F}$ stand for true and false (but there may be others)
  - Example $B : \{a, b, c\} \rightarrow \{\boldsymbol{T}, \boldsymbol{F}\}$ such that $B = \{a \mapsto \boldsymbol{T}, b \mapsto \boldsymbol{T}, c \mapsto \boldsymbol{F}\}$

Potassco

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

- Interpretation An assignment mapping variables to truth values
  - Truth values $\boldsymbol{T}$ and $\boldsymbol{F}$ stand for true and false (but there may be others)
  - Example $B : \{a, b, c\} \to \{\boldsymbol{T}, \boldsymbol{F}\}$ such that $B = \{a \mapsto \boldsymbol{T}, b \mapsto \boldsymbol{T}, c \mapsto \boldsymbol{F}\}$
  - An interpretation satisfies a formula if it evaluates the formula to $\boldsymbol{T}$

Potassco

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

- Interpretation An assignment mapping variables to truth values
  - Truth values $\boldsymbol{T}$ and $\boldsymbol{F}$ stand for true and false (but there may be others)
  - Example $B : \{a, b, c\} \rightarrow \{\boldsymbol{T}, \boldsymbol{F}\}$ such that $B = \{a \mapsto \boldsymbol{T}, b \mapsto \boldsymbol{T}, c \mapsto \boldsymbol{F}\}$
  - An interpretation satisfies a formula if it evaluates the formula to $\boldsymbol{T}$
- Model An interpretation satisfying a set of formulas (or rules)

Potassco

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

- Interpretation An assignment mapping variables to truth values
  - Truth values $\boldsymbol{T}$ and $\boldsymbol{F}$ stand for true and false (but there may be others)
  - Example $B : \{a, b, c\} \to \{\boldsymbol{T}, \boldsymbol{F}\}$ such that $B = \{a \mapsto \boldsymbol{T}, b \mapsto \boldsymbol{T}, c \mapsto \boldsymbol{F}\}$
  - An interpretation satisfies a formula if it evaluates the formula to $\boldsymbol{T}$
- Model An interpretation satisfying a set of formulas (or rules)
  - Example B is a model of $\{a \wedge b, a \vee c\}$

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

- Interpretation An assignment mapping variables to truth values
- Model An interpretation satisfying a set of formulas (or rules)

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

- Interpretation An assignment mapping variables to truth values
- Model An interpretation satisfying a set of formulas (or rules)

- Representation
  We often denote interpretations by the set of their true atoms

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

- Interpretation An assignment mapping variables to truth values
- Model An interpretation satisfying a set of formulas (or rules)

- Representation
  We often denote interpretations by the set of their true atoms
  - Example We use $\{a, b\}$ to represent $\{a \mapsto \boldsymbol{T}, b \mapsto \boldsymbol{T}, c \mapsto \boldsymbol{F}\}$

What is a model?

- Assignment A function mapping variables to values
- Solution An assignment satisfying a set of constraints

- Interpretation An assignment mapping variables to truth values
- Model An interpretation satisfying a set of formulas (or rules)

- Representation
  We often denote interpretations by the set of their true atoms
  and (alternatively) use sets as the semantic cornerstones
  (rather than assignments or interpretations)

What is a stable model?

- Reduct-based characterization
- Logical characterization
- Axiomatic characterization
- Operational characterization
- Proof-theoretic characterization
- Constraint-based characterization
- Algorithmic characterization
- C++-based characterization

Vladimir Lifschitz, Thirteen Definitions of a Stable Model, [42, 41]

What is a stable model?

- Reduct-based characterization
- Logical characterization
- Axiomatic characterization
- Operational characterization
- Proof-theoretic characterization
- Constraint-based characterization
- Algorithmic characterization
- C++-based characterization

- Vladimir Lifschitz, Thirteen Definitions of a Stable Model, [42, 41]

What is a stable model?

- Reduct-based characterization
- Logical characterization
- Axiomatic characterization
- Operational characterization
- Proof-theoretic characterization
- Constraint-based characterization
- Algorithmic characterization
- C++-based characterization

- Vladimir Lifschitz, Thirteen Definitions of a Stable Model, [42, 41]

What is a stable model?

- Reduct-based characterization
- Logical characterization
- Axiomatic characterization
- Operational characterization
- Proof-theoretic characterization
- Constraint-based characterization
- Algorithmic characterization
- C++-based characterization

- Vladimir Lifschitz, Thirteen Definitions of a Stable Model, [42, 41]

What is the meaning of a logic program?

- Idea The set of atoms derivable from the rules in program
- Question How to characterize X given P?

Potassco

What is the meaning of a set of rules?

- Idea The set of atoms derivable from the rules in program
- Question How to characterize X given P?

Potasso

What is the meaning of a set of positive rules?

- Idea The set of atoms derivable from the rules in program
- Question How to characterize X given P?

What is the meaning of a set of positive rules?

- Idea The set of atoms derivable from the rules in program
- Question How to characterize X given P?

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

$$
\begin{array}{l}
X := \emptyset \\
\textbf{while } \{h(r) \mid r \in P, B(r)^{+} \subseteq X\} \neq X \\
\qquad X := \{h(r) \mid r \in P, B(r)^{+} \subseteq X\} \\
\textbf{return } X
\end{array}
$$

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow, b \leftarrow a\}$

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

$$
\begin{array}{l}
\textbf{let } T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\} \textbf{ in} \\
\quad X := \emptyset \\
\quad \textbf{while } T_P X \neq X \\
\quad\quad X := T_P X \\
\quad \textbf{return } X
\end{array}
$$

- Example $P = \{a \leftarrow, b \leftarrow a\}$

  $X_0 = \emptyset$

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow , b \leftarrow a\}$

  $X_0 = \emptyset$

  $T_P X_0 = \{a\}$

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**
  $X := \emptyset$
  while $T_P X \neq X$
  $X := T_P X$
  return X

- Example $P = \{a \leftarrow, b \leftarrow a\}$

$$X_0 = \emptyset \qquad X_1 = \{a\}$$
$$T_P X_0 = \{a\}$$

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

$$
\begin{array}{l}
\textbf{let } T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\} \textbf{ in} \\
\quad X := \emptyset \\
\quad \textbf{while } T_P X \neq X \\
\quad\quad X := T_P X \\
\quad \textbf{return } X
\end{array}
$$

- Example $P = \{a \leftarrow, b \leftarrow a\}$

$$
\begin{array}{ll}
X_0 = \emptyset & X_1 = \{a\} \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\}
\end{array}
$$

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**
  $X := \emptyset$
  while $T_P X \neq X$
  $X := T_P X$
  return X

- Example $P = \{a \leftarrow\ , b \leftarrow a\}$

$$X_0 = \emptyset \qquad X_1 = \{a\} \qquad X_2 = \{a, b\}$$
$$T_P X_0 = \{a\} \qquad T_P X_1 = \{a, b\}$$

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow, b \leftarrow a\}$

$$
\begin{array}{ccc}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b\}
\end{array}
$$

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**
  $X := \emptyset$
  while $T_P X \neq X$
  $X := T_P X$
  return X

- Example $P = \{a \leftarrow \ , b \leftarrow a\}$

$$X_0 = \emptyset \qquad X_1 = \{a\} \qquad X_2 = \{a, b\} \qquad X = \{a, b\}$$
$$T_P X_0 = \{a\} \qquad T_P X_1 = \{a, b\} \qquad T_P X_2 = \{a, b\}$$

What is the meaning of a set of positive rules?

- Idea: The set X of atoms derivable from the rules in program P
- Question: How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow , b \leftarrow a\}$

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

$$
\begin{array}{l}
\textbf{let } T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\} \textbf{ in} \\
\quad X := \emptyset \\
\quad \textbf{while } T_P X \neq X \\
\quad\quad X := T_P X \\
\quad \textbf{return } X
\end{array}
$$

- Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$

  $X_0 = \emptyset$

What is the meaning of a set of positive rules?

- **Idea** The set X of atoms derivable from the rules in program P
- **Question** How to characterize X given P?

- **Procedural characterization**

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- **Example** $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$

  $X_0 = \emptyset$

  $T_P X_0 = \{a\}$

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$

  $X_0 = \emptyset$ $X_1 = \{a\}$

  $T_P X_0 = \{a\}$

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$

$X_0 = \emptyset$ $\quad X_1 = \{a\}$

$T_P X_0 = \{a\}$ $\quad T_P X_1 = \{a, b\}$

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

$$
\begin{array}{l}
\textbf{let } T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\} \textbf{ in} \\
\qquad X := \emptyset \\
\qquad \textbf{while } T_P X \neq X \\
\qquad\qquad X := T_P X \\
\qquad \textbf{return } X
\end{array}
$$

- Example $P = \{a \leftarrow \; , b \leftarrow a, d \leftarrow c\}$

$$
\begin{array}{lll}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} &
\end{array}
$$

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

$$
\begin{array}{l}
\textbf{let } T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\} \textbf{ in} \\
\quad X := \emptyset \\
\quad \textbf{while } T_P X \neq X \\
\quad\quad X := T_P X \\
\quad \textbf{return } X
\end{array}
$$

- Example $P = \{a \leftarrow\ , b \leftarrow a, d \leftarrow c\}$

$$
\begin{array}{ccc}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b\}
\end{array}
$$

Potassco

What is the meaning of a set of positive rules?

- Idea: The set X of atoms derivable from the rules in program P
- Question: How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$

$$X_0 = \emptyset \quad X_1 = \{a\} \quad X_2 = \{a, b\} \quad X = \{a, b\}$$
$$T_P X_0 = \{a\} \quad T_P X_1 = \{a, b\} \quad T_P X_2 = \{a, b\}$$

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural characterization

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Procedural answer The "meaning of P" is given by the value X returned by the procedure applied to P

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a\}$

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a\}$
    Is $\emptyset$ closed under P?

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a\}$
    Is $\emptyset$ closed under P? ✘

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a\}$
    Is $\{a\}$ closed under P ?

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a\}$
    Is $\{a\}$ closed under P ? ✘

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a\}$
    Is $\{b\}$ closed under P?

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a\}$
    Is $\{b\}$ closed under P ? ✘

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a\}$
    Is $\{a, b\}$ closed under P ?

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a\}$
    Is $\{a, b\}$ closed under P ? ✔

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$

    Is $\emptyset$ closed under P ?

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^{+} \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$

    Is $\emptyset$ closed under P? ✘

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$
    Is $\{a\}$ closed under P?

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$
    Is $\{a\}$ closed under P ? ✘

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$
    Is $\{b\}$ closed under P ?

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$
    Is $\{b\}$ closed under P ? ✘

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$
    Is $\{a, b\}$ closed under P ?

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$
    Is $\{a, b\}$ closed under P? ✔

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$

    Is $\{a, b, c\}$ closed under P ?

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$
    Is $\{a, b, c\}$ closed under P ? ✘

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$
    Is $\{a, b, d\}$ closed under P ?

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$
    Is $\{a, b, d\}$ closed under P? ✔

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$
    Is $\{a, b, c, d\}$ closed under P ?

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow\ , b \leftarrow a, d \leftarrow c\}$
    Is $\{a, b, c, d\}$ closed under P ? ✔

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$
    Is $\{a, b, c, d, e\}$ closed under P ?

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$
    Is $\{a, b, c, d, e\}$ closed under P ? ✔

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Mathematical answer The "meaning of P" is given by the $\subseteq$-smallest set of atoms closed under P

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^+ \subseteq X$ for all $r \in P$
  - Mathematical answer The "meaning of P" is given by the $\subseteq$-smallest set of atoms closed under P
  - and denoted by $Cn(P)$ and called the consequences of P

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Mathematical characterization
  - A set X of atoms is closed under a positive program P if $h(r) \in X$ whenever $B(r)^{+} \subseteq X$ for all $r \in P$
  - Mathematical answer The "meaning of P" is given by the $\subseteq$-smallest set of atoms closed under P
  - and denoted by $Cn(P)$ and called the consequences of P

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural answer Value X returned by the procedure applied to P
- Mathematical answer Consequences $Cn(P)$ of P

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural answer Value X returned by the procedure applied to P
- Mathematical answer Consequences $Cn(P)$ of P

- Common answer $X = Cn(P)$

Potassco

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural answer Value X returned by the procedure applied to P
- Mathematical answer Consequences $Cn(P)$ of P

- Common answer $X = Cn(P)$

- Logical answer
  - A set of atoms closed under P is a model of P and vice versa
  - $Cn(P)$ corresponds to the smallest model of P

What is the meaning of a set of positive rules?

- Idea The set X of atoms derivable from the rules in program P
- Question How to characterize X given P?

- Procedural answer Value X returned by the procedure applied to P
- Mathematical answer Consequences $Cn(P)$ of P

- Common answer $X = Cn(P)$

- Logical answer
  - A set of atoms closed under P is a model of P and vice versa
  - $Cn(P)$ corresponds to the smallest model of P

Potassco

Some "logical" remarks

- Positive rules are also referred to as definite clauses
  - Definite clauses are disjunctions with exactly one positive atom:

$$a_0 \vee \neg a_1 \vee \cdots \vee \neg a_m$$

  - A set of definite clauses has a (unique) smallest model
- Horn clauses are clauses with at most one positive atom
  - Every definite clause is a Horn clause but not vice versa
  - Non-definite Horn clauses can be regarded as integrity constraints
  - A set of Horn clauses has a smallest model or none
- This smallest model is the intended semantics of such sets of clauses
  - Given a positive program P, $Cn(P)$ corresponds to the smallest model of the set of definite clauses corresponding to P

Potassco

Some "logical" remarks

- Positive rules are also referred to as definite clauses
  - Definite clauses are disjunctions with exactly one positive atom:

$$a_0 \vee \neg a_1 \vee \cdots \vee \neg a_m$$

  - A set of definite clauses has a (unique) smallest model

- Horn clauses are clauses with at most one positive atom
  - Every definite clause is a Horn clause but not vice versa
  - Non-definite Horn clauses can be regarded as integrity constraints
  - A set of Horn clauses has a smallest model or none

- This smallest model is the intended semantics of such sets of clauses
  - Given a positive program P, $Cn(P)$ corresponds to the smallest model of the set of definite clauses corresponding to P

Potassco

Some "logical" remarks

- Positive rules are also referred to as definite clauses
  - Definite clauses are disjunctions with exactly one positive atom:

$$a_0 \vee \neg a_1 \vee \cdots \vee \neg a_m$$

  - A set of definite clauses has a (unique) smallest model
- Horn clauses are clauses with at most one positive atom
  - Every definite clause is a Horn clause but not vice versa
  - Non-definite Horn clauses can be regarded as integrity constraints
  - A set of Horn clauses has a smallest model or none
- This smallest model is the intended semantics of such sets of clauses
  - Given a positive program P, $Cn(P)$ corresponds to the smallest model of the set of definite clauses corresponding to P

Potassco

Some "logical" remarks

- Positive rules are also referred to as definite clauses
  - Definite clauses are disjunctions with exactly one positive atom:

$$a_0 \vee \neg a_1 \vee \cdots \vee \neg a_m$$

  - A set of definite clauses has a (unique) smallest model
- Horn clauses are clauses with at most one positive atom
  - Every definite clause is a Horn clause but not vice versa
  - Non-definite Horn clauses can be regarded as integrity constraints
  - A set of Horn clauses has a smallest model or none
- This smallest model is the intended semantics of such sets of clauses
  - Given a positive program P, $Cn(P)$ corresponds to the smallest model of the set of definite clauses corresponding to P

Potassco

What is the meaning of a logic program?

What is the meaning of a logic program?

- Lessons learned from positive programs
  - $Cn(P)$ is the smallest set closed under P, eliminating all others
  - Every atom in $Cn(P)$ is justified by a proof

What is the meaning of a logic program?

- Lessons learned from positive programs
  - $Cn(P)$ is the smallest set closed under P, eliminating all others
  - Every atom in $Cn(P)$ is justified by a proof
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$ yields $\{a, b\}$ only
    - a is justified by $a \leftarrow$
    - b is justified by $b \leftarrow a$ and $a \leftarrow$

Potassco

What is the meaning of a logic program?

- Lessons learned from positive programs
  - $Cn(P)$ is the smallest set closed under P, eliminating all others
  - Every atom in $Cn(P)$ is justified by a proof
  - Example $P = \{a \leftarrow, b \leftarrow a, d \leftarrow c\}$ yields $\{a, b\}$ only
    - a is justified by $a \leftarrow$
    - b is justified by $b \leftarrow a$ and $a \leftarrow$

What is the meaning of a logic program?

- Lessons learned from positive programs
  - $Cn(P)$ is the smallest set closed under P, eliminating all others
  - Every atom in $Cn(P)$ is justified by a proof

What is the meaning of a logic program?

- Lessons learned from positive programs
  - $Cn(P)$ is the smallest model of P, eliminating all others
  - Every atom in $Cn(P)$ is justified by a proof
- Logical attempt
  - Hypothesis The smallest model of P ?

What is the meaning of a logic program?

- Lessons learned from positive programs
  - $Cn(P)$ is the smallest model of P, eliminating all others
  - Every atom in $Cn(P)$ is justified by a proof

- Logical attempt
  - Hypothesis The smallest model of P ?

What is the meaning of a logic program?

- Lessons learned from positive programs
  - $Cn(P)$ is the smallest model of P, eliminating all others
  - Every atom in $Cn(P)$ is justified by a proof
- Logical attempt
  - Hypothesis The smallest model of P ?
  - Example $P = \{a \leftarrow \neg b\}$ yields (stable model) $\{a\}$

What is the meaning of a logic program?

- Lessons learned from positive programs
  - $Cn(P)$ is the smallest model of P, eliminating all others
  - Every atom in $Cn(P)$ is justified by a proof

- Logical attempt
  - Hypothesis The smallest model of P ?
  - Example $P = \{a \leftarrow \neg b\}$ yields (stable model) $\{a\}$
    but P has three models, $\{a\}$, $\{b\}$, and $\{a, b\}$

Potassco

What is the meaning of a logic program?

- Lessons learned from positive programs
  - $Cn(P)$ is the smallest model of P, eliminating all others
  - Every atom in $Cn(P)$ is justified by a proof
- Logical attempt
  - Hypothesis The smallest model of P ?
  - Example $P = \{a \leftarrow \neg b\}$ yields (stable model) $\{a\}$
    but P has two minimal models, $\{a\}$ and $\{b\}$

Potassco

What is the meaning of a logic program?

- Lessons learned from positive programs
  - $Cn(P)$ is the smallest model of P, eliminating all others
  - Every atom in $Cn(P)$ is justified by a proof
- Logical attempt — failed
  - Hypothesis The smallest model of P ?
  - Example $P = \{a \leftarrow \neg b\}$ yields (stable model) $\{a\}$
    but P has two minimal models, $\{a\}$ and $\{b\}$

What is the meaning of a logic program?

- Lessons learned from positive programs
  - $Cn(P)$ is the smallest set closed under P, eliminating all others
  - Every atom in $Cn(P)$ is justified by a proof
- Procedural attempt

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow \ , \ b \leftarrow a, \neg c\}$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow ,\ b \leftarrow a, \neg c\}$

  $X_0 = \emptyset$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$
  while $T_P X \neq X$
  $\quad X := T_P X$
  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$

$$X_0 = \emptyset$$
$$T_P X_0 = \{a\}$$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$

$$X_0 = \emptyset \qquad X_1 = \{a\}$$
$$T_P X_0 = \{a\}$$

Potassco

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$

$$\begin{array}{rr} X_0 = \emptyset & X_1 = \{a\} \\ T_P X_0 = \{a\} & T_P X_1 = \{a, b\} \end{array}$$

Potassco

What is the meaning of a logic program?

- Procedural attempt

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

$X := \emptyset$

while $T_P X \neq X$

$X := T_P X$

return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$

$$
\begin{array}{ccc}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} &
\end{array}
$$

Potassco

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$
  while $T_P X \neq X$
  $X := T_P X$
  return X

- Example $P = \{a \leftarrow \,, \; b \leftarrow a, \neg c\}$

$$
\begin{array}{ccc}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b\}
\end{array}
$$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$

$$
\begin{array}{rrrr}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X = \{a, b\} \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b\} &
\end{array}
$$

Potassco

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$
  while $T_P X \neq X$
  $X := T_P X$
  return X

- Example $P = \{a \leftarrow \,,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$
  while $T_P X \neq X$
  $X := T_P X$
  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$

  $X_0 = \emptyset$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$

$$X_0 = \emptyset$$
$$T_P X_0 = \{a\}$$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$

  $X_0 = \emptyset \qquad X_1 = \{a\}$

  $T_P X_0 = \{a\}$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$

$$\begin{array}{rlrl} X_0 &= \emptyset & X_1 &= \{a\} \\ T_P X_0 &= \{a\} & T_P X_1 &= \{a, b\} \end{array}$$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$

$$X_0 = \emptyset \qquad X_1 = \{a\} \qquad X_2 = \{a, b\}$$
$$T_P X_0 = \{a\} \qquad T_P X_1 = \{a, b\}$$

What is the meaning of a logic program?

- Procedural attempt

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$
  while $T_P X \neq X$
    $X := T_P X$
  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$

$$\begin{array}{ccc} X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} \\ T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b, c\} \end{array}$$

What is the meaning of a logic program?

- Procedural attempt

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

$X := \emptyset$

while $T_P X \neq X$

$X := T_P X$

return X

- Example $P = \{a \leftarrow , \ b \leftarrow a, \neg c, \ c \leftarrow b\}$

$$\begin{array}{cccc} X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X_3 = \{a, b, c\} \\ T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b, c\} & \end{array}$$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$

$$
\begin{array}{cccc}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X_3 = \{a, b, c\} \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b, c\} & T_P X_3 = \{a, c\}
\end{array}
$$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow, \; b \leftarrow a, \neg c, \; c \leftarrow b\}$

$$X_0 = \emptyset \qquad X_1 = \{a\} \qquad X_2 = \{a, b\} \qquad X_3 = \{a, b, c\}$$
$$T_P X_0 = \{a\} \qquad T_P X_1 = \{a, b\} \qquad T_P X_2 = \{a, b, c\} \qquad T_P X_3 = \{a, c\}$$

$$X_4 = \{a, c\}$$

Potassco

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$

$$\begin{array}{cccc} X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X_3 = \{a, b, c\} \\ T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b, c\} & T_P X_3 = \{a, c\} \end{array}$$

$$\begin{array}{c} X_4 = \{a, c\} \\ T_P X_4 = \{a\} \end{array}$$

Potassco

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$

$$X_0 = \emptyset \quad X_1 = \{a\} \quad X_2 = \{a, b\} \quad X_3 = \{a, b, c\}$$
$$T_P X_0 = \{a\} \quad T_P X_1 = \{a, b\} \quad T_P X_2 = \{a, b, c\} \quad T_P X_3 = \{a, c\}$$

$$X_5 = \{a\} \qquad X_4 = \{a, c\}$$
$$T_P X_4 = \{a\}$$

Potassco

What is the meaning of a logic program?

- Procedural attempt

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

$X := \emptyset$

while $T_P X \neq X$

$X := T_P X$

return X

- Example $P = \{a \leftarrow ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$

$$X_0 = \emptyset \quad X_1 = \{a\} \quad X_2 = \{a, b\} \quad X_3 = \{a, b, c\}$$
$$T_P X_0 = \{a\} \quad T_P X_1 = \{a, b\} \quad T_P X_2 = \{a, b, c\} \quad T_P X_3 = \{a, c\}$$

$$X_5 = \{a\} \quad X_4 = \{a, c\}$$
$$\dots \quad T_P X_4 = \{a\}$$

What is the meaning of a logic program?

- Procedural attempt

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  return X

- Example $P = \{a \leftarrow \;,\; b \leftarrow a, \neg c,\; c \leftarrow b\}$

Potassco

What is the meaning of a logic program?

- Procedural attempt — failed

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap X = \emptyset\}$ **in**

$X := \emptyset$

while $T_P X \neq X$

$X := T_P X$

return X

What is the meaning of a logic program?

- Procedural attempt patched

guess Y

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

$X := \emptyset$

while $T_P X \neq X$

$X := T_P X$

if $X = Y$ **then return** X **else fail**

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $\quad X := \emptyset$

  $\quad$ **while** $T_P X \neq X$

  $\quad\quad X := T_P X$

  $\quad$ **if** $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$ $\qquad Y = \emptyset$

What is the meaning of a logic program?

- Procedural attempt patched

guess Y

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

$X := \emptyset$

while $T_P X \neq X$

$X := T_P X$

if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$ $\qquad Y = \emptyset$

$$\begin{array}{rcr}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b\}
\end{array}$$

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$ $\qquad Y = \emptyset$

$$\begin{array}{cccc} X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X = \{a, b\} \\ T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b\} & \end{array}$$

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow \ ,\ b \leftarrow a, \neg c\}$ $\quad Y = \emptyset$

$$
\begin{array}{cccc}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X = \{a, b\} \neq Y \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b\} &
\end{array}
$$

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$ $\quad Y = \emptyset$ ✘

$$
\begin{array}{cccc}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X = \{a, b\} \neq Y \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b\} &
\end{array}
$$

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := \ T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow \ , \ b \leftarrow a, \neg c\}$ $\qquad Y = \{a\}$

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_PX = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_PX \neq X$

  $X := T_PX$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow \ , \ b \leftarrow a, \neg c\}$ $\qquad Y = \{a\}$

$$\begin{array}{cccc} X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a,b\} & X = \{a,b\} \neq Y \\ T_PX_0 = \{a\} & T_PX_1 = \{a,b\} & T_PX_2 = \{a,b\} & \end{array}$$

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$ $\qquad Y = \{a\}$ ✘

$$\begin{array}{cccc} X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X = \{a, b\} \neq Y \\ T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b\} & \end{array}$$

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y
  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**
  $X := \emptyset$
  while $T_P X \neq X$
  $X := T_P X$
  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$ $\qquad Y = \{b\}$

What is the meaning of a logic program?

- Procedural attempt patched

guess Y

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

$X := \emptyset$

while $T_P X \neq X$

$X := T_P X$

if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$ $\quad Y = \{b\}$

$$
\begin{array}{cccc}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X = \{a, b\} \neq Y \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b\} &
\end{array}
$$

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y
  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**
  $X := \emptyset$
  while $T_P X \neq X$
  $X := T_P X$
  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow ,\ b \leftarrow a, \neg c\}$ $Y = \{b\}$ ✘

$$X_0 = \emptyset \quad X_1 = \{a\} \quad X_2 = \{a,b\} \quad X = \{a,b\} \neq Y$$
$$T_P X_0 = \{a\} \quad T_P X_1 = \{a,b\} \quad T_P X_2 = \{a,b\}$$

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$ $\qquad Y = \{a, b\}$

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$ $\quad Y = \{a, b\}$

$$
\begin{array}{cccc}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X = \{a, b\} = Y \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b\} &
\end{array}
$$

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y
  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**
  $X := \emptyset$
  while $T_P X \neq X$
  $X := T_P X$
  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c\}$ $\quad Y = \{a, b\}$ ✔

$$
\begin{array}{cccc}
X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X = \{a, b\} = Y \\
T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b\} &
\end{array}
$$

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y
  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**
  $X := \emptyset$
  while $T_P X \neq X$
  $X := T_P X$
  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow \ , \ b \leftarrow a, \neg c, \ c \leftarrow b\}$ $Y = \{a\}$

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$ $\quad Y = \{a\}$

$$\begin{array}{cccc} X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X_3 = \{a, b, c\} \\ T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b, c\} & T_P X_3 = \{a, b, c\} \end{array}$$

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $\quad X := \emptyset$

  $\quad$ **while** $T_P X \neq X$

  $\quad\quad X := T_P X$

  $\quad$ **if** $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$ $\qquad Y = \{a\}$ ✘

$$\begin{array}{cccc} X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a,b\} & X_3 = \{a,b,c\} \\ T_P X_0 = \{a\} & T_P X_1 = \{a,b\} & T_P X_2 = \{a,b,c\} & T_P X_3 = \{a,b,c\} \end{array}$$

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$ $\qquad Y = \{a, b\}$

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$ $\quad Y = \{a, b\}$

$$\begin{array}{cccc} X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X_3 = \{a, b, c\} \\ T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b, c\} & T_P X_3 = \{a, b, c\} \end{array}$$

What is the meaning of a logic program?

- Procedural attempt patched

guess Y

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

$X := \emptyset$

while $T_P X \neq X$

$X := T_P X$

if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow \ , \ b \leftarrow a, \neg c, \ c \leftarrow b\}$ $Y = \{a, b\}$ ✘

$$\begin{array}{cccc} X_0 = \emptyset & X_1 = \{a\} & X_2 = \{a, b\} & X_3 = \{a, b, c\} \\ T_P X_0 = \{a\} & T_P X_1 = \{a, b\} & T_P X_2 = \{a, b, c\} & T_P X_3 = \{a, b, c\} \end{array}$$

What is the meaning of a logic program?

- Procedural attempt patched

guess Y

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

$X := \emptyset$

while $T_P X \neq X$

$X := T_P X$

if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$ $\quad Y = \{a, b, c\}$

$X_0 = \emptyset$

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$ $\quad Y = \{a, b, c\}$

$$X_0 = \emptyset$$
$$T_P X_0 = \{a\}$$

What is the meaning of a logic program?

- Procedural attempt patched

guess Y

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

$X := \emptyset$

while $T_P X \neq X$

$X := T_P X$

if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$ $\quad Y = \{a, b, c\}$

$$X_0 = \emptyset \qquad X_1 = \{a\}$$
$$T_P X_0 = \{a\}$$

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow \ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$ $\quad Y = \{a, b, c\}$

$$X_0 = \emptyset \qquad X_1 = \{a\} \qquad X = \{a\}$$
$$T_P X_0 = \{a\} \qquad T_P X_1 = \{a\}$$

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

guess Y

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

$X := \emptyset$

while $T_P X \neq X$

$X := T_P X$

if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow\ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$ $\quad Y = \{a, b, c\}$

$X_0 = \emptyset \qquad X_1 = \{a\} \qquad X = \{a\} \neq Y$

$T_P X_0 = \{a\} \quad T_P X_1 = \{a\}$

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Example $P = \{a \leftarrow \ ,\ b \leftarrow a, \neg c,\ c \leftarrow b\}$ $\quad Y = \{a, b, c\}$ ✘

  $X_0 = \emptyset$ $\quad X_1 = \{a\}$ $\quad X = \{a\} \neq Y$

  $T_P X_0 = \{a\}$ $\quad T_P X_1 = \{a\}$

What is the meaning of a logic program?

- Procedural attempt patched — interesting ...

guess Y

let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

$X := \emptyset$

while $T_P X \neq X$

$X := T_P X$

if $X = Y$ **then return** X **else fail**

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Question Can this idea be used for a mathematical characterization ?

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Question Can this idea be used for a mathematical characterization ?
  1. guess Y
  2. evaluate $B(r)^-$ of each rule $r \in P$ wrt to Y
     1. if $B(r)^- \cap Y \neq \emptyset$ drop r
     2. if $B(r)^- \cap Y = \emptyset$ replace $B(r)$ by $B(r)^+$ from r
  3. if $Y = Cn(P')$ for the resulting (positive) program P' then ✔ else ✘

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Question Can this idea be used for a mathematical characterization ?
  1. guess Y
  2. evaluate $B(r)^-$ of each rule $r \in P$ wrt to Y
     1. if $B(r)^- \cap Y \neq \emptyset$ drop r
     2. if $B(r)^- \cap Y = \emptyset$ replace $B(r)$ by $B(r)^+$ from r
  3. if $Y = Cn(P')$ for the resulting (positive) program P' then ✔ else ✘

Potassco

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Question Can this idea be used for a mathematical characterization ?
  1. guess Y
  2. evaluate $B(r)^-$ of each rule $r \in P$ wrt to Y
     1. if $B(r)^- \cap Y \neq \emptyset$ drop r
     2. if $B(r)^- \cap Y = \emptyset$ replace $B(r)$ by $B(r)^+$ from r
  3. if $Y = Cn(P')$ for the resulting (positive) program P' then ✔ else ✘

What is the meaning of a logic program?

- Procedural attempt patched

  guess Y

  let $T_P X = \{h(r) \mid r \in P, B(r)^+ \subseteq X, B(r)^- \cap Y = \emptyset\}$ **in**

  $X := \emptyset$

  while $T_P X \neq X$

  $X := T_P X$

  if $X = Y$ **then return** X **else fail**

- Question Can this idea be used for a mathematical characterization ?
  1. guess Y
  2. evaluate $B(r)^-$ of each rule $r \in P$ wrt to Y
     1. if $B(r)^- \cap Y \neq \emptyset$ drop r
     2. if $B(r)^- \cap Y = \emptyset$ replace $B(r)$ by $B(r)^+$ from r
  3. if $Y = Cn(P')$ for the resulting (positive) program P' then ✔ else ✘

What is the meaning of a logic program?

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

What is the meaning of a logic program?

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

What is the meaning of a logic program?

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

- Note P^X can be obtained from P by
  1. deleting each rule r satisfying $B(r)^- \cap X \neq \emptyset$
     and then
  2. deleting all negative body literals from the remaining rules

What is the meaning of a logic program?

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

- Note P^X can be obtained from P by
  1. deleting each rule r satisfying $B(r)^- \cap X \neq \emptyset$
     and then
  2. deleting all negative body literals from the remaining rules

Only negative body literals are evaluated!

What is the meaning of a logic program?

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

What is the meaning of a logic program?
Stable models!

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

What is the meaning of a logic program?
Stable models!

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

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

Stable models

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

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

Stable models

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

- 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 set closed under P^X
  - Each atom in X is justified by a proof from P^X

Stable models

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

- 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 model of P^X
  - Each atom in X is justified by a proof from P^X

Stable models

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

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

Stable models

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

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

- Procedural counterpart

guess Y

let $T_{P^Y}X = \{h(r) \mid r \in P^Y, B(r)^+ \subseteq X\}$ **in**

$X := \emptyset$

while $T_{P^Y}X \neq X$

$X := T_{P^Y}X$

if $X = Y$ **then return** X **else fail**

Potassco

Stable models

- The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

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

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✗ no stable model

Potassco

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✗ no stable model

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Potassco

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Potassco

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example one

- $P = \{p \leftarrow p,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example one

- $P = \{p \leftarrow p, \ q \leftarrow \neg p\}$

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

✔ model ✘ no model

Example two

- $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example two

- $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example two

- $P = \{p \leftarrow \neg q,\ \ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Potassco

Example two

- $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example two

- $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example two

- $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example two

- $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example two

- $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example two

- $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Potassco

Example two

- $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Potassco

Example two

- $P = \{p \leftarrow \neg q, \ q \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example two

- $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$

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

✔ model ✘ no model

Example three

- $P = \{p \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example three

- $P = \{p \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Potassco

Example three

- $P = \{p \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example three

- $P = \{p \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example three

- $P = \{p \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example three

- $P = \{p \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Potasssco

Example three

- $P = \{p \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example three

- $P = \{p \leftarrow \neg p\}$

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

✔ stable model ✘ no stable model

Example three

- $P = \{p \leftarrow \neg p\}$

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

✔ model ✘ no model

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 $X \subseteq H(P)$
- If X is a stable model of a logic program P,
  then X is a model of P
- If X and Y are stable models of a normal program P,
  then $X \not\subset Y$

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 $X \subseteq H(P)$
- If X is a stable model of a logic program P, then X is a model of P
- If X and Y are stable models of a normal program P, then $X \not\subset Y$

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 $X \subseteq H(P)$
- If X is a stable model of a logic program P, then X is a model of P
- If X and Y are stable models of a normal program P, then $X \not\subset Y$

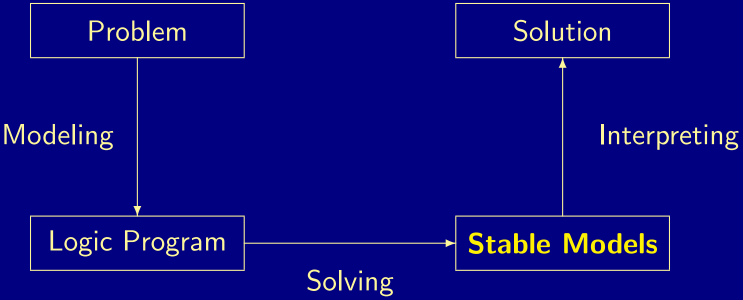
Exemplars

Logic program	Stable models
`a.`	{a}
`a :- b.`	{}
`a :- b.      b.`	{a,b}
`a :- b.      b :- a.`	{}
`a :- not c.`	{a}
`a :- not c.  c.`	{c}
`a :- not c.  c :- not a.`	{a}, {c}
`a :- not a.`	

Outline

Potassco

Semantics


Problem
Solution
Modeling
Interpreting
Logic Program
Stable Models
Solving

Reasoning modes

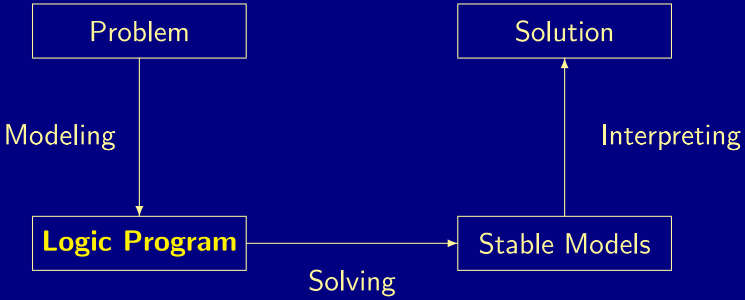
- Satisfiability
- Enumeration†
- Projection†
- Intersection‡
- Union‡
- Optimization
- and combinations of them

† without solution recording

‡ without solution enumeration

Outline

Syntax



Problem
Solution
Modeling
Interpreting
Logic Program
Stable Models
Solving

Language constructs

- Facts `q(42).`
- Rules `p(X) :- q(X), not r(X).`
- Conditional literals `p :- q(X) : r(X).`
- Disjunction `p(X) ; q(X) :- r(X).`
- Integrity constraints `:- q(X), p(X).`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y).`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7.`

- Multi-objective optimization `:~ q(X), p(X,C). [C@42]`

`#minimize { C@42 : q(X), p(X,C) }`

Language constructs

- Facts `q(42).`
- Rules `p(X) :- q(X), not r(X).`
- Conditional literals `p :- q(X) : r(X).`
- Disjunction `p(X) ; q(X) :- r(X).`
- Integrity constraints `:- q(X), p(X).`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y).`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7.`
- Multi-objective optimization `:~ q(X), p(X,C). [C@42]`

  `#minimize { C@42 : q(X), p(X,C) }`

Potassco

Language constructs

- Facts `q(42).`
- Rules `p(42) :- q(42), not r(42).`
- Conditional literals `p :- q(X) : r(X).`
- Disjunction `p(X) ; q(X) :- r(X).`
- Integrity constraints `:- q(X), p(X).`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y).`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7.`
- Multi-objective optimization `:~ q(X), p(X,C). [C@42]`

  `#minimize { C@42 : q(X), p(X,C) }`

Language constructs

- Facts `q(42).`
- Rules `p(X) :- q(X), not r(X).`
- Conditional literals `p :- q(X) : r(X).`
- Disjunction `p(X) ; q(X) :- r(X).`
- Integrity constraints `:- q(X), p(X).`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y).`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7.`
- Multi-objective optimization `:~ q(X), p(X,C). [C@42]`

  `#minimize { C@42 : q(X), p(X,C) }`

Potassco

Language constructs

- Facts `q(42).`
- Rules `p(X) :- q(X), not r(X).`
- Conditional literals `p :- q(X) : r(X).`
- Disjunction `p(X) ; q(X) :- r(X).`
- Integrity constraints `:- q(X), p(X).`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y).`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7.`
- Multi-objective optimization `:~ q(X), p(X,C). [C@42]`

  `#minimize { C@42 : q(X), p(X,C) }`

Language constructs

- Facts `q(42).`
- Rules `p(X) :- q(X), not r(X).`
- Conditional literals `p :- q(X) : r(X).`
- Disjunction `p(X) ; q(X) :- r(X).`
- Integrity constraints `:- q(X), p(X).`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y).`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7.`
- Multi-objective optimization `:~ q(X), p(X,C). [C@42]`

  `#minimize { C@42 : q(X), p(X,C) }`

Potassco

Language constructs

- Facts `q(42).`
- Rules `p(X) :- q(X), not r(X).`
- Conditional literals `p :- q(X) : r(X).`
- Disjunction `p(X) ; q(X) :- r(X).`
- Integrity constraints `:- q(X), p(X).`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y).`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7.`
- Multi-objective optimization `:~ q(X), p(X,C). [C@42]`

  `#minimize { C@42 : q(X), p(X,C) }`

Language constructs

- Facts `q(42).`
- Rules `p(X) :- q(X), not r(X).`
- Conditional literals `p :- q(X) : r(X).`
- Disjunction `p(X) ; q(X) :- r(X).`
- Integrity constraints `:- q(X), p(X).`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y).`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7.`
- Multi-objective optimization `:~ q(X), p(X,C). [C@42]`

  `#minimize { C@42 : q(X), p(X,C) }`

Potassco

Language constructs

- Facts `q(42).`
- Rules `p(X) :- q(X), not r(X).`
- Conditional literals `p :- q(X) : r(X).`
- Disjunction `p(X) ; q(X) :- r(X).`
- Integrity constraints `:- q(X), p(X).`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y).`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7.`

- Multi-objective optimization `:~ q(X), p(X,C). [C@42]`

  `#minimize { C@42 : q(X), p(X,C) }`

Potassco

Language constructs

- Facts `q(42).`
- Rules `p(X) :- q(X), not r(X).`
- Conditional literals `p :- q(X) : r(X).`
- Disjunction `p(X) ; q(X) :- r(X).`
- Integrity constraints `:- q(X), p(X).`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y).`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7.`

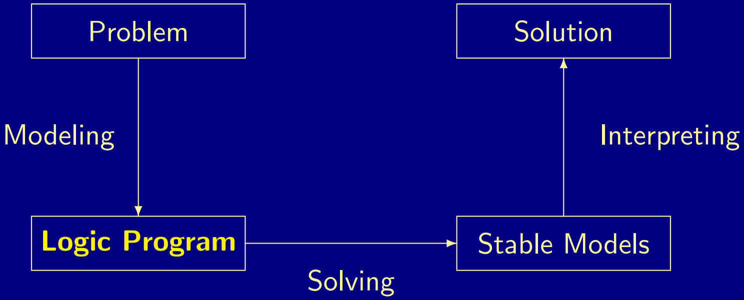
- Multi-objective optimization `:~ q(X), p(X,C). [C@42]`

  `#minimize { C@42 : q(X), p(X,C) }`

Potassco

Outline

Syntax



Problem
Solution
Modeling
Interpreting
Logic Program
Stable Models
Solving

Ground instantiation

- Let $\mathcal{T}$ be a set of (variable-free) terms
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
- A variable-free atom is also called ground
- Ground instances of a rule r are obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \to \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

  where $var(r)$ stands for the set of all variables occurring in r and θ is a (ground) substitution

- Ground instantiation of logic program P

$$ground(P) = \bigcup_{r \in P} ground(r)$$

Potassco

Ground instantiation

- Let $\mathcal{T}$ be a set of (variable-free) terms
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
- A variable-free atom is also called ground
- Ground instances of a rule r are obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \to \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

  where $var(r)$ stands for the set of all variables occurring in r and θ is a (ground) substitution
- Ground instantiation of logic program P

$$ground(P) = \bigcup_{r \in P} ground(r)$$

Ground instantiation

- Let $\mathcal{T}$ be a set of (variable-free) terms
  - Examples 42, "*coucou*", *Zorro*, $grandfather(leon)$, $3 + X$
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
  - Examples $q(42)$, $married(grandfather(leon))$, $prime(3 + X)$
- A variable-free atom is also called ground
- Ground instances of a rule r are obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \rightarrow \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

  where $var(r)$ stands for the set of all variables occurring in r and θ is a (ground) substitution
- Ground instantiation of logic program P

$$ground(P) = \bigcup_{r \in P} ground(r)$$

Ground instantiation

- Let $\mathcal{T}$ be a set of (variable-free) terms
  - Examples 42, "*coucou*", *Zorro*, $\mathit{grandfather}(\mathit{leon})$, $3 + X$
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
  - Examples $q(42)$, $\mathit{married}(\mathit{grandfather}(X))$, $\mathit{prime}(3 + X)$
- A variable-free atom is also called ground
- Ground instances of a rule r are obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$\mathit{ground}(r) = \{r\theta \mid \theta : \mathit{var}(r) \rightarrow \mathcal{T} \text{ and } \mathit{var}(r\theta) = \emptyset\}$$

  where $\mathit{var}(r)$ stands for the set of all variables occurring in r and θ is a (ground) substitution
- Ground instantiation of logic program P

$$\mathit{ground}(P) = \bigcup_{r \in P} \mathit{ground}(r)$$

Ground instantiation

- Let $\mathcal{T}$ be a set of (variable-free) terms
  - Examples 42, "*coucou*", *Zorro*, *grandfather*(*leon*), $3 + X$
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
  - Examples $q(42)$, $married(X)$, $prime(3 + X)$
- A variable-free atom is also called ground
- Ground instances of a rule r are obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \to \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

  where $var(r)$ stands for the set of all variables occurring in r and θ is a (ground) substitution
- Ground instantiation of logic program P

$$ground(P) = \bigcup_{r \in P} ground(r)$$

Ground instantiation

- Let $\mathcal{T}$ be a set of variable-free terms (also called Herbrand universe)
- Let $\mathcal{A}$ be a set of variable-free atoms constructible from $\mathcal{T}$ (also called Herbrand base)
- A variable-free atom is also called ground
- Ground instances of a rule r are obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \rightarrow \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

  where $var(r)$ stands for the set of all variables occurring in r and θ is a (ground) substitution
- Ground instantiation of logic program P

$$ground(P) = \bigcup_{r \in P} ground(r)$$

Potassco

Ground instantiation

- Let $\mathcal{T}$ be a set of variable-free terms
- Let $\mathcal{A}$ be a set of variable-free atoms constructible from $\mathcal{T}$ (also called alphabet)
- A variable-free atom is also called ground
- Ground instances of a rule r are obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \to \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

  where $var(r)$ stands for the set of all variables occurring in r and θ is a (ground) substitution
- Ground instantiation of logic program P

$$ground(P) = \bigcup_{r \in P} ground(r)$$

Potassco

Ground instantiation

- Let $\mathcal{T}$ be a set of (variable-free) terms
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
- A variable-free atom is also called ground
- Ground instances of a rule r are obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \rightarrow \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

where $var(r)$ stands for the set of all variables occurring in r and θ is a (ground) substitution

- Ground instantiation of logic program P

$$ground(P) = \bigcup_{r \in P} ground(r)$$

Ground instantiation

- Let $\mathcal{T}$ be a set of (variable-free) terms
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
- A variable-free atom is also called ground
- Ground instances of a rule r are obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \rightarrow \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

where $var(r)$ stands for the set of all variables occurring in r and θ is a (ground) substitution

- Ground instantiation of logic program P

$$ground(P) = \bigcup_{r \in P} ground(r)$$

Potassco

Ground instantiation

- Let $\mathcal{T}$ be a set of (variable-free) terms
- Let $\mathcal{A}$ be a set of (variable-free) atoms constructible from $\mathcal{T}$
- A variable-free atom is also called ground
- Ground instances of a rule r are obtained by replacing all variables in r by elements from $\mathcal{T}$:

$$ground(r) = \{r\theta \mid \theta : var(r) \rightarrow \mathcal{T} \text{ and } var(r\theta) = \emptyset\}$$

  where $var(r)$ stands for the set of all variables occurring in r and θ is a (ground) substitution
- Ground instantiation of logic program P

$$ground(P) = \bigcup_{r \in P} ground(r)$$

An example

$$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$$

$$\mathcal{T} = \{a,b,c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{lllllllll} r(a,b) & \leftarrow & , & & & & & & \\ r(b,c) & \leftarrow & , & & & & & & \\ t(a,a) & \leftarrow & r(a,a), & t(b,a) & \leftarrow & r(b,a), & t(c,a) & \leftarrow & r(c,a), \\ t(a,b) & \leftarrow & r(a,b), & t(b,b) & \leftarrow & r(b,b), & t(c,b) & \leftarrow & r(c,b), \\ t(a,c) & \leftarrow & r(a,c), & t(b,c) & \leftarrow & r(b,c), & t(c,c) & \leftarrow & r(c,c) \end{array} \right\}$$

Grounding aims at reducing the ground instantiation

by applying semantic principles

Potassco

An example

$$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$$

$$\mathcal{T} = \{a,b,c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{llllllll} r(a,b) & \leftarrow & , & & & & & \\ r(b,c) & \leftarrow & , & & & & & \\ t(a,a) & \leftarrow & r(a,a), & t(b,a) & \leftarrow & r(b,a), & t(c,a) & \leftarrow r(c,a), \\ t(a,b) & \leftarrow & r(a,b), & t(b,b) & \leftarrow & r(b,b), & t(c,b) & \leftarrow r(c,b), \\ t(a,c) & \leftarrow & r(a,c), & t(b,c) & \leftarrow & r(b,c), & t(c,c) & \leftarrow r(c,c) \end{array} \right\}$$

Grounding aims at reducing the ground instantiation by applying semantic principles

An example

$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$

$\mathcal{T} = \{a,b,c\}$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{lllllllll} r(a,b) & \leftarrow & , & & & & & & \\ r(b,c) & \leftarrow & , & & & & & & \\ t(a,a) & \leftarrow & r(a,a), & t(b,a) & \leftarrow & r(b,a), & t(c,a) & \leftarrow & r(c,a), \\ t(a,b) & \leftarrow & r(a,b), & t(b,b) & \leftarrow & r(b,b), & t(c,b) & \leftarrow & r(c,b), \\ t(a,c) & \leftarrow & r(a,c), & t(b,c) & \leftarrow & r(b,c), & t(c,c) & \leftarrow & r(c,c) \end{array} \right\}$$

Grounding aims at reducing the ground instantiation
by applying semantic principles

An example

$$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$$

$$\mathcal{T} = \{a,b,c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{llllllll} r(a,b) & \leftarrow & , & & & & & \\ r(b,c) & \leftarrow & , & & & & & \\ t(a,a) & \leftarrow & r(a,a), & t(b,a) & \leftarrow & r(b,a), & t(c,a) \leftarrow & r(c,a), \\ t(a,b) & \leftarrow & r(a,b), & t(b,b) & \leftarrow & r(b,b), & t(c,b) \leftarrow & r(c,b), \\ t(a,c) & \leftarrow & r(a,c), & t(b,c) & \leftarrow & r(b,c), & t(c,c) \leftarrow & r(c,c) \end{array} \right\}$$

Grounding aims at reducing the ground instantiation

by applying semantic principles

An example

$$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$$

$$\mathcal{T} = \{a, b, c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{llllllll} r(a,b) & \leftarrow & , & & & & & \\ r(b,c) & \leftarrow & , & & & & & \\ t(a,a) & \leftarrow & r(a,a), & t(b,a) & \leftarrow & r(b,a), & t(c,a) & \leftarrow\ r(c,a), \\ t(a,b) & \leftarrow & r(a,b), & t(b,b) & \leftarrow & r(b,b), & t(c,b) & \leftarrow\ r(c,b), \\ t(a,c) & \leftarrow & r(a,c), & t(b,c) & \leftarrow & r(b,c), & t(c,c) & \leftarrow\ r(c,c) \end{array} \right\}$$

➡ Grounding aims at reducing the ground instantiation by applying semantic principles

Potassco

An example

$$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$$

$$\mathcal{T} = \{a, b, c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{lllllllll} r(a,b) & \leftarrow & , & & & & & & \\ r(b,c) & \leftarrow & , & & & & & & \\ t(a,a) & \leftarrow & r(a,a), & t(b,a) & \leftarrow & r(b,a), & t(c,a) & \leftarrow & r(c,a), \\ t(a,b) & \leftarrow & r(a,b), & t(b,b) & \leftarrow & r(b,b), & t(c,b) & \leftarrow & r(c,b), \\ t(a,c) & \leftarrow & r(a,c), & t(b,c) & \leftarrow & r(b,c), & t(c,c) & \leftarrow & r(c,c) \end{array} \right\}$$

➡ Grounding aims at reducing the ground instantiation by applying semantic principles

Potassco

An example

$$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$$

$$\mathcal{T} = \{a, b, c\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{llllll} r(a,b) \leftarrow & , & & & & \\ r(b,c) \leftarrow & , & & & & \\ t(a,a) \leftarrow & r(a,a), & t(b,a) \leftarrow & r(b,a), & t(c,a) \leftarrow & r(c,a), \\ t(a,b) \leftarrow & , & t(b,b) \leftarrow & r(b,b), & t(c,b) \leftarrow & r(c,b), \\ t(a,c) \leftarrow & r(a,c), & t(b,c) \leftarrow & r(b,c), & t(c,c) \leftarrow & r(c,c) \end{array} \right\}$$

➡ Grounding aims at reducing the ground instantiation by applying semantic principles

Potassco

An example

$$P = \{\ r(a,b) \leftarrow,\ \ r(b,c) \leftarrow,\ \ t(X,Y) \leftarrow r(X,Y)\ \}$$

$$\mathcal{T} = \{a,b,c\}$$

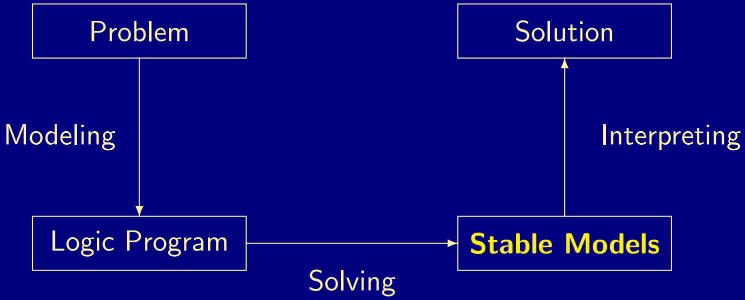
$$\mathcal{A} = \left\{ \begin{array}{l} r(a,a), r(a,b), r(a,c), r(b,a), r(b,b), r(b,c), r(c,a), r(c,b), r(c,c), \\ t(a,a), t(a,b), t(a,c), t(b,a), t(b,b), t(b,c), t(c,a), t(c,b), t(c,c) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{llllll} r(a,b) \leftarrow & , & & & & \\ r(b,c) \leftarrow & , & & & & \\ t(a,a) \leftarrow & r(a,a), & t(b,a) \leftarrow & r(b,a), & t(c,a) \leftarrow & r(c,a), \\ t(a,b) \leftarrow & , & t(b,b) \leftarrow & r(b,b), & t(c,b) \leftarrow & r(c,b), \\ t(a,c) \leftarrow & r(a,c), & t(b,c) \leftarrow & r(b,c), & t(c,c) \leftarrow & r(c,c) \end{array} \right\}$$

➥ Grounding aims at reducing the ground instantiation by applying semantic principles

Potassco

Semantics


Problem
Solution
Modeling
Interpreting
Logic Program
Stable Models
Solving

Stable models of programs with Variables

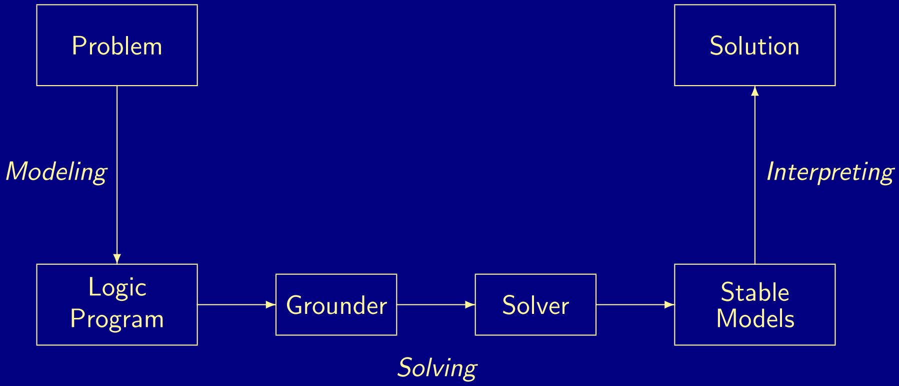
- Let P be a normal logic program with variables
- A set X of (ground) atoms is a stable model of P, if X is a stable model of $ground(P)$

Potassco

Stable models of programs with Variables

- Let P be a normal logic program with variables
- A set X of (ground) atoms is a stable model of P,
  if X is a stable model of $\mathit{ground}(P)$

Modeling, grounding, and solving



Safety

- A normal rule is safe, if all its variables occur in its positive body
- Examples
  - $p(a) \leftarrow$
  - $p(X) \leftarrow$
  - $p(X) \leftarrow q(X)$
  - $p(X) \leftarrow \neg q(X)$
  - $p(X) \leftarrow \neg q(X), r(X)$
- A normal program is safe, if all of its rules are safe

Safety

- A normal rule is safe, if all its variables occur in its positive body
- Examples
  - $p(a) \leftarrow$
  - $p(X) \leftarrow$
  - $p(X) \leftarrow q(X)$
  - $p(X) \leftarrow \neg q(X)$
  - $p(X) \leftarrow \neg q(X), r(X)$
- A normal program is safe, if all of its rules are safe

Safety

- A normal rule is safe, if all its variables occur in its positive body
- Examples
  - $p(a) \leftarrow$
  - $p(X) \leftarrow$
  - $p(X) \leftarrow q(X)$
  - $p(X) \leftarrow \neg q(X)$
  - $p(X) \leftarrow \neg q(X), r(X)$
- A normal program is safe, if all of its rules are safe

Safety

- A normal rule is safe, if all its variables occur in its positive body
- Examples
  - $p(a) \leftarrow$ ✔
  - $p(X) \leftarrow$
  - $p(X) \leftarrow q(X)$
  - $p(X) \leftarrow \neg q(X)$
  - $p(X) \leftarrow \neg q(X), r(X)$
- A normal program is safe, if all of its rules are safe

Safety

- A normal rule is safe, if all its variables occur in its positive body
- Examples
  - $p(a) \leftarrow$ ✔
  - $p(X) \leftarrow$ ✘
  - $p(X) \leftarrow q(X)$
  - $p(X) \leftarrow \neg q(X)$
  - $p(X) \leftarrow \neg q(X), r(X)$
- A normal program is safe, if all of its rules are safe

Safety

- A normal rule is safe, if all its variables occur in its positive body
- Examples
  - $p(a) \leftarrow$ ✔
  - $p(X) \leftarrow$ ✘
  - $p(X) \leftarrow q(X)$ ✔
  - $p(X) \leftarrow \neg q(X)$
  - $p(X) \leftarrow \neg q(X), r(X)$
- A normal program is safe, if all of its rules are safe

Safety

- A normal rule is **safe**, if all its variables occur in its positive body
- Examples
  - $p(a) \leftarrow$ ✔
  - $p(X) \leftarrow$ ✘
  - $p(X) \leftarrow q(X)$ ✔
  - $p(X) \leftarrow \neg q(X)$ ✘
  - $p(X) \leftarrow \neg q(X), r(X)$
- A normal program is safe, if all of its rules are safe

Safety

- A normal rule is safe, if all its variables occur in its positive body
- Examples
  - $p(a) \leftarrow$ ✔
  - $p(X) \leftarrow$ ✘
  - $p(X) \leftarrow q(X)$ ✔
  - $p(X) \leftarrow \neg q(X)$ ✘
  - $p(X) \leftarrow \neg q(X), r(X)$ ✔
- A normal program is safe, if all of its rules are safe

Safety

- A normal rule is safe, if all its variables occur in its positive body
- Examples
  - $p(a) \leftarrow$ ✔
  - $p(X) \leftarrow$ ✘
  - $p(X) \leftarrow q(X)$ ✔
  - $p(X) \leftarrow \neg q(X)$ ✘
  - $p(X) \leftarrow \neg q(X), r(X)$ ✔
- A normal program is safe, if all of its rules are safe

Potassco

Grounding safe programs

- $P = \{\, r(a,b) \leftarrow\ ,\ r(b,c) \leftarrow\ ,\ t(X,Y) \leftarrow r(X,Y)\,\}$

- Grounding intuitively
  1. Partition program along predicate dependencies
     - $P_1 = \{\, r(a,b) \leftarrow\ ,\ r(b,c) \leftarrow\,\}$
     - $P_2 = \{\, t(X,Y) \leftarrow r(X,Y)\,\}$
  2. Ground P_1

     Rules: $\{\, r(a,b) \leftarrow\ ,\ r(b,c) \leftarrow\,\}$

     Atoms: $\{\, r(a,b), r(b,c)\,\}$
  3. Ground P_2 relative to $\{\, r(a,b), r(b,c)\,\}$

     Rules: $\{\, t(a,b) \leftarrow r(a,b),\ t(b,c) \leftarrow r(b,c)\,\}$

     Atoms: $\{\, r(a,b), r(b,c), t(a,b), t(b,c)\,\}$
  4. Resulting ground rules

     $\{r(a,b) \leftarrow\ ,\ r(b,c) \leftarrow\,\} \cup \{t(a,b) \leftarrow r(a,b),\ t(b,c) \leftarrow r(b,c)\}$

Grounding safe programs

- $P = \{ r(a,b) \leftarrow ,\ r(b,c) \leftarrow ,\ t(X,Y) \leftarrow r(X,Y) \}$
- Grounding intuitively
  0. Partition program along predicate dependencies
     - $P_1 = \{ r(a,b) \leftarrow ,\ r(b,c) \leftarrow \}$
     - $P_2 = \{ t(X,Y) \leftarrow r(X,Y) \}$
  1. Ground P_1
     - Rules: $\{ r(a,b) \leftarrow ,\ r(b,c) \leftarrow \}$
     - Atoms: $\{ r(a,b), r(b,c) \}$
  2. Ground P_2 relative to $\{ r(a,b), r(b,c) \}$
     - Rules: $\{ t(a,b) \leftarrow r(a,b),\ t(b,c) \leftarrow r(b,c) \}$
     - Atoms: $\{ r(a,b), r(b,c), t(a,b), t(b,c) \}$
  3. Resulting ground rules
     - $\{r(a,b) \leftarrow ,\ r(b,c) \leftarrow \} \cup \{t(a,b) \leftarrow r(a,b),\ t(b,c) \leftarrow r(b,c)\}$

Grounding safe programs

- $P = \{ r(a,b) \leftarrow ,\ r(b,c) \leftarrow ,\ t(X,Y) \leftarrow r(X,Y) \}$
- Grounding intuitively
  0. Partition program along predicate dependencies
     - $P_1 = \{ r(a,b) \leftarrow ,\ r(b,c) \leftarrow \}$
     - $P_2 = \{ t(X,Y) \leftarrow r(X,Y) \}$
  1. Ground P_1
     - Rules: $\{ r(a,b) \leftarrow ,\ r(b,c) \leftarrow \}$
     - Atoms: $\{ r(a,b), r(b,c) \}$
  2. Ground P_2 relative to $\{ r(a,b), r(b,c) \}$
     - Rules: $\{ t(a,b) \leftarrow r(a,b),\ t(b,c) \leftarrow r(b,c) \}$
     - Atoms: $\{ r(a,b), r(b,c), t(a,b), t(b,c) \}$
  3. Resulting ground rules
     - $\{r(a,b) \leftarrow ,\ r(b,c) \leftarrow \} \cup \{t(a,b) \leftarrow r(a,b),\ t(b,c) \leftarrow r(b,c)\}$

Grounding safe programs

- $P = \{ r(a,b) \leftarrow , \ r(b,c) \leftarrow , \ t(X,Y) \leftarrow r(X,Y) \}$
- Grounding intuitively
  0. Partition program along predicate dependencies
     - $P_1 = \{ r(a,b) \leftarrow , \ r(b,c) \leftarrow \}$
     - $P_2 = \{ t(X,Y) \leftarrow r(X,Y) \}$
  1. Ground P_1
     - Rules: $\{ r(a,b) \leftarrow , \ r(b,c) \leftarrow \}$
     - Atoms: $\{ r(a,b), r(b,c) \}$
  2. Ground P_2 relative to $\{ r(a,b), r(b,c) \}$
     - Rules: $\{ t(a,b) \leftarrow r(a,b), \ t(b,c) \leftarrow r(b,c) \}$
     - Atoms: $\{ r(a,b), r(b,c), t(a,b), t(b,c) \}$
  3. Resulting ground rules
     - $\{r(a,b) \leftarrow , \ r(b,c) \leftarrow \} \cup \{t(a,b) \leftarrow r(a,b), \ t(b,c) \leftarrow r(b,c)\}$

Potassco

Grounding safe programs

- $P = \{ r(a,b) \leftarrow,\ r(b,c) \leftarrow,\ t(X,Y) \leftarrow r(X,Y) \}$
- Grounding intuitively
  0. Partition program along predicate dependencies
     - $P_1 = \{ r(a,b) \leftarrow,\ r(b,c) \leftarrow \}$
     - $P_2 = \{ t(X,Y) \leftarrow r(X,Y) \}$
  1. Ground P_1
     - Rules: $\{ r(a,b) \leftarrow,\ r(b,c) \leftarrow \}$
     - Atoms: $\{ r(a,b), r(b,c) \}$
  2. Ground P_2 relative to $\{ r(a,b), r(b,c) \}$
     - Rules: $\{ t(a,b) \leftarrow r(a,b),\ t(b,c) \leftarrow r(b,c) \}$
     - Atoms: $\{ r(a,b), r(b,c), t(a,b), t(b,c) \}$
  3. Resulting ground rules
     - $\{r(a,b) \leftarrow,\ r(b,c) \leftarrow \} \cup \{t(a,b) \leftarrow r(a,b),\ t(b,c) \leftarrow r(b,c)\}$

Grounding safe programs

- $P = \{ r(a,b) \leftarrow ,\ r(b,c) \leftarrow ,\ t(X,Y) \leftarrow r(X,Y) \}$
- Grounding intuitively
  0. Partition program along predicate dependencies
     - $P_1 = \{ r(a,b) \leftarrow ,\ r(b,c) \leftarrow \}$
     - $P_2 = \{ t(X,Y) \leftarrow r(X,Y) \}$
  1. Ground P_1
     - Rules: $\{ r(a,b) \leftarrow ,\ r(b,c) \leftarrow \}$
     - Atoms: $\{ r(a,b), r(b,c) \}$
  2. Ground P_2 relative to $\{ r(a,b), r(b,c) \}$
     - Rules: $\{ t(a,b) \leftarrow r(a,b),\ t(b,c) \leftarrow r(b,c) \}$
     - Atoms: $\{ r(a,b), r(b,c), t(a,b), t(b,c) \}$
  3. Resulting ground rules
     - $\{r(a,b) \leftarrow ,\ r(b,c) \leftarrow \} \cup \{t(a,b) \leftarrow r(a,b),\ t(b,c) \leftarrow r(b,c)\}$

Outline

Things to remember

- Syntax
  - atoms, literals, heads, bodies, rules, positive rules, positive and normal logic programs
  - terms, variables, safety, ground terms, atoms, and rules
  - integrity constraints, cardinality and weight constraints, optimization statements
- Semantics
  - assignments/solutions, interpretations/models, sets/closed sets
  - T_P operator, fixpoint
  - closure, reduct, stable models

ASP's syntax and semantics in a nutshell

- Syntax
  - 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

$$a_0 \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

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

- Semantics
  - The reduct, P^X, of a program P relative to a set X of atoms is

$$P^X = \{h(r) \leftarrow B(r)^+ \mid r \in P,\ B(r)^- \cap X = \emptyset\}$$

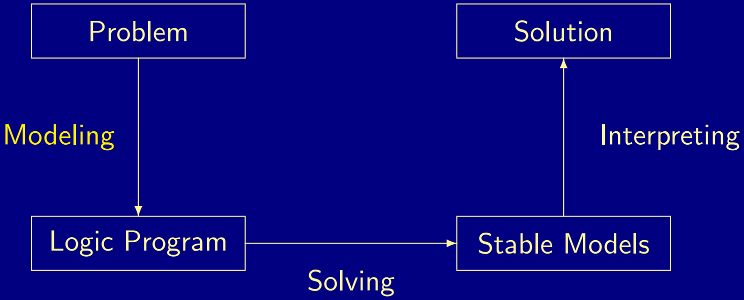
  - A set X of atoms is a stable model of a program P if $Cn(P^X) = X$

Potassco

Basic Modeling: Overview

Potassco

Modeling



Potassco

Outline

Guiding principle

- Elaboration Tolerance (McCarthy, 1998)

  "A formalism is elaboration tolerant [if] *it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances."*

- Uniform problem representation

  For solving a problem instance **I** of a problem class **C**,
  - **I** is represented as a set of facts $P_{\mathbf{I}}$,
  - **C** is represented as a set of rules $P_{\mathbf{C}}$, and
  - $P_{\mathbf{C}}$ can be used to solve all problem instances in **C**

Guiding principle

- Elaboration Tolerance (McCarthy, 1998)

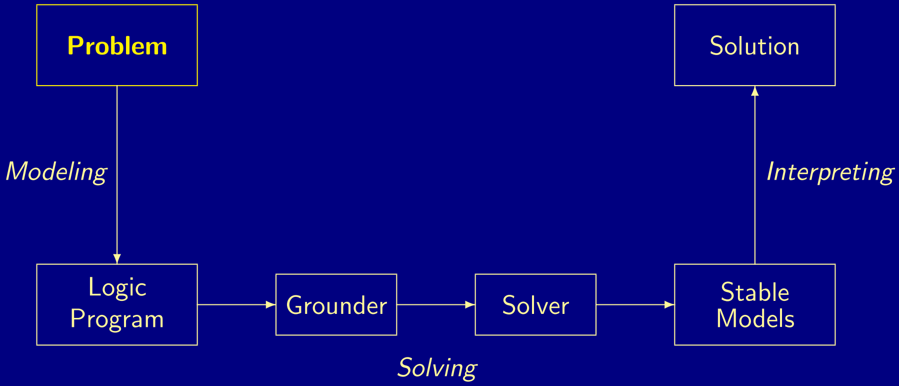
  "A formalism is elaboration tolerant [if] *it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances."*

- Uniform problem representation

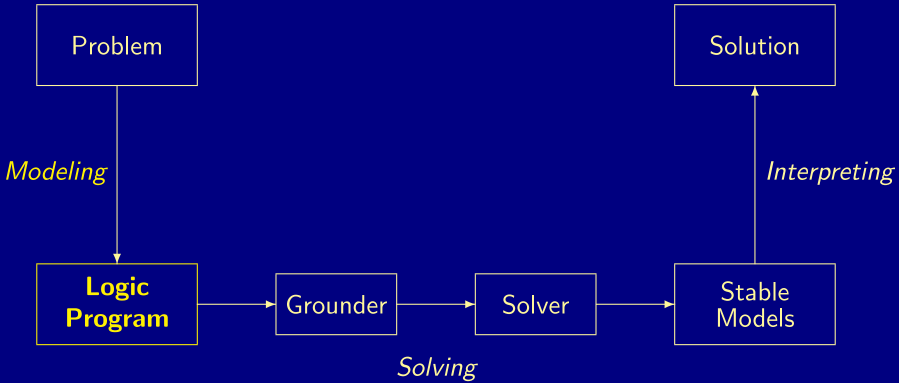
  For solving a problem instance **I** of a problem class **C**,
  - **I** is represented as a set of facts $P_{\mathbf{I}}$,
  - **C** is represented as a set of rules $P_{\mathbf{C}}$, and
  - $P_{\mathbf{C}}$ can be used to solve all problem instances in **C**

Outline

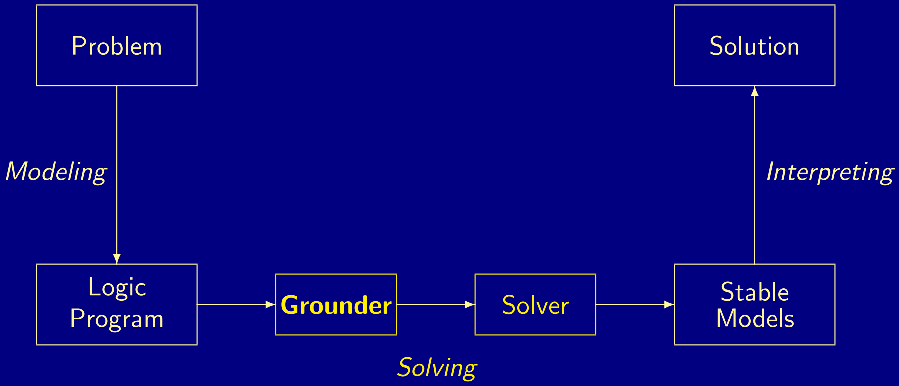
ASP workflow



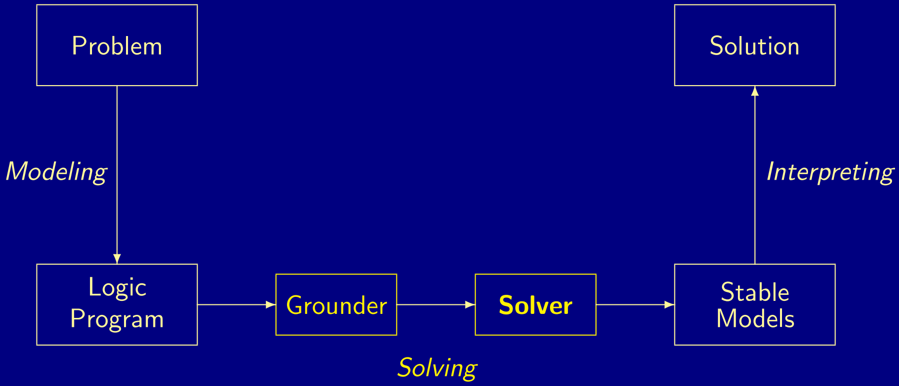
ASP workflow



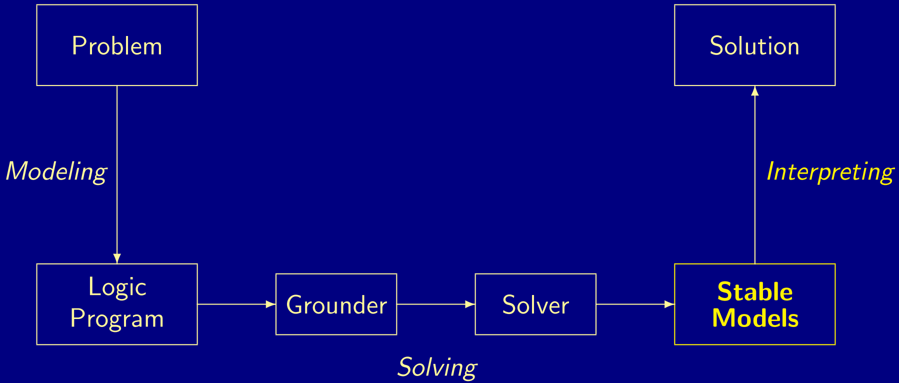
ASP workflow



ASP workflow

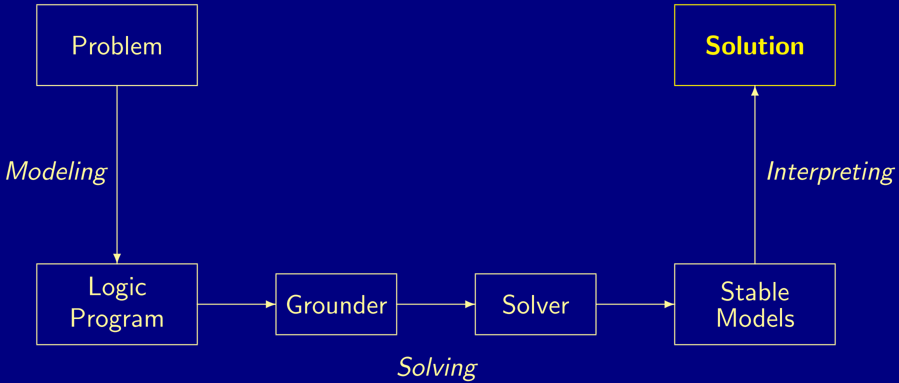


ASP workflow

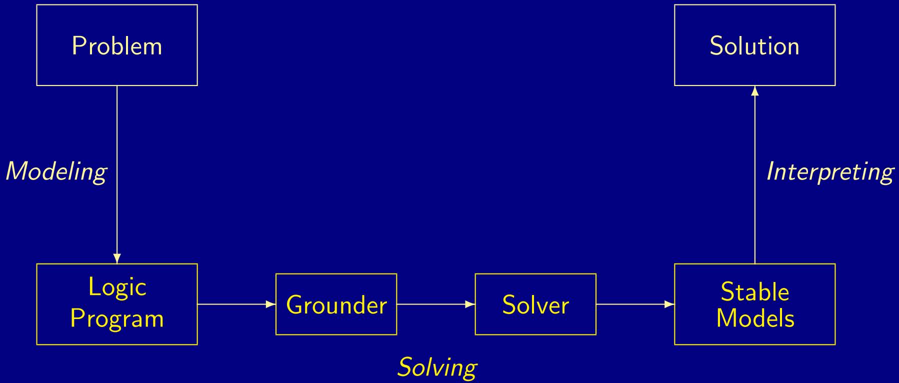


Potassco

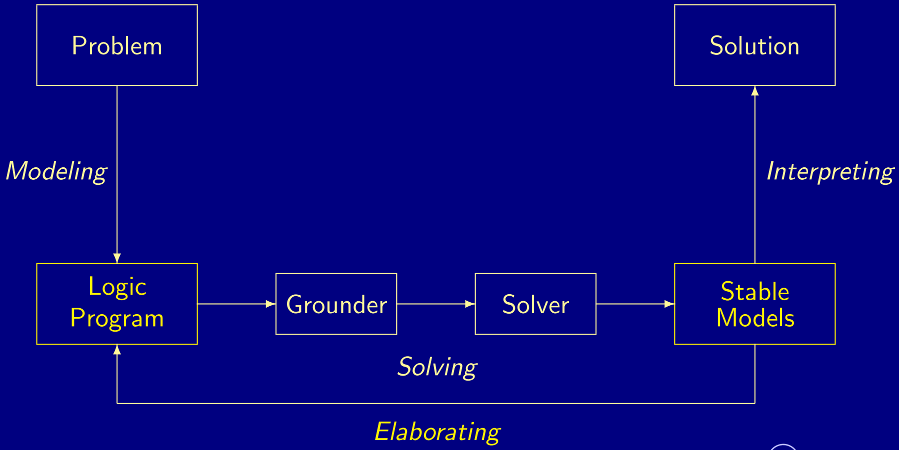
ASP workflow



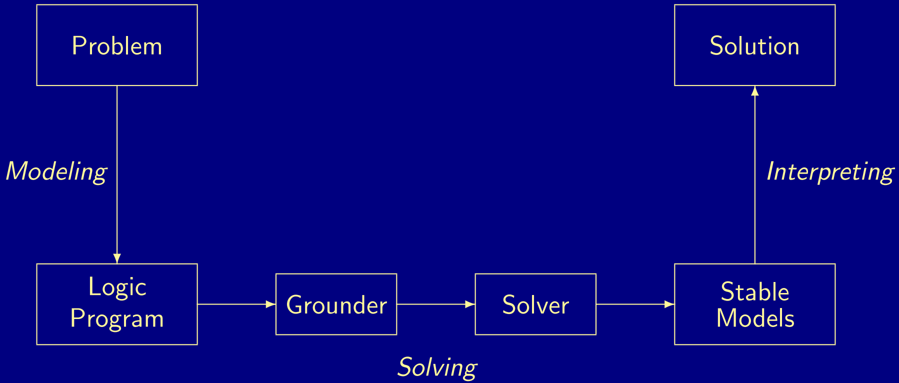
ASP workflow



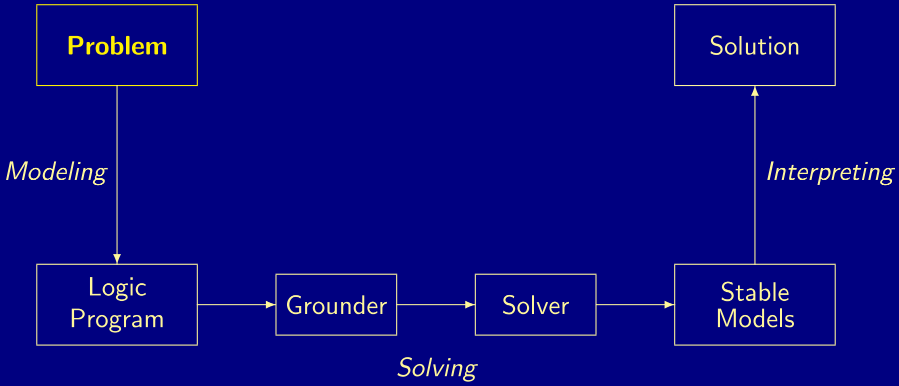
ASP workflow



ASP workflow



ASP workflow: Problem



A case-study: Graph coloring

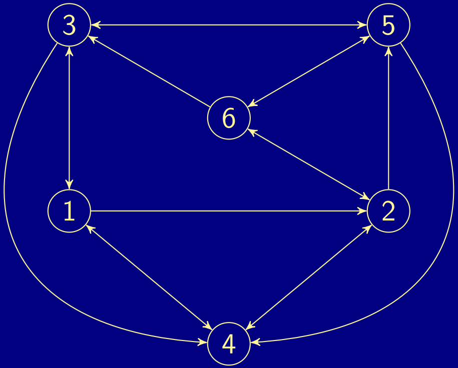
- Problem instance A graph consisting of nodes and edges

A case-study: Graph coloring

- Problem instance A graph consisting of nodes and edges

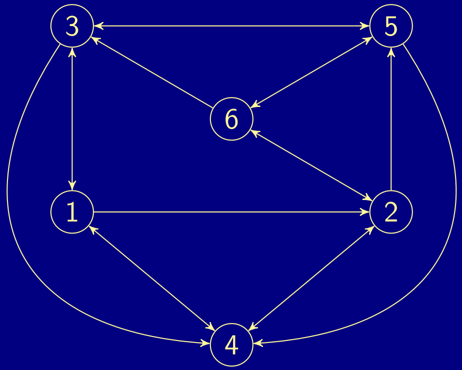
A case-study: Graph coloring

- Problem instance A graph consisting of nodes and edges



A case-study: Graph coloring

- Problem instance A graph consisting of nodes and edges
  - facts formed by predicates `node/1` and `edge/2`



A case-study: Graph coloring

- Problem instance A graph consisting of nodes and edges
  - facts formed by predicates `node/1` and `edge/2`
  - facts formed by predicate `color/1`

A case-study: Graph coloring

- Problem instance A graph consisting of nodes and edges
  - facts formed by predicates `node/1` and `edge/2`
  - facts formed by predicate `color/1`

- Problem class Assign each node one color such that no two nodes connected by an edge have the same color

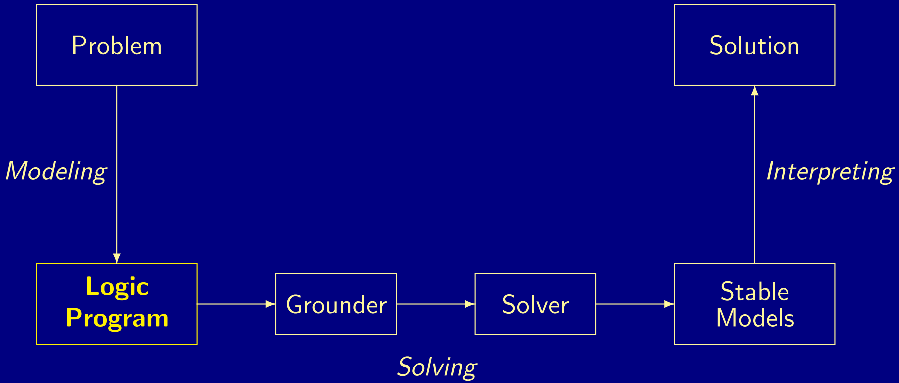
A case-study: Graph coloring

- Problem instance A graph consisting of nodes and edges
  - facts formed by predicates `node/1` and `edge/2`
  - facts formed by predicate `color/1`
- Problem class Assign each node one color such that no two nodes connected by an edge have the same color

  In other words,

  1. Each node has one color
  2. Two connected nodes must not have the same color

ASP workflow: Problem representation



Graph coloring

```
node(1..6).

edge(1,2).   edge(1,3).   edge(1,4).
edge(2,4).   edge(2,5).   edge(2,6).
edge(3,1).   edge(3,4).   edge(3,5).
edge(4,1).   edge(4,2).
edge(5,3).   edge(5,4).   edge(5,6).
edge(6,2).   edge(6,3).   edge(6,5).

color(r).    color(b).    color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).   edge(1,3).   edge(1,4).
edge(2,4).   edge(2,5).   edge(2,6).
edge(3,1).   edge(3,4).   edge(3,5).
edge(4,1).   edge(4,2).
edge(5,3).   edge(5,4).   edge(5,6).
edge(6,2).   edge(6,3).   edge(6,5).

color(r).    color(b).    color(g).

{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem instance

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).   edge(1,3).   edge(1,4).
edge(2,4).   edge(2,5).   edge(2,6).
edge(3,1).   edge(3,4).   edge(3,5).
edge(4,1).   edge(4,2).
edge(5,3).   edge(5,4).   edge(5,6).
edge(6,2).   edge(6,3).   edge(6,5).

color(r).    color(b).    color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).   edge(1,3).   edge(1,4).
edge(2,4).   edge(2,5).   edge(2,6).
edge(3,1).   edge(3,4).   edge(3,5).
edge(4,1).   edge(4,2).
edge(5,3).   edge(5,4).   edge(5,6).
edge(6,2).   edge(6,3).   edge(6,5).

color(r).    color(b).    color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).   edge(1,3).   edge(1,4).
edge(2,4).   edge(2,5).   edge(2,6).
edge(3,1).   edge(3,4).   edge(3,5).
edge(4,1).   edge(4,2).
edge(5,3).   edge(5,4).   edge(5,6).
edge(6,2).   edge(6,3).   edge(6,5).

color(r).    color(b).    color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).   edge(1,3).   edge(1,4).
edge(2,4).   edge(2,5).   edge(2,6).
edge(3,1).   edge(3,4).   edge(3,5).
edge(4,1).   edge(4,2).
edge(5,3).   edge(5,4).   edge(5,6).
edge(6,2).   edge(6,3).   edge(6,5).

color(r).    color(b).    color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Graph coloring

```
node(1..6).

edge(1,2).   edge(1,3).   edge(1,4).
edge(2,4).   edge(2,5).   edge(2,6).
edge(3,1).   edge(3,4).   edge(3,5).
edge(4,1).   edge(4,2).
edge(5,3).   edge(5,4).   edge(5,6).
edge(6,2).   edge(6,3).   edge(6,5).

color(r).    color(b).    color(g).
```

Problem instance

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

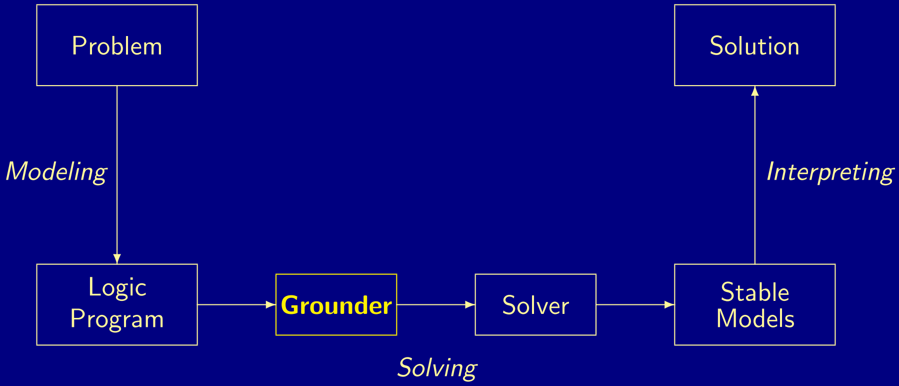
$\}$ graph.lp

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

$\}$ color.lp

Potassco

ASP workflow: Grounding



Graph coloring: Grounding

```
$ gringo --text graph.lp color.lp
```

```
node(1).  node(2).  node(3).  node(4).  node(5).  node(6).

edge(1,2).  edge(2,4).  edge(3,1).  edge(4,1).  edge(5,3).  edge(6,2).
edge(1,3).  edge(2,5).  edge(3,4).  edge(4,2).  edge(5,4).  edge(6,3).
edge(1,4).  edge(2,6).  edge(3,5).              edge(5,6).  edge(6,5).

color(r).  color(b).  color(g).

{assign(1,r);assign(1,b);assign(1,g)} = 1. {assign(4,r);assign(4,b);assign(4,g)} = 1.
{assign(2,r);assign(2,b);assign(2,g)} = 1. {assign(5,r);assign(5,b);assign(5,g)} = 1.
{assign(3,r);assign(3,b);assign(3,g)} = 1. {assign(6,r);assign(6,b);assign(6,g)} = 1.

:- assign(1,r),assign(2,r).  :- assign(2,r),assign(4,r). [...] :- assign(6,r),assign(2,r).
:- assign(1,b),assign(2,b).  :- assign(2,b),assign(4,b).       :- assign(6,b),assign(2,b).
:- assign(1,g),assign(2,g).  :- assign(2,g),assign(4,g).       :- assign(6,g),assign(2,g).
:- assign(1,r),assign(3,r).  :- assign(2,r),assign(5,r).       :- assign(6,r),assign(3,r).
:- assign(1,b),assign(3,b).  :- assign(2,b),assign(5,b).       :- assign(6,b),assign(3,b).
:- assign(1,g),assign(3,g).  :- assign(2,g),assign(5,g).       :- assign(6,g),assign(3,g).
:- assign(1,r),assign(4,r).  :- assign(2,r),assign(6,r).       :- assign(6,r),assign(5,r).
:- assign(1,b),assign(4,b).  :- assign(2,b),assign(6,b).       :- assign(6,b),assign(5,b).
:- assign(1,g),assign(4,g).  :- assign(2,g),assign(6,g).       :- assign(6,g),assign(5,g).
```

Graph coloring: Grounding

```
$ gringo --text graph.lp color.lp

node(1).  node(2).  node(3).  node(4).  node(5).  node(6).

edge(1,2).  edge(2,4).  edge(3,1).  edge(4,1).  edge(5,3).  edge(6,2).
edge(1,3).  edge(2,5).  edge(3,4).  edge(4,2).  edge(5,4).  edge(6,3).
edge(1,4).  edge(2,6).  edge(3,5).              edge(5,6).  edge(6,5).

color(r).  color(b).  color(g).

{assign(1,r);assign(1,b);assign(1,g)} = 1. {assign(4,r);assign(4,b);assign(4,g)} = 1.
{assign(2,r);assign(2,b);assign(2,g)} = 1. {assign(5,r);assign(5,b);assign(5,g)} = 1.
{assign(3,r);assign(3,b);assign(3,g)} = 1. {assign(6,r);assign(6,b);assign(6,g)} = 1.

:- assign(1,r),assign(2,r).  :- assign(2,r),assign(4,r). [...] :- assign(6,r),assign(2,r).
:- assign(1,b),assign(2,b).  :- assign(2,b),assign(4,b).       :- assign(6,b),assign(2,b).
:- assign(1,g),assign(2,g).  :- assign(2,g),assign(4,g).       :- assign(6,g),assign(2,g).
:- assign(1,r),assign(3,r).  :- assign(2,r),assign(5,r).       :- assign(6,r),assign(3,r).
:- assign(1,b),assign(3,b).  :- assign(2,b),assign(5,b).       :- assign(6,b),assign(3,b).
:- assign(1,g),assign(3,g).  :- assign(2,g),assign(5,g).       :- assign(6,g),assign(3,g).
:- assign(1,r),assign(4,r).  :- assign(2,r),assign(6,r).       :- assign(6,r),assign(5,r).
:- assign(1,b),assign(4,b).  :- assign(2,b),assign(6,b).       :- assign(6,b),assign(5,b).
:- assign(1,g),assign(4,g).  :- assign(2,g),assign(6,g).       :- assign(6,g),assign(5,g).
```

Graph coloring: Grounding

```
$ gringo --text graph.lp color.lp

node(1).  node(2).  node(3).  node(4).  node(5).  node(6).

edge(1,2).  edge(2,4).  edge(3,1).  edge(4,1).  edge(5,3).  edge(6,2).
edge(1,3).  edge(2,5).  edge(3,4).  edge(4,2).  edge(5,4).  edge(6,3).
edge(1,4).  edge(2,6).  edge(3,5).              edge(5,6).  edge(6,5).

color(r).  color(b).  color(g).

{assign(1,r);assign(1,b);assign(1,g)} = 1. {assign(4,r);assign(4,b);assign(4,g)} = 1.
{assign(2,r);assign(2,b);assign(2,g)} = 1. {assign(5,r);assign(5,b);assign(5,g)} = 1.
{assign(3,r);assign(3,b);assign(3,g)} = 1. {assign(6,r);assign(6,b);assign(6,g)} = 1.

:- assign(1,r),assign(2,r).  :- assign(2,r),assign(4,r). [...] :- assign(6,r),assign(2,r).
:- assign(1,b),assign(2,b).  :- assign(2,b),assign(4,b).       :- assign(6,b),assign(2,b).
:- assign(1,g),assign(2,g).  :- assign(2,g),assign(4,g).       :- assign(6,g),assign(2,g).
:- assign(1,r),assign(3,r).  :- assign(2,r),assign(5,r).       :- assign(6,r),assign(3,r).
:- assign(1,b),assign(3,b).  :- assign(2,b),assign(5,b).       :- assign(6,b),assign(3,b).
:- assign(1,g),assign(3,g).  :- assign(2,g),assign(5,g).       :- assign(6,g),assign(3,g).
:- assign(1,r),assign(4,r).  :- assign(2,r),assign(6,r).       :- assign(6,r),assign(5,r).
:- assign(1,b),assign(4,b).  :- assign(2,b),assign(6,b).       :- assign(6,b),assign(5,b).
:- assign(1,g),assign(4,g).  :- assign(2,g),assign(6,g).       :- assign(6,g),assign(5,g).
```

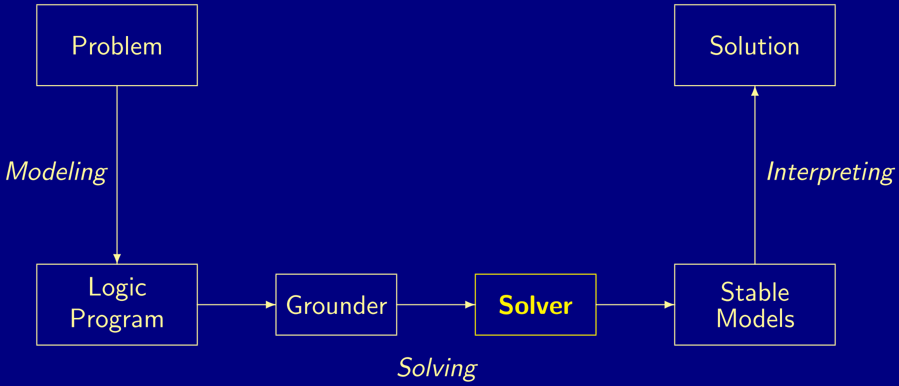
Graph coloring: Grounding

```
$ gringo --text graph.lp color.lp

node(1).  node(2).  node(3).  node(4).  node(5).  node(6).

edge(1,2).  edge(2,4).  edge(3,1).  edge(4,1).  edge(5,3).  edge(6,2).
edge(1,3).  edge(2,5).  edge(3,4).  edge(4,2).  edge(5,4).  edge(6,3).
edge(1,4).  edge(2,6).  edge(3,5).              edge(5,6).  edge(6,5).

color(r).  color(b).  color(g).

{assign(1,r);assign(1,b);assign(1,g)} = 1. {assign(4,r);assign(4,b);assign(4,g)} = 1.
{assign(2,r);assign(2,b);assign(2,g)} = 1. {assign(5,r);assign(5,b);assign(5,g)} = 1.
{assign(3,r);assign(3,b);assign(3,g)} = 1. {assign(6,r);assign(6,b);assign(6,g)} = 1.

:- assign(1,r),assign(2,r).  :- assign(2,r),assign(4,r). [...] :- assign(6,r),assign(2,r).
:- assign(1,b),assign(2,b).  :- assign(2,b),assign(4,b).       :- assign(6,b),assign(2,b).
:- assign(1,g),assign(2,g).  :- assign(2,g),assign(4,g).       :- assign(6,g),assign(2,g).
:- assign(1,r),assign(3,r).  :- assign(2,r),assign(5,r).       :- assign(6,r),assign(3,r).
:- assign(1,b),assign(3,b).  :- assign(2,b),assign(5,b).       :- assign(6,b),assign(3,b).
:- assign(1,g),assign(3,g).  :- assign(2,g),assign(5,g).       :- assign(6,g),assign(3,g).
:- assign(1,r),assign(4,r).  :- assign(2,r),assign(6,r).       :- assign(6,r),assign(5,r).
:- assign(1,b),assign(4,b).  :- assign(2,b),assign(6,b).       :- assign(6,b),assign(5,b).
:- assign(1,g),assign(4,g).  :- assign(2,g),assign(6,g).       :- assign(6,g),assign(5,g).
```

Graph coloring: Grounding

```
$ clingo --text graph.lp color.lp

node(1).  node(2).  node(3).  node(4).  node(5).  node(6).

edge(1,2).  edge(2,4).  edge(3,1).  edge(4,1).  edge(5,3).  edge(6,2).
edge(1,3).  edge(2,5).  edge(3,4).  edge(4,2).  edge(5,4).  edge(6,3).
edge(1,4).  edge(2,6).  edge(3,5).              edge(5,6).  edge(6,5).

color(r).  color(b).  color(g).

{assign(1,r);assign(1,b);assign(1,g)} = 1. {assign(4,r);assign(4,b);assign(4,g)} = 1.
{assign(2,r);assign(2,b);assign(2,g)} = 1. {assign(5,r);assign(5,b);assign(5,g)} = 1.
{assign(3,r);assign(3,b);assign(3,g)} = 1. {assign(6,r);assign(6,b);assign(6,g)} = 1.

:- assign(1,r),assign(2,r).  :- assign(2,r),assign(4,r). [...] :- assign(6,r),assign(2,r).
:- assign(1,b),assign(2,b).  :- assign(2,b),assign(4,b).       :- assign(6,b),assign(2,b).
:- assign(1,g),assign(2,g).  :- assign(2,g),assign(4,g).       :- assign(6,g),assign(2,g).
:- assign(1,r),assign(3,r).  :- assign(2,r),assign(5,r).       :- assign(6,r),assign(3,r).
:- assign(1,b),assign(3,b).  :- assign(2,b),assign(5,b).       :- assign(6,b),assign(3,b).
:- assign(1,g),assign(3,g).  :- assign(2,g),assign(5,g).       :- assign(6,g),assign(3,g).
:- assign(1,r),assign(4,r).  :- assign(2,r),assign(6,r).       :- assign(6,r),assign(5,r).
:- assign(1,b),assign(4,b).  :- assign(2,b),assign(6,b).       :- assign(6,b),assign(5,b).
:- assign(1,g),assign(4,g).  :- assign(2,g),assign(6,g).       :- assign(6,g),assign(5,g).
```

ASP workflow: Solving



Graph coloring: Solving

```
$ gringo graph.lp color.lp | clasp 0
```

```
clasp version 2.1.0
Reading from stdin
Solving...
Answer: 1
node(1) [...] assign(6,b) assign(5,g) assign(4,b) assign(3,r) assign(2,r) assign(1,g)
Answer: 2
node(1) [...] assign(6,r) assign(5,g) assign(4,r) assign(3,b) assign(2,b) assign(1,g)
Answer: 3
node(1) [...] assign(6,g) assign(5,b) assign(4,g) assign(3,r) assign(2,r) assign(1,b)
Answer: 4
node(1) [...] assign(6,r) assign(5,b) assign(4,r) assign(3,g) assign(2,g) assign(1,b)
Answer: 5
node(1) [...] assign(6,g) assign(5,r) assign(4,g) assign(3,b) assign(2,b) assign(1,r)
Answer: 6
node(1) [...] assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
SATISFIABLE

Models      : 6
Time        : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
```

Graph coloring: Solving

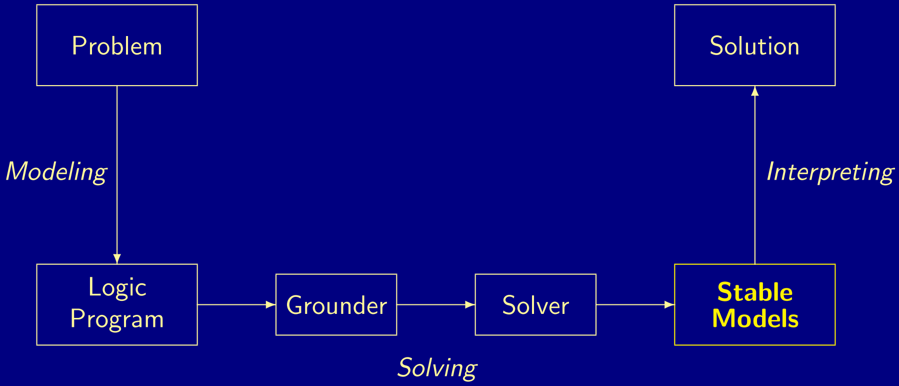
```
$ gringo graph.lp color.lp | clasp 0

clasp version 2.1.0
Reading from stdin
Solving...
Answer: 1
node(1) [...] assign(6,b) assign(5,g) assign(4,b) assign(3,r) assign(2,r) assign(1,g)
Answer: 2
node(1) [...] assign(6,r) assign(5,g) assign(4,r) assign(3,b) assign(2,b) assign(1,g)
Answer: 3
node(1) [...] assign(6,g) assign(5,b) assign(4,g) assign(3,r) assign(2,r) assign(1,b)
Answer: 4
node(1) [...] assign(6,r) assign(5,b) assign(4,r) assign(3,g) assign(2,g) assign(1,b)
Answer: 5
node(1) [...] assign(6,g) assign(5,r) assign(4,g) assign(3,b) assign(2,b) assign(1,r)
Answer: 6
node(1) [...] assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
SATISFIABLE

Models       : 6
Time         : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
```

Graph coloring: Solving

```
$ clingo graph.lp color.lp 0

clasp version 2.1.0
Reading from stdin
Solving...
Answer: 1
node(1) [...] assign(6,b) assign(5,g) assign(4,b) assign(3,r) assign(2,r) assign(1,g)
Answer: 2
node(1) [...] assign(6,r) assign(5,g) assign(4,r) assign(3,b) assign(2,b) assign(1,g)
Answer: 3
node(1) [...] assign(6,g) assign(5,b) assign(4,g) assign(3,r) assign(2,r) assign(1,b)
Answer: 4
node(1) [...] assign(6,r) assign(5,b) assign(4,r) assign(3,g) assign(2,g) assign(1,b)
Answer: 5
node(1) [...] assign(6,g) assign(5,r) assign(4,g) assign(3,b) assign(2,b) assign(1,r)
Answer: 6
node(1) [...] assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
SATISFIABLE

Models      : 6
Time        : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
```

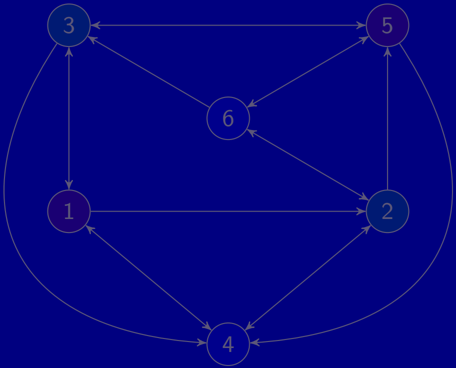
Potassco

ASP workflow: Stable models



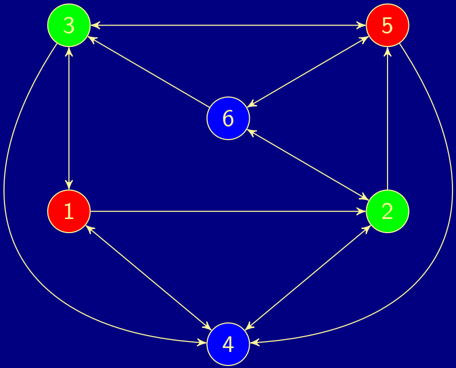
A coloring

```
Answer: 6
node(1)   [...]    \
assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
```

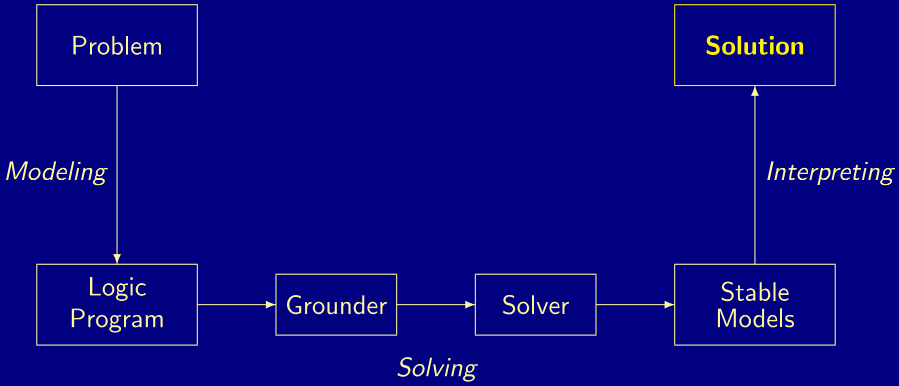


A coloring

```
Answer: 6
node(1)   [...]   \
assign(6,b) assign(5,r) assign(4,b) assign(3,g) assign(2,g) assign(1,r)
```



ASP workflow: Solutions



Outline

Basic methodology

Methodology

Generate and Test (or: Guess and Check)

Generator Generate potential stable model candidates
(typically through non-deterministic constructs)

Tester Eliminate invalid candidates
(typically through integrity constraints)

Nutshell

Logic program = Data + Generator + Tester (+ Optimizer)

Basic methodology

Methodology

Generate and Test (or: Guess and Check)

- **Generator** Generate potential stable model candidates
  (typically through non-deterministic constructs)
- **Tester** Eliminate invalid candidates
  (typically through integrity constraints)

Nutshell

Logic program = Data + Generator + Tester (+ Optimizer)

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

$\left.\right\}$ **Problem instance**

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

$\left.\right\}$ **Problem encoding**

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Data

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Problem encoding

Potassco

Graph coloring

```
node(1..6).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).
edge(3,1).  edge(3,4).  edge(3,5).
edge(4,1).  edge(4,2).
edge(5,3).  edge(5,4).  edge(5,6).
edge(6,2).  edge(6,3).  edge(6,5).

color(r).   color(b).   color(g).
```

Data

```
{ assign(N,C) : color(C) } = 1 :- node(N).

:- edge(N,M), assign(N,C), assign(M,C).
```

Generator

Tester

Potassco

Outline

Potassco

Outline

Satisfiability testing

- Problem Instance A propositional formula ϕ in CNF
- Problem Class Is there an assignment of propositional variables to true and false such that a given formula ϕ is true
- Example Consider formula

$$(a \vee \neg b) \wedge (\neg a \vee b)$$

- Logic Program

Generator	Tester	Stable models
`{a}.`	`:- not a, b.`	$X_1 = \{a, b\}$
`{b}.`	`:- a, not b.`	$X_2 = \{\}$

- Note The generator puts `a` and `b` under the open world assumption
  The tester eliminates interpretations; it is expressed negatively

Potassco

Satisfiability testing

- Problem Instance A propositional formula ϕ in CNF
- Problem Class Is there an assignment of propositional variables to true and false such that a given formula ϕ is true
- Example Consider formula

$$(a \vee \neg b) \wedge (\neg a \vee b)$$

- Logic Program

Generator	Tester	Stable models
`{a}.`	`:- not a, b.`	$X_1 = \{a, b\}$
`{b}.`	`:- a, not b.`	$X_2 = \{\}$

- Note The generator puts `a` and `b` under the open world assumption
  The tester eliminates interpretations; it is expressed negatively

Potassco

Satisfiability testing

- Problem Instance A propositional formula ϕ in CNF
- Problem Class Is there an assignment of propositional variables to true and false such that a given formula ϕ is true
- Example Consider formula

$$(a \vee \neg b) \wedge (\neg a \vee b)$$

- Logic Program

Generator	**Tester**	**Stable models**
`{a}.`	`:- not a, b.`	$X_1 = \{\mathtt{a}, \mathtt{b}\}$
`{b}.`	`:- a, not b.`	$X_2 = \{\}$

- Note The generator puts a and b under the open world assumption
  The tester eliminates interpretations; it is expressed negatively

Satisfiability testing

- Problem Instance A propositional formula ϕ in CNF
- Problem Class Is there an assignment of propositional variables to true and false such that a given formula ϕ is true
- Example Consider formula

$$(a \vee \neg b) \wedge (\neg a \vee b)$$

- Logic Program

Generator	**Tester**	**Stable models**
`{a}.`	`:- not a, b.`	$X_1 = \{\mathtt{a}, \mathtt{b}\}$
`{b}.`	`:- a, not b.`	$X_2 = \{\}$

- Note The generator puts `a` and `b` under the open world assumption
  The tester eliminates interpretations; it is expressed negatively

Satisfiability testing

- Problem Instance A propositional formula ϕ in CNF
- Problem Class Is there an assignment of propositional variables to true and false such that a given formula ϕ is true
- Example Consider formula

$$\neg(\neg a \wedge b) \wedge \neg(a \wedge \neg b)$$

- Logic Program

Generator	**Tester**	**Stable models**
`{a}.`	`:- not a, b.`	$X_1 = \{\mathtt{a}, \mathtt{b}\}$
`{b}.`	`:- a, not b.`	$X_2 = \{\}$

- Note The generator puts `a` and `b` under the open world assumption
  The tester eliminates interpretations; it is expressed negatively

Satisfiability testing

- Problem Instance A propositional formula ϕ in CNF
- Problem Class Is there an assignment of propositional variables to true and false such that a given formula ϕ is true
- Example Consider formula

$$(\neg a \wedge b \rightarrow \bot) \wedge (a \wedge \neg b \rightarrow \bot)$$

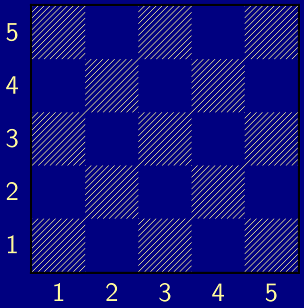
- Logic Program

Generator	**Tester**	**Stable models**
`{a}.`	`:- not a, b.`	$X_1 = \{\mathtt{a}, \mathtt{b}\}$
`{b}.`	`:- a, not b.`	$X_2 = \{\}$

- Note The generator puts `a` and `b` under the open world assumption
  The tester eliminates interpretations; it is expressed negatively

Outline

The n-queens problem

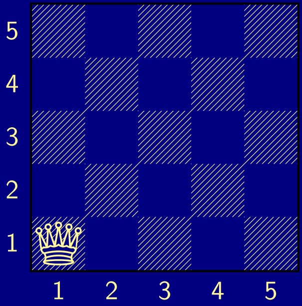


- Place n queens on an $n \times n$ chess board
- Queens must not attack one another

- Example $n = 5$

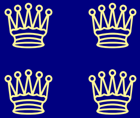
♕ ♕ ♕
♕ ♕

The n-queens problem

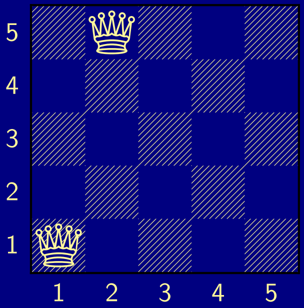


- Place n queens on an $n \times n$ chess board
- Queens must not attack one another

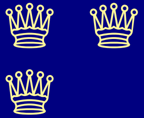
- Example $n = 5$



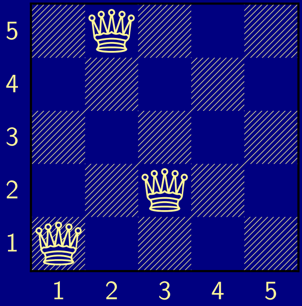
The n-queens problem



- Place n queens on an $n \times n$ chess board
- Queens must not attack one another
- Example $n = 5$

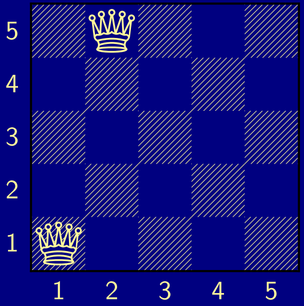


The n-queens problem

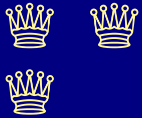


- Place n queens on an $n \times n$ chess board
- Queens must not attack one another
- Example $n = 5$

The n-queens problem

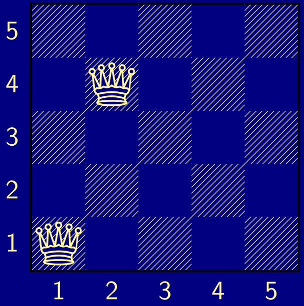


- Place n queens on an $n \times n$ chess board
- Queens must not attack one another
- Example $n = 5$

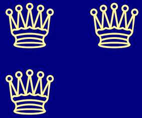


Potassco

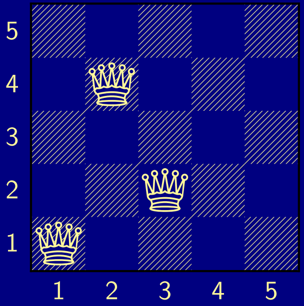
The n-queens problem



- Place n queens on an $n \times n$ chess board
- Queens must not attack one another
- Example $n = 5$

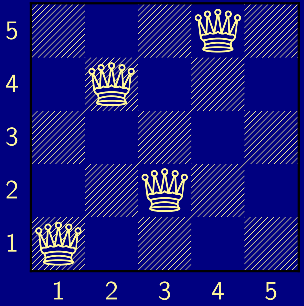


The n-queens problem



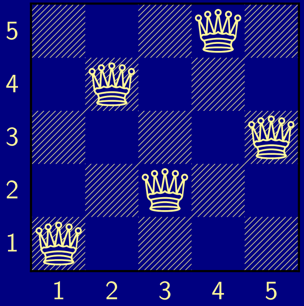
- Place n queens on an $n \times n$ chess board
- Queens must not attack one another
- Example $n = 5$

The n-queens problem



- Place n queens on an $n \times n$ chess board
- Queens must not attack one another
- Example $n = 5$

The n-queens problem



- Place n queens on an $n \times n$ chess board
- Queens must not attack one another

- Example $n = 5$

Defining the field

queens.lp

```
row(1..n).
col(1..n).
```

➥ Define the field
- n rows
- n columns

Defining the field

queens.lp

```
row(1..n).
col(1..n).
```

➥ Define the field

- n rows
- n columns

Defining the field

Running ...

```
$ clingo queens.lp --const n=5
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5)
SATISFIABLE

Models      : 1
Time        : 0.000
```

Potassco

Placing some queens

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
```

➥ Guess a solution candidate
by placing some queens on the board

Placing some queens

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
```

➥ Guess a solution candidate
by placing some queens on the board

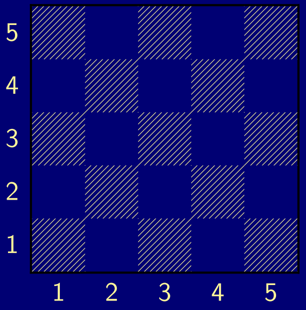
Placing some queens

Running ...

```
$ clingo queens.lp --const n=5 3
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5)
Answer: 2
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) queen(1,1)
Answer: 3
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) queen(2,1)
SATISFIABLE

Models      : 3+
```

Placing some queens

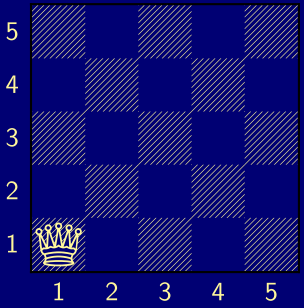
Answer: 1




```
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5)
```

Placing some queens

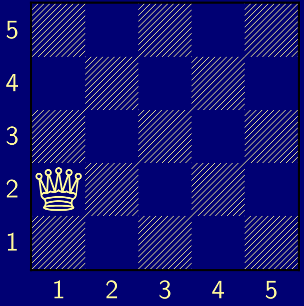
Answer: 2




```
Answer: 2
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(1,1)
```

Placing some queens

Answer: 3




```
Answer: 3
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(2,1)
```

Placing n queens

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
```

➥ Place exactly n queens on the board

Potassco

Placing n queens

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
```

➥ Place exactly n queens on the board

Placing n queens directly

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) } = n.
```

➥ Place exactly n queens on the board

Potassco

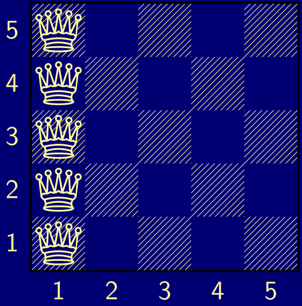
Placing n queens

Running ...

```
$ clingo queens.lp --const n=5 2
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(5,1) queen(4,1) queen(3,1) queen(2,1) queen(1,1)
Answer: 2
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(1,2) queen(4,1) queen(3,1) queen(2,1) queen(1,1)
```

Placing n queens

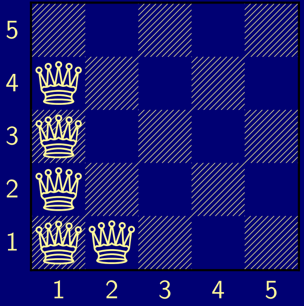
Answer: 1




```
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(5,1) queen(4,1) queen(3,1) \
queen(2,1) queen(1,1)
```

Placing *n* queens

Answer: 2




```
Answer: 2
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(1,2) queen(4,1) queen(3,1) \
queen(2,1) queen(1,1)
```

Horizontal and vertical attack

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
```

➥ Forbid horizontal and vertical attacks

Horizontal and vertical attack

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
```

➥ Forbid horizontal and vertical attacks

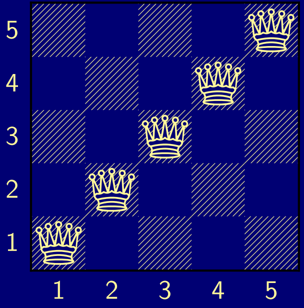
Horizontal and vertical attack

Running ...

```
$ clingo queens.lp --const n=5
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(5,5) queen(4,4) queen(3,3) queen(2,2) queen(1,1)
```

Horizontal and vertical attack

Answer: 1



```
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(5,5) queen(4,4) queen(3,3) \
queen(2,2) queen(1,1)
```

Diagonal attack

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I-J == I'-J'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I+J == I'+J'.
```

➥ Forbid diagonal attacks

Diagonal attack

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I-J == I'-J'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I+J == I'+J'.
```

➥ Forbid diagonal attacks

Diagonal attack

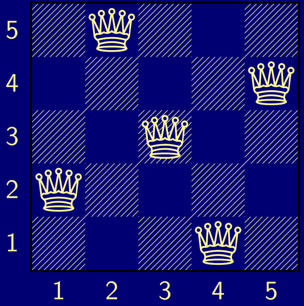
Running ...

```
$ clingo queens.lp --const n=5
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(4,5) queen(1,4) queen(3,3) queen(5,2) queen(2,1)
SATISFIABLE

Models       : 1+
Time         : 0.000
```

Potassco

Diagonal attack

Answer: 1




```
Answer: 1
row(1) row(2) row(3) row(4) row(5) \
col(1) col(2) col(3) col(4) col(5) \
queen(4,5) queen(1,4) queen(3,3) \
queen(5,2) queen(2,1)
```

Optimizing

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I-J == I'-J'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I+J == I'+J'.
```

- Encoding can be optimized
- Much faster to solve

Optimizing

queens.lp

```
row(1..n).
col(1..n).
{ queen(I,J) : row(I), col(J) }.
:- { queen(I,J) } != n.
:- queen(I,J), queen(I,J'), J != J'.
:- queen(I,J), queen(I',J), I != I'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I-J == I'-J'.
:- queen(I,J), queen(I',J'), (I,J) != (I',J'), I+J == I'+J'.
```

- Encoding can be optimized
- Much faster to solve

Optimizing

queens-opt.lp

```
{ queen(I,1..n) } = 1 :- I = 1..n.
{ queen(1..n,J) } = 1 :- J = 1..n.
 :- { queen(D-J,J) } > 1, D =   2..2*n.
 :- { queen(D+J,J) } > 1, D = 1-n..n-1.
```

- Encoding can be optimized
- Much faster to solve

And sometimes it rocks

```
$ clingo -c n=5000 queens-opt-diag.lp --config=jumpy -q --stats=2
```

```
clingo version 4.1.0
Solving...
SATISFIABLE

Models       : 1+
Time         : 3758.143s (Solving: 1905.22s 1st Model: 1896.20s Unsat: 0.00s)
CPU Time     : 3758.320s

Choices      : 288594554
Conflicts    : 3442   (Analyzed: 3442)
Restarts     : 17     (Average: 202.47 Last: 3442)
Model-Level  : 7594728.0
Problems     : 1      (Average Length: 0.00 Splits: 0)
Lemmas       : 3442   (Deleted: 0)
  Binary     : 0      (Ratio:   0.00%)
  Ternary    : 0      (Ratio:   0.00%)
  Conflict   : 3442   (Average Length: 229056.5 Ratio: 100.00%)
  Loop       : 0      (Average Length:    0.0 Ratio:   0.00%)
  Other      : 0      (Average Length:    0.0 Ratio:   0.00%)

Atoms        : 75084857 (Original: 75069989 Auxiliary: 14868)
Rules        : 100129956 (1: 50059992/100090100 2: 39990/29856 3: 10000/10000)
Bodies       : 25090103
Equivalences : 125029999 (Atom=Atom: 50009999 Body=Body: 0 Other: 75020000)
Tight        : Yes
Variables    : 25024868 (Eliminated: 11781 Frozen: 25000000)
Constraints  : 66664  (Binary: 35.6% Ternary:  0.0% Other: 64.4%)

Backjumps    : 3442   (Average: 681.19 Max: 169512 Sum: 2344658)
  Executed   : 3442   (Average: 681.19 Max: 169512 Sum: 2344658 Ratio: 100.00%)
  Bounded    : 0      (Average:  0.00 Max:    0 Sum:       0 Ratio:   0.00%)
```

And sometimes it rocks

```
$ clingo -c n=5000 queens-opt-diag.lp --config=jumpy -q --stats=2
```

```
clingo version 4.1.0
Solving...
SATISFIABLE

Models       : 1+
Time         : 3758.143s (Solving: 1905.22s 1st Model: 1896.20s Unsat: 0.00s)
CPU Time     : 3758.320s

Choices      : 288594554
Conflicts    : 3442   (Analyzed: 3442)
Restarts     : 17     (Average: 202.47 Last: 3442)
Model-Level  : 7594728.0
Problems     : 1      (Average Length: 0.00 Splits: 0)
Lemmas       : 3442   (Deleted: 0)
  Binary     : 0      (Ratio:   0.00%)
  Ternary    : 0      (Ratio:   0.00%)
  Conflict   : 3442   (Average Length: 229056.5 Ratio: 100.00%)
  Loop       : 0      (Average Length:    0.0 Ratio:   0.00%)
  Other      : 0      (Average Length:    0.0 Ratio:   0.00%)

Atoms        : 75084857 (Original: 75069989 Auxiliary: 14868)
Rules        : 100129956 (1: 50059992/100090100 2: 39990/29856 3: 10000/10000)
Bodies       : 25090103
Equivalences : 125029999 (Atom=Atom: 50009999 Body=Body: 0 Other: 75020000)
Tight        : Yes
Variables    : 25024868 (Eliminated: 11781 Frozen: 25000000)
Constraints  : 66664  (Binary: 35.6% Ternary:  0.0% Other: 64.4%)

Backjumps    : 3442   (Average: 681.19 Max: 169512 Sum: 2344658)
  Executed   : 3442   (Average: 681.19 Max: 169512 Sum: 2344658 Ratio: 100.00%)
  Bounded    : 0      (Average:   0.00 Max:      0 Sum:       0 Ratio:   0.00%)
```

Outline

The traveling salesperson problem (TSP)

- Problem Instance A set of cities and distances among them, or simply a weighted graph
- Problem Class What is the shortest possible route visiting each city once and returning to the city of origin?
- Note
  - TSP extends the Hamiltonian cycle problem:
    Is there a cycle in a graph visiting each node exactly once
  - TSP is relevant to applications in logistics, planning, chip design, and the core of the vehicle routing problem

Potassco

The traveling salesperson problem (TSP)

- Problem Instance A set of cities and distances among them, or simply a weighted graph
- Problem Class What is the shortest possible route visiting each city once and returning to the city of origin?

- Note
  - TSP extends the Hamiltonian cycle problem:
    Is there a cycle in a graph visiting each node exactly once
  - TSP is relevant to applications in logistics, planning, chip design, and the core of the vehicle routing problem

Traveling salesperson

Problem instance, `cities.lp`

```
start(a).

city(a). city(b). city(c). city(d).

road(a,b,10). road(b,c,20). road(c,d,25). road(d,a,40).
road(b,d,30). road(d,c,25). road(c,a,35).
```

Traveling salesperson

Problem encoding, tsp.lp

```
{ travel(X,Y) } :- road(X,Y,_).

visited(Y) :- travel(X,Y), start(X).
visited(Y) :- travel(X,Y), visited(X).

:- city(X), not visited(X).

:- city(X), 2 { travel(X,Y) }.
:- city(X), 2 { travel(Y,X) }.
```

Traveling salesperson

Problem encoding, `tsp.lp`

```
{ travel(X,Y) } :- road(X,Y,_).

visited(Y) :- travel(X,Y), start(X).
visited(Y) :- travel(X,Y), visited(X).

:- city(X), not visited(X).

:- city(X), 2 { travel(X,Y) }.
:- city(X), 2 { travel(Y,X) }.

:~ travel(X,Y), road(X,Y,D). [D,X,Y]
```

Traveling salesperson

Problem encoding, tsp.lp

```
{ travel(X,Y) } :- road(X,Y,_).

visited(Y) :- travel(X,Y), start(X).
visited(Y) :- travel(X,Y), visited(X).

:- city(X), not visited(X).

:- city(X), 2 { travel(X,Y) }.
:- city(X), 2 { travel(Y,X) }.

#minimize { D,X,Y : travel(X,Y), road(X,Y,D) }.
```

Running salesperson

```
$ clingo tsp.lp cities.lp
clingo version 5.3.1
Reading...
Solving...
Answer: 1
start(a) [...] road(c,a,35)
travel(a,b) travel(b,d) travel(d,c) travel(c,a)
visited(b) visited(c) visited(d) visited(a)
Optimization: 100
Answer: 2
start(a) [...] road(c,a,35)
travel(a,b) travel(b,c) travel(c,d) travel(d,a)
visited(b) visited(c) visited(d) visited(a)
Optimization: 95
OPTIMUM FOUND

Models       : 2
  Optimum    : yes
Optimization : 95
Calls        : 1
Time         : 0.005s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.002s
```

Potassco

Running salesperson

```
$ clingo tsp.lp cities.lp
clingo version 5.3.1
Reading...
Solving...
Answer: 1
start(a) [...] road(c,a,35)
travel(a,b) travel(b,d) travel(d,c) travel(c,a)
visited(b) visited(c) visited(d) visited(a)
Optimization: 100
Answer: 2
start(a) [...] road(c,a,35)
travel(a,b) travel(b,c) travel(c,d) travel(d,a)
visited(b) visited(c) visited(d) visited(a)
Optimization: 95
OPTIMUM FOUND

Models       : 2
  Optimum    : yes
Optimization : 95
Calls        : 1
Time         : 0.005s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.002s
```

Running salesperson

```
$ clingo tsp.lp cities.lp
clingo version 5.3.1
Reading...
Solving...
Answer: 1
start(a) [...] road(c,a,35)
travel(a,b) travel(b,d) travel(d,c) travel(c,a)
visited(b) visited(c) visited(d) visited(a)
Optimization: 100
Answer: 2
start(a) [...] road(c,a,35)
travel(a,b) travel(b,c) travel(c,d) travel(d,a)
visited(b) visited(c) visited(d) visited(a)
Optimization: 95
OPTIMUM FOUND

Models       : 2
  Optimum    : yes
Optimization : 95
Calls        : 1
Time         : 0.005s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.002s
```

Potassco

Running salesperson

```
$ clingo tsp.lp cities.lp
clingo version 5.3.1
Reading...
Solving...
Answer: 1
start(a) [...] road(c,a,35)
travel(a,b) travel(b,d) travel(d,c) travel(c,a)
visited(b) visited(c) visited(d) visited(a)
Optimization: 100
Answer: 2
start(a) [...] road(c,a,35)
travel(a,b) travel(b,c) travel(c,d) travel(d,a)
visited(b) visited(c) visited(d) visited(a)
Optimization: 95
OPTIMUM FOUND

Models       : 2
  Optimum    : yes
Optimization : 95
Calls        : 1
Time         : 0.005s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.002s
```

Running salesperson

```
$ clingo tsp.lp cities.lp
clingo version 5.3.1
Reading...
Solving...
Answer: 1
start(a) [...] road(c,a,35)
travel(a,b) travel(b,d) travel(d,c) travel(c,a)
visited(b) visited(c) visited(d) visited(a)
Optimization: 100
Answer: 2
start(a) [...] road(c,a,35)
travel(a,b) travel(b,c) travel(c,d) travel(d,a)
visited(b) visited(c) visited(d) visited(a)
Optimization: 95
OPTIMUM FOUND

Models       : 2
  Optimum    : yes
Optimization : 95
Calls        : 1
Time         : 0.005s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.002s
```

Traveling salesperson

Alternative problem encoding

```
{ travel(X,Y) : road(X,Y,_) } = 1 :- city(X).
{ travel(X,Y) : road(X,Y,_) } = 1 :- city(Y).

visited(Y) :- travel(X,Y), start(X).
visited(Y) :- travel(X,Y), visited(X).

:- city(X), not visited(X).

#minimize { D,X,Y : travel(X,Y), road(X,Y,D) }.
```

Potassco

Outline

Reviewer Assignment

- Problem Instance A set of papers and a set of reviewers along with their first and second choices of papers and conflict of interests
- Problem Class A nice assignment of three reviewers to each paper

Reviewer Assignment

- Problem Instance A set of papers and a set of reviewers along with their first and second choices of papers and conflict of interests
- Problem Class A "nice" assignment of three reviewers to each paper

Reviewer Assignment

by Ilkka Niemelä

```
paper(p1).  reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2).  reviewer(r2). classA(r2,p3). classB(r2,p4). coi(r2,p6).
[...]

{ assigned(P,R) : reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Potassco

Reviewer Assignment

by Ilkka Niemelä

```
paper(p1).  reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2).  reviewer(r2). classA(r2,p3). classB(r2,p4). coi(r2,p6).
[...]

{ assigned(P,R) : reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Potassco

Reviewer Assignment

by Ilkka Niemelä

```
paper(p1).  reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2).  reviewer(r2). classA(r2,p3). classB(r2,p4). coi(r2,p6).
[...]

{ assigned(P,R) : reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Reviewer Assignment

by Ilkka Niemelä

```
paper(p1).  reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2).  reviewer(r2). classA(r2,p3). classB(r2,p4). coi(r2,p6).
[...]

{ assigned(P,R) : reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Potassco

Reviewer Assignment

by Ilkka Niemelä

```
paper(p1).  reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2).  reviewer(r2). classA(r2,p3). classB(r2,p4). coi(r2,p6).
[...]

{ assigned(P,R) : reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 { assigned(P,R) : paper(P) } 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 { assignedB(P,R) : paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Potassco

Reviewer Assignment

by Ilkka Niemelä

```
paper(p1).  reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2).  reviewer(r2). classA(r2,p3). classB(r2,p4). coi(r2,p6).
[...]

#count { P,R : assigned(P,R) , reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 <= #count { P,R : assigned(P,R), paper(P) } <= 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 <= #count { P,R : assignedB(P,R), paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Reviewer Assignment

by Ilkka Niemelä

```
paper(p1).  reviewer(r1). classA(r1,p1). classB(r1,p2). coi(r1,p3).
paper(p2).  reviewer(r2). classA(r2,p3). classB(r2,p4). coi(r2,p6).
[...]

#count { P,R : assigned(P,R) , reviewer(R) } = 3 :-  paper(P).

 :- assigned(P,R), coi(R,P).
 :- assigned(P,R), not classA(R,P), not classB(R,P).
 :- not 6 <= #count { P,R : assigned(P,R), paper(P) } <= 9, reviewer(R).

assignedB(P,R) :-  classB(R,P), assigned(P,R).
 :- 3 <= #count { P,R : assignedB(P,R), paper(P) }, reviewer(R).

#minimize { 1,P,R : assignedB(P,R), paper(P), reviewer(R) }.
```

Outline

Simplified STRIPS[2] Planning

- Problem Instance
  - set of fluents
  - initial and goal state
  - set of actions, consisting of pre- and postconditions
  - number k of allowed actions
- Problem Class Find a plan, that is, a sequence of k actions leading from the initial state to the goal state
- Example
  - fluents $\{p, q, r\}$
  - initial state $\{p, \neg q, \neg r\}$
  - goal state $\{r\}$
  - actions $a = (\{p\}, \{q, \neg p\})$ and $b = (\{q\}, \{r, \neg q\})$
  - length 2

  plan $\langle a, b \rangle$ $\quad \{p, \neg q, \neg r\} \xrightarrow{a} \{\neg p, q, \neg r\} \xrightarrow{b} \{\neg p, \neg q, r\}$

[2] Stanford Research Institute Problem Solver, 1971

Potassco

Simplified STRIPS Planning

- Problem Instance
  - set of fluents
  - initial and goal state
  - set of actions, consisting of pre- and postconditions
  - number k of allowed actions
- Problem Class Find a plan, that is, a sequence of k actions leading from the initial state to the goal state
- Example
  - fluents $\{p, q, r\}$
  - initial state $\{p, \neg q, \neg r\}$
  - goal state $\{r\}$
  - actions $a = (\{p\}, \{q, \neg p\})$ and $b = (\{q\}, \{r, \neg q\})$
  - length 2

  plan $\langle a, b \rangle$ $\{p, \neg q, \neg r\} \xrightarrow{a} \{\neg p, q, \neg r\} \xrightarrow{b} \{\neg p, \neg q, r\}$

Simplified STRIPS Planning

- Problem Instance
  - set of fluents
  - initial and goal state
  - set of actions, consisting of pre- and postconditions
  - number k of allowed actions
- Problem Class Find a plan, that is, a sequence of k actions leading from the initial state to the goal state
- Example
  - fluents $\{p, q, r\}$
  - initial state $\{p, \neg q, \neg r\}$
  - goal state $\{r\}$
  - actions $a = (\{p\}, \{q, \neg p\})$ and $b = (\{q\}, \{r, \neg q\})$
  - length 2
  - plan $\langle a, b \rangle$ $\quad \{p, \neg q, \neg r\} \xrightarrow{a} \{\neg p, q, \neg r\} \xrightarrow{b} \{\neg p, \neg q, r\}$

Potassco

Simplified STRIPS Planning

- Problem Instance
  - set of fluents
  - initial and goal state
  - set of actions, consisting of pre- and postconditions
  - number k of allowed actions
- Problem Class Find a plan, that is, a sequence of k actions leading from the initial state to the goal state
- Example
  - fluents $\{p, q, r\}$
  - initial state $\{p, \neg q, \neg r\}$
  - goal state $\{r\}$
  - actions $a = (\{p\}, \{q, \neg p\})$ and $b = (\{q\}, \{r, \neg q\})$
  - length 2
  - plan $\langle a, b \rangle$ $\{p, \neg q, \neg r\} \xrightarrow{a} \{\neg p, q, \neg r\} \xrightarrow{b} \{\neg p, \neg q, r\}$

Simplistic STRIPS Planning

Problem instance

```
time(1..k).

fluent(p).   action(a).       action(b).       init(p).
fluent(q).      pre(a,p).        pre(b,q).
fluent(r).      add(a,q).        add(b,r).     query(r).
                del(a,p).        del(b,q).
```

Simplistic STRIPS Planning

Problem encoding

```
holds(P,0) :- init(P).

{ occ(A,T) : action(A) } = 1 :- time(T).
:- occ(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- occ(A,T), add(A,F).
holds(F,T) :- holds(F,T-1), time(T), not occ(A,T) : del(A,F).

:- query(F), not holds(F,k).
```

Simplistic STRIPS Planning

Solving

```
$ clingo planning-encoding.lp planning-instance.lp -c k=2 0
clingo version 5.5.0
Reading from planning-encoding.lp ...
Solving...
Answer: 1
[...] occ(a,1) occ(b,2)
SATISFIABLE

Models       : 1
Time         : 0.001s (Solving: 0.00s)
CPU Time     : 0.001s
```

Potassco

Simplistic STRIPS Planning

Solving

```
$ clingo planning-encoding.lp planning-instance.lp -c k=2 0
clingo version 5.5.0
Reading from planning-encoding.lp ...
Solving...
Answer: 1
[...] occ(a,1) occ(b,2)
SATISFIABLE

Models       : 1
Time         : 0.001s (Solving: 0.00s)
CPU Time     : 0.001s
```

Outline

Potassco

Things to remember

- Elaboration tolerance, uniform problem representation, problem instance, problem encoding
- Generate and test methodology
- ASP's workflow, modeling, grounding, solving (and optimizing)
- *clingo* = *gringo*+*clasp*+...

Potassco

Things to remember

- Elaboration tolerance, uniform problem representation, problem instance, problem encoding
- Generate and test methodology
- ASP's workflow, modeling, grounding, solving (and optimizing)
- *clingo* = *gringo*+*clasp*+...

Language: Overview

Outline

Outline

Basic language extensions

- Fact The expressiveness of a language can be enhanced by adding interesting language constructs
- Questions
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?
- Answer
  - A way of providing semantics is to furnish a translation removing the new constructs
  - This translation might also be used for implementing the language extension

Potassco

Basic language extensions

- Fact The expressiveness of a language can be enhanced by adding interesting language constructs
- Questions
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?
- Answer
  - A way of providing semantics is to furnish a translation removing the new constructs
  - This translation might also be used for implementing the language extension

Potassco

Basic language extensions

- Fact The expressiveness of a language can be enhanced by adding interesting language constructs
- Questions
  - What is the syntax of the new language construct?
  - What is the semantics of the new language construct?
  - How to implement the new language construct?
- Answer
  - A way of providing semantics is to furnish a translation removing the new constructs
  - This translation might also be used for implementing the language extension

Outline

Integrity constraint

- Purpose Eliminate unwanted solution candidates
- Syntax An integrity constraint is of the form

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

where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$

Integrity constraint

- Purpose Eliminate unwanted solution candidates
- Syntax An integrity constraint is of the form

$$\leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$

- Example

```
:- edge(3,7), color(3,red), color(7,red).
```

Integrity constraint

- Purpose Eliminate unwanted solution candidates
- Syntax An integrity constraint is of the form

$$\leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$

- Example programs

$$\{ a \leftarrow \neg b, \; b \leftarrow \neg a \} \qquad \{a\} \quad \{b\}$$

$$\{ a \leftarrow \neg b, \; b \leftarrow \neg a \} \cup \{ \leftarrow a \}$$

$$\{ a \leftarrow \neg b, \; b \leftarrow \neg a \} \cup \{ \leftarrow \neg a \}$$

Integrity constraint

- Purpose Eliminate unwanted solution candidates
- Syntax An integrity constraint is of the form

$$\leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$

- Example programs

$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \}$ $\{a\}$ $\{b\}$

$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \} \cup \{ \leftarrow a \}$

$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \} \cup \{ \leftarrow \neg a \}$

Integrity constraint

- Purpose Eliminate unwanted solution candidates
- Syntax An integrity constraint is of the form

$$\leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

  where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$

- Example programs

$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \}$	$\{a\}$ $\{b\}$
$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \} \cup \{ \leftarrow a \}$	$\{b\}$
$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \} \cup \{ \leftarrow \neg a \}$	

Potassco

Integrity constraint

- Purpose Eliminate unwanted solution candidates
- Syntax An integrity constraint is of the form

$$\leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

  where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$

- Example programs

$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \}$	$\{a\}$	$\{b\}$
$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \} \cup \{ \leftarrow a \}$		$\{b\}$
$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \} \cup \{ \leftarrow \neg a \}$	$\{a\}$	

Potassco

Embedding in normal rules

- Translation An integrity constraint of form

$$\leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

can be translated into the normal rule

$$x \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n, \neg x$$

where x is a new symbol

- Example programs

$$\{\, a \leftarrow \neg b,\ b \leftarrow \neg a \,\} \qquad \{a\} \quad \{b\}$$

$$\{\, a \leftarrow \neg b,\ b \leftarrow \neg a \,\} \cup \{\, \leftarrow a \,\}$$

$$\{\, a \leftarrow \neg b,\ b \leftarrow \neg a \,\} \cup \{\, \leftarrow \neg a \,\}$$

Potassco

Embedding in normal rules

- Translation An integrity constraint of form

$$\leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

can be translated into the normal rule

$$x \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n, \neg x$$

where x is a new symbol

- Example programs

$$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \} \qquad \{a\} \quad \{b\}$$

$$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \} \cup \{ \leftarrow a \}$$

$$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \} \cup \{ \leftarrow \neg a \}$$

Embedding in normal rules

- Translation An integrity constraint of form

$$\leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

  can be translated into the normal rule

$$x \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n, \neg x$$

  where x is a new symbol

- Example programs

$\{\ a \leftarrow \neg b,\ \ b \leftarrow \neg a\ \}$	$\{a\}\quad\{b\}$
$\{\ a \leftarrow \neg b,\ \ b \leftarrow \neg a\ \} \cup \{\ \leftarrow a\ \}$	
$\{\ a \leftarrow \neg b,\ \ b \leftarrow \neg a\ \} \cup \{\ \leftarrow \neg a\ \}$	

Potassco

Embedding in normal rules

- Translation An integrity constraint of form

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

can be translated into the normal rule

$$x \leftarrow a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n, \neg x$$

where x is a new symbol

- Example programs

$$\{ a \leftarrow \neg b, \; b \leftarrow \neg a \} \qquad \{a\} \quad \{b\}$$

$$\{ a \leftarrow \neg b, \; b \leftarrow \neg a \} \cup \{ x \leftarrow a, \neg x \}$$

$$\{ a \leftarrow \neg b, \; b \leftarrow \neg a \} \cup \{ x \leftarrow \neg a, \neg x \}$$

Embedding in normal rules

- Translation An integrity constraint of form

$$\leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n$$

can be translated into the normal rule

$$x \leftarrow a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n, \neg x$$

where x is a new symbol

- Example programs

$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \}$	$\{a\}$	$\{b\}$
$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \} \cup \{ x \leftarrow a, \neg x \}$		$\{b\}$
$\{ a \leftarrow \neg b,\ b \leftarrow \neg a \} \cup \{ x \leftarrow \neg a, \neg x \}$	$\{a\}$	

Potassco

Outline

Choice rule

- Purpose Provide choices over subsets of atoms
- Syntax A choice rule is of the form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o$$

where $0 \leq m \leq n \leq o$ and each a_i is an atom for $1 \leq i \leq o$

- Informal meaning If the body is satisfied by the stable model, any subset of $\{a_1, \dots, a_m\}$ can be included in the stable model

Choice rule

- Purpose Provide choices over subsets of atoms
- Syntax A choice rule is of the form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o$$

  where $0 \leq m \leq n \leq o$ and each a_i is an atom for $1 \leq i \leq o$

- Informal meaning If the body is satisfied by the stable model, any subset of $\{a_1, \dots, a_m\}$ can be included in the stable model

Choice rule

- Purpose Provide choices over subsets of atoms
- Syntax A choice rule is of the form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o$$

  where $0 \leq m \leq n \leq o$ and each a_i is an atom for $1 \leq i \leq o$

- Informal meaning If the body is satisfied by the stable model, any subset of $\{a_1, \dots, a_m\}$ can be included in the stable model
- Example

```
{ buy(pizza);buy(wine);buy(corn) } :- at(grocery).
```

Choice rule

- Purpose Provide choices over subsets of atoms
- Syntax A choice rule is of the form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o$$

where $0 \leq m \leq n \leq o$ and each a_i is an atom for $1 \leq i \leq o$

- Informal meaning If the body is satisfied by the stable model, any subset of $\{a_1, \dots, a_m\}$ can be included in the stable model
- Example program

$$\{\ \{a\} \leftarrow b,\ \ b \leftarrow\ \} \qquad \{b\} \quad \{a, b\}$$

Choice rule

- Purpose Provide choices over subsets of atoms
- Syntax A choice rule is of the form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o$$

  where $0 \leq m \leq n \leq o$ and each a_i is an atom for $1 \leq i \leq o$
- Informal meaning If the body is satisfied by the stable model, any subset of $\{a_1, \dots, a_m\}$ can be included in the stable model
- Example program

$$\{\ \{a\} \leftarrow b,\ \ b \leftarrow\ \} \qquad\qquad \{b\} \quad \{a, b\}$$

Potassco

Embedding in normal rules

- A choice rule of form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o$$

can be translated into $2m + 1$ normal rules

$$\begin{array}{rcl}
x & \leftarrow & a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o \\
a_1 & \leftarrow & x, \neg x_1 \quad \dots \quad a_m \leftarrow x, \neg x_m \\
x_1 & \leftarrow & \neg a_1 \qquad \dots \quad x_m \leftarrow \neg a_m
\end{array}$$

by introducing new atoms $x, x_1, \dots, x_m$

- Example program

Embedding in normal rules

- A choice rule of form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o$$

can be translated into $2m + 1$ normal rules

$$\begin{array}{rclcrcl} x & \leftarrow & a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o & & & & \\ a_1 & \leftarrow & x, \neg x_1 & \dots & a_m & \leftarrow & x, \neg x_m \\ x_1 & \leftarrow & \neg a_1 & \dots & x_m & \leftarrow & \neg a_m \end{array}$$

by introducing new atoms $x, x_1, \dots, x_m$

- Example program

Embedding in normal rules

- A choice rule of form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o$$

can be translated into $2m + 1$ normal rules

$$\begin{array}{rcl} x & \leftarrow & a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o \\ a_1 & \leftarrow & x, \neg x_1 \quad \dots \quad a_m \leftarrow x, \neg x_m \\ x_1 & \leftarrow & \neg a_1 \qquad \dots \quad x_m \leftarrow \neg a_m \end{array}$$

by introducing new atoms $x, x_1, \dots, x_m$

- Example program

Embedding in normal rules

- A choice rule of form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o$$

can be translated into $2m + 1$ normal rules

$$\begin{array}{rclcrcl} x & \leftarrow & a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o & & & & \\ a_1 & \leftarrow & x, \neg x_1 & \dots & a_m & \leftarrow & x, \neg x_m \\ x_1 & \leftarrow & \neg a_1 & \dots & x_m & \leftarrow & \neg a_m \end{array}$$

by introducing new atoms $x, x_1, \dots, x_m$

- Example program

$$\{\ \{a\} \leftarrow b,\ \ b \leftarrow\ \} \qquad\qquad \{b\} \quad \{a, b\}$$

Embedding in normal rules

- A choice rule of form

$$\{a_1, \dots, a_m\} \leftarrow a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o$$

can be translated into $2m + 1$ normal rules

$$\begin{array}{rcl} x & \leftarrow & a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o \\ a_1 & \leftarrow & x, \neg x_1 \quad \dots \quad a_m \leftarrow x, \neg x_m \\ x_1 & \leftarrow & \neg a_1 \quad\quad \dots \quad x_m \leftarrow \neg a_m \end{array}$$

by introducing new atoms $x, x_1, \dots, x_m$

- Example program

$$\left\{ \begin{array}{rcl} x & \leftarrow & b \\ a & \leftarrow & x, \neg x_1 \\ x_1 & \leftarrow & \neg a \end{array} \right\} \cup \{ b \leftarrow \} \qquad \{b, x, x_1\} \quad \{a, b, x\}$$

Embedding in normal rules

- A choice rule of form

$$\{a_1,\dots,a_m\} \leftarrow a_{m+1},\dots,a_n,\neg a_{n+1},\dots,\neg a_o$$

can be translated into $2m+1$ normal rules

$$\begin{array}{rcl} x & \leftarrow & a_{m+1},\dots,a_n,\neg a_{n+1},\dots,\neg a_o \\ a_1 & \leftarrow & x,\neg x_1 \quad \dots \quad a_m \leftarrow x,\neg x_m \\ x_1 & \leftarrow & \neg a_1 \qquad \dots \quad x_m \leftarrow \neg a_m \end{array}$$

by introducing new atoms $x,x_1,\dots,x_m$

- Example program

$$\left\{ \begin{array}{rcl} x & \leftarrow & b \\ a & \leftarrow & x,\neg x_1 \\ x_1 & \leftarrow & \neg a \end{array} \right\} \cup \{\, b \leftarrow \,\} \qquad \{b,x,x_1\} \quad \{a,b,x\}$$

Outline

Cardinality rule

- Purpose Control (lower) cardinality of subsets of literals
- Syntax A cardinality rule is the form

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \; \}$$

where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$ and l is a non-negative integer called lower bound

- Informal meaning The head belongs to the stable model, if at least l positive/negative body literals are in/excluded in the stable model

Cardinality rule

- Purpose Control (lower) cardinality of subsets of literals
- Syntax A cardinality rule is the form

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \; \}$$

  where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$ and l is a non-negative integer called lower bound
- Informal meaning The head belongs to the stable model, if at least l positive/negative body literals are in/excluded in the stable model

Cardinality rule

- Purpose Control (lower) cardinality of subsets of literals
- Syntax A cardinality rule is the form

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \; \}$$

where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$
and l is a non-negative integer called lower bound

- Informal meaning The head belongs to the stable model, if at least l positive/negative body literals are in/excluded in the stable model
- Example

```
pass(c42) :- 2 { pass(a1); pass(a2); pass(a3) }.
```

Potassco

Cardinality rule

- Purpose Control (lower) cardinality of subsets of literals
- Syntax A cardinality rule is the form

$$a_0 \leftarrow l \; \{ \; a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n \; \}$$

  where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$ and l is a non-negative integer called lower bound

- Informal meaning The head belongs to the stable model, if at least l positive/negative body literals are in/excluded in the stable model
- Example program

$$\{ \; a \leftarrow 1 \; \{b, c\}, \;\; b \leftarrow \; \} \qquad\qquad \{a, b\}$$

Potassco

Cardinality rule

- Purpose Control (lower) cardinality of subsets of literals
- Syntax A cardinality rule is the form

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \; \}$$

  where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$ and l is a non-negative integer called lower bound
- Informal meaning The head belongs to the stable model, if at least l positive/negative body literals are in/excluded in the stable model
- Example program

$$\{ \; a \leftarrow 1 \, \{b, c\}, \;\; b \leftarrow \; \} \qquad\qquad \{a, b\}$$

Potassco

Embedding in normal rules

- A cardinality rule of form

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \; \}$$

is translated into the normal rule $a_0 \leftarrow x(1, l)$

Embedding in normal rules

- A cardinality rule of form

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \; \}$$

is translated into the normal rule $a_0 \leftarrow x(1, l)$

Embedding in normal rules

- A cardinality rule of form

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \; \}$$

is translated into the normal rule $a_0 \leftarrow x(1, l)$

- Idea The atom $x(i, j)$ represents that at least j of the literals having an equal or greater index than i are in a stable model

Embedding in normal rules

- A cardinality rule of form

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \; \}$$

is translated into the normal rule $a_0 \leftarrow x(1, l)$ and for $0 \leq k \leq l$ the rules

$$\begin{array}{rcll} x(i, k{+}1) & \leftarrow & x(i+1, k), a_i & \\ x(i, k) & \leftarrow & x(i+1, k) & \text{for } 1 \leq i \leq m \\ x(j, k{+}1) & \leftarrow & x(j+1, k), \neg a_j & \\ x(j, k) & \leftarrow & x(j+1, k) & \text{for } m+1 \leq j \leq n \\ x(n+1, 0) & \leftarrow & & \end{array}$$

- Idea The atom $x(i, j)$ represents that at least j of the literals having an equal or greater index than i are in a stable model

Embedding in normal rules

- A cardinality rule of form

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

is translated into the normal rule $a_0 \leftarrow x(1, l)$ and for $0 \leq k \leq l$ the rules

$$\begin{array}{rcll} x(i, k{+}1) & \leftarrow & x(i+1, k), a_i & \\ x(i, k) & \leftarrow & x(i+1, k) & \text{for } 1 \leq i \leq m \\ x(j, k{+}1) & \leftarrow & x(j+1, k), \neg a_j & \\ x(j, k) & \leftarrow & x(j+1, k) & \text{for } m+1 \leq j \leq n \\ x(n+1, 0) & \leftarrow & & \end{array}$$

- Idea The atom $x(i, j)$ represents that at least j of the literals having an equal or greater index than i are in a stable model

Potassco

Embedding in normal rules

- A cardinality rule of form

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \; \}$$

is translated into the normal rule $a_0 \leftarrow x(1, l)$ and
for $0 \leq k \leq l$ the rules

$$\begin{array}{rcll} x(i, k{+}1) & \leftarrow & x(i+1, k), a_i & \\ x(i, k) & \leftarrow & x(i+1, k) & \text{for } 1 \leq i \leq m \\ x(j, k{+}1) & \leftarrow & x(j+1, k), \neg a_j & \\ x(j, k) & \leftarrow & x(j+1, k) & \text{for } m+1 \leq j \leq n \\ x(n+1, 0) & \leftarrow & & \end{array}$$

- Idea The atom $x(i, j)$ represents that at least j of the literals having an equal or greater index than i are in a stable model

Embedding in normal rules

- A cardinality rule of form

$$a_0 \leftarrow l \; \{ \; a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n \; \}$$

is translated into the normal rule $a_0 \leftarrow x(1, l)$ and
for $0 \leq k \leq l$ the rules

$$\begin{array}{rcll}
x(i, k{+}1) & \leftarrow & x(i+1, k), a_i & \\
x(i, k) & \leftarrow & x(i+1, k) & \text{for } 1 \leq i \leq m \\
x(j, k{+}1) & \leftarrow & x(j+1, k), \neg a_j & \\
x(j, k) & \leftarrow & x(j+1, k) & \text{for } m+1 \leq j \leq n \\
x(n+1, 0) & \leftarrow & &
\end{array}$$

- Idea The atom $x(i, j)$ represents that at least j of the literals having an equal or greater index than i are in a stable model

An example

- Program $\{\ a \leftarrow 1\ \{b, c\},\ \ b \leftarrow\ \}$ has the stable model $\{a, b\}$
- Translating the cardinality rule yields the rules

$$
\begin{array}{rcl}
a & \leftarrow & x(1,1) \\
x(1,2) & \leftarrow & x(2,1), b \\
x(1,1) & \leftarrow & x(2,1) \\
x(2,2) & \leftarrow & x(3,1), c \\
x(2,1) & \leftarrow & x(3,1) \\
x(1,1) & \leftarrow & x(2,0), b \\
x(1,0) & \leftarrow & x(2,0) \\
x(2,1) & \leftarrow & x(3,0), c \\
x(2,0) & \leftarrow & x(3,0) \\
x(3,0) & \leftarrow &
\end{array}
\qquad
\begin{array}{rcl}
b & \leftarrow &
\end{array}
$$

having stable model $\{a, b, x(3,0), x(2,0), x(1,0), x(1,1)\}$

An example

- Program $\{\ a \leftarrow 1\ \{b, c\},\ \ b \leftarrow\ \}$ has the stable model $\{a, b\}$
- Translating the cardinality rule yields the rules

$$
\begin{array}{rclcrcl}
a & \leftarrow & x(1,1) & \qquad & b & \leftarrow & \\
x(1,2) & \leftarrow & x(2,1), b & & & & \\
x(1,1) & \leftarrow & x(2,1) & & & & \\
x(2,2) & \leftarrow & x(3,1), c & & & & \\
x(2,1) & \leftarrow & x(3,1) & & & & \\
x(1,1) & \leftarrow & x(2,0), b & & & & \\
x(1,0) & \leftarrow & x(2,0) & & & & \\
x(2,1) & \leftarrow & x(3,0), c & & & & \\
x(2,0) & \leftarrow & x(3,0) & & & & \\
x(3,0) & \leftarrow & & & & &
\end{array}
$$

having stable model $\{a, b, x(3,0), x(2,0), x(1,0), x(1,1)\}$

. . . and vice versa

- A normal rule

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

can be represented by the cardinality rule

$$a_0 \leftarrow n \; \{a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n\}$$

Cardinality rules with upper bounds

- A rule of the form

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \; \} \; u$$

where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$ and l and u are non-negative integers

Cardinality rules with upper bounds

- A rule of the form

$$a_0 \leftarrow l \; \{ \; a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n \; \} \; u$$

where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$ and l and u are non-negative integers

stands for

$$\begin{array}{rcl} a_0 & \leftarrow & x, \neg y \\ x & \leftarrow & l \; \{ \; a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n \; \} \\ y & \leftarrow & u{+}1 \; \{ \; a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n \; \} \end{array}$$

where x and y are new symbols

Cardinality rules with upper bounds

- A rule of the form

$$a_0 \leftarrow l \; \{ \; a_1, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \; \} \; u$$

  where $0 \leq m \leq n$ and each a_i is an atom for $1 \leq i \leq n$ and l and u are non-negative integers

- Note The expression in the body of the cardinality rule is referred to as a cardinality constraint with lower and upper bound l and u

Cardinality constraints as heads

- A rule of the form

$$l\ \{a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n\}\ u \leftarrow a_{n+1}, \dots, a_o, \neg a_{o+1}, \dots, \neg a_p$$

where $0 \leq m \leq n \leq o \leq p$ and each a_i is an atom for $1 \leq i \leq p$ and l and u are non-negative integers

Potassco

Cardinality constraints as heads

- A rule of the form

$$l\ \{a_1,\dots,a_m,\neg a_{m+1},\dots,\neg a_n\}\ u \leftarrow a_{n+1},\dots,a_o,\neg a_{o+1},\dots,\neg a_p$$

  where $0 \leq m \leq n \leq o \leq p$ and each a_i is an atom for $1 \leq i \leq p$ and l and u are non-negative integers

- Example

```
1 {color(2,red); color(2,green); color(2,blue)} 1.
```

Cardinality constraints as heads

- A rule of the form

$$l \; \{a_1, \dots, a_m, \neg a_{m+1}, \dots, \neg a_n\} \; u \leftarrow a_{n+1}, \dots, a_o, \neg a_{o+1}, \dots, \neg a_p$$

where $0 \leq m \leq n \leq o \leq p$ and each a_i is an atom for $1 \leq i \leq p$ and l and u are non-negative integers

stands for

$$\begin{array}{rcl} x & \leftarrow & a_{n+1}, \dots, a_o, \neg a_{o+1}, \dots, \neg a_p \\ \{a_1, \dots, a_m\} & \leftarrow & x \\ y & \leftarrow & l \; \{a_1, \dots, a_m, , \neg a_{m+1}, \dots, \neg a_n\} \; u \\ & \leftarrow & x, \neg y \end{array}$$

where x and y are new symbols

Full-fledged cardinality rules

- A rule of the form

$$l_0\ S_0\ u_0 \leftarrow l_1\ S_1\ u_1, \ldots, l_n\ S_n\ u_n$$

where each $l_i\ S_i\ u_i$ is a cardinality constraint for $0 \leq i \leq n$

Full-fledged cardinality rules

- A rule of the form

$$l_0 \; S_0 \; u_0 \leftarrow l_1 \; S_1 \; u_1, \ldots, l_n \; S_n \; u_n$$

where each $l_i \; S_i \; u_i$ is a cardinality constraint for $0 \leq i \leq n$ stands for

$$\begin{array}{rcl} x & \leftarrow & y_1, \ldots, y_n, \neg z_1, \ldots, \neg z_n \\ {S_0}^{+} & \leftarrow & x \\ & \leftarrow & x, \neg y_0 \\ & \leftarrow & x, z_0 \end{array} \qquad \begin{array}{rcl} y_i & \leftarrow & l_i \; S_i \\ z_i & \leftarrow & u_i{+}1 \; S_i \end{array}$$

where x, y_i, z_i are new symbols and S_0^{+} gives all atoms in S_0

Full-fledged cardinality rules

- A rule of the form

$$l_0 \; S_0 \; u_0 \leftarrow l_1 \; S_1 \; u_1, \ldots, l_n \; S_n \; u_n$$

where each $l_i \; S_i \; u_i$ is a cardinality constraint for $0 \leq i \leq n$ stands for

$$\begin{array}{rcl} x & \leftarrow & y_1, \ldots, y_n, \neg z_1, \ldots, \neg z_n \\ {S_0}^{+} & \leftarrow & x \\ & \leftarrow & x, \neg y_0 \\ & \leftarrow & x, z_0 \end{array} \qquad \begin{array}{rcl} y_i & \leftarrow & l_i \; S_i \\ z_i & \leftarrow & u_i{+}1 \; S_i \end{array}$$

where x, y_i, z_i are new symbols and S_0^+ gives all atoms in S_0

Full-fledged cardinality rules

- A rule of the form

$$l_0\ S_0\ u_0 \leftarrow l_1\ S_1\ u_1, \ldots, l_n\ S_n\ u_n$$

where each $l_i\ S_i\ u_i$ is a cardinality constraint for $0 \leq i \leq n$ stands for

$$\begin{array}{rcl} x & \leftarrow & y_1, \ldots, y_n, \neg z_1, \ldots, \neg z_n \\[1ex] {S_0}^{+} & \leftarrow & x \\ & \leftarrow & x, \neg y_0 \\ & \leftarrow & x, z_0 \end{array} \qquad \begin{array}{rcl} y_i & \leftarrow & l_i\ S_i \\ z_i & \leftarrow & u_i{+}1\ S_i \end{array}$$

where x, y_i, z_i are new symbols and S_0^+ gives all atoms in S_0

Full-fledged cardinality rules

- A rule of the form

$$l_0 \ S_0 \ u_0 \leftarrow l_1 \ S_1 \ u_1, \dots, l_n \ S_n \ u_n$$

where each $l_i \ S_i \ u_i$ is a cardinality constraint for $0 \leq i \leq n$ stands for

$$\begin{array}{rcl} x & \leftarrow & y_1, \dots, y_n, \neg z_1, \dots, \neg z_n \\ S_0{}^+ & \leftarrow & x \\ & \leftarrow & x, \neg y_0 \\ & \leftarrow & x, z_0 \end{array} \qquad \begin{array}{rcl} y_i & \leftarrow & l_i \ S_i \\ z_i & \leftarrow & u_i{+}1 \ S_i \end{array}$$

where x, y_i, z_i are new symbols and S_0^+ gives all atoms in S_0

Full-fledged cardinality rules

- A rule of the form

$$l_0 \ S_0 \ u_0 \leftarrow l_1 \ S_1 \ u_1, \ldots, l_n \ S_n \ u_n$$

where each $l_i \ S_i \ u_i$ is a cardinality constraint for $0 \leq i \leq n$ stands for

$$\begin{array}{rcl} x & \leftarrow & y_1, \ldots, y_n, \neg z_1, \ldots, \neg z_n \\ {S_0}^+ & \leftarrow & x \\ & \leftarrow & x, \neg y_0 \\ & \leftarrow & x, z_0 \end{array} \qquad \begin{array}{rcl} y_i & \leftarrow & l_i \ S_i \\ z_i & \leftarrow & u_i{+}1 \ S_i \end{array}$$

where x, y_i, z_i are new symbols and S_0^+ gives all atoms in S_0

Outline

Weight rule

- Purpose Bound (lower) sum of subsets of literal weights
- Syntax A weighted literal $w : k$ associates weight w with literal k
- Syntax A weight rule is the form

$$a_0 \leftarrow l \; \{ \; w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \neg a_{m+1}, \ldots, w_n : \neg a_n \; \}$$

where $0 \leq m \leq n$ and each a_i is an atom
and l and w_i are integers for $1 \leq i \leq n$

Weight rule

- Purpose Bound (lower) sum of subsets of literal weights
- Syntax A weighted literal $w : k$ associates weight w with literal k
- Syntax A weight rule is the form

$$a_0 \leftarrow l \ \{ \ w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \neg a_{m+1}, \ldots, w_n : \neg a_n \ \}$$

  where $0 \leq m \leq n$ and each a_i is an atom

  and l and w_i are integers for $1 \leq i \leq n$

- Informal meaning The head belongs to the stable model, if the sum of weights associated with positive/negative body literals in/excluded in the stable model is at least l

Weight rule

- Syntax A weighted literal $w : k$ associates weight w with literal k
- Syntax A weight rule is the form

$$a_0 \leftarrow l \; \{ \; w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \neg a_{m+1}, \ldots, w_n : \neg a_n \; \}$$

  where $0 \leq m \leq n$ and each a_i is an atom

  and l and w_i are integers for $1 \leq i \leq n$
- Note
  - A cardinality rule is a weight rule where $w_i = 1$ for $0 \leq i \leq n$

Weight rule

- Syntax A weighted literal $w : k$ associates weight w with literal k
- Syntax A weight rule is the form

$$a_0 \leftarrow l \; \{ \; w_1 : a_1, \dots, w_m : a_m, w_{m+1} : \neg a_{m+1}, \dots, w_n : \neg a_n \; \}$$

  where $0 \leq m \leq n$ and each a_i is an atom

  and l and w_i are integers for $1 \leq i \leq n$

- Note
  - A cardinality rule is a weight rule where $w_i = 1$ for $0 \leq i \leq n$
  - Weight constraints generalize cardinality constraints accordingly and amount to constraints on count and sum aggregate functions

Weight rule

- Syntax A weighted literal $w : k$ associates weight w with literal k
- Syntax A weight rule is the form

$$a_0 \leftarrow l \; \{ \; w_1 : a_1, \dots, w_m : a_m, w_{m+1} : \neg a_{m+1}, \dots, w_n : \neg a_n \; \}$$

where $0 \leq m \leq n$ and each a_i is an atom

and l and w_i are integers for $1 \leq i \leq n$

- Note
  - A cardinality rule is a weight rule where $w_i = 1$ for $0 \leq i \leq n$
  - Weight constraints generalize cardinality constraints accordingly and amount to constraints on count and sum aggregate functions
- Example

```
5 { 4:course(db); 6:course(ai); 3:course(xml) } 10
```

Potassco

Outline

Potassco

Conditional literals

- Syntax A conditional literal is of the form

$$l : l_1, \dots, l_n$$

where l and l_i are literals for $0 \leq i \leq n$

Conditional literals

- Syntax A conditional literal is of the form

$$l : l_1, \dots, l_n$$

where l and l_i are literals for $0 \leq i \leq n$

- Informal meaning A (non-ground) conditional literal can be regarded as the collection of elements in the set $\{l \mid l_1, \dots, l_n\}$

Conditional literals

- Syntax A conditional literal is of the form

$$l : l_1, \ldots, l_n$$

  where l and l_i are literals for $0 \leq i \leq n$

- Informal meaning A (non-ground) conditional literal can be regarded as the collection of elements in the set $\{l \mid l_1, \ldots, l_n\}$

  ➥ The expansion of this collection is context dependent

Conditional literals

- Syntax A conditional literal is of the form

$$l : l_1, \dots, l_n$$

  where l and l_i are literals for $0 \leq i \leq n$

- Example Assume '`p(1..3).  q(2).`'
  - `r(X) : p(X), not q(X)` yields `r(1)` and `r(3)`

Conditional literals

- Syntax A conditional literal is of the form

$$l : l_1, \dots, l_n$$

where l and l_i are literals for $0 \leq i \leq n$

- Example Assume '`p(1..3). q(2).`'
  - `r(X) : p(X), not q(X)` yields `r(1)` and `r(3)`

The rule

```
r(X) : p(X), not q(X) :- r(X) : p(X), not q(X);
                          1 { r(X) : p(X), not q(X) }.
```

is instantiated to

```
r(1); r(3) :- r(1), r(3), 1 { r(1); r(3) }.
```

Outline

Potassco

Optimization statement

- Purpose Express (multiple) cost functions subject to minimization (and/or maximization)
- Syntax A minimize statement is of the form

$$minimize \; \{ \; w_1@p_1 : l_{1_1}, \dots, l_{m_1}; \dots; w_n@p_n : l_{1_n}, \dots, l_{m_n} \; \}.$$

  where each l_{j_i} is a literal and w_i and p_i are integers for $1 \leq i \leq n$

  priority levels, p_i, allow for representing lexicographically ordered minimization objectives

- Meaning A minimize statement is a directive that instructs the ASP solver to compute optimal stable models by minimizing a sum of weights (by descending levels)

Potassco

Optimization statement

- Purpose Express (multiple) cost functions subject to minimization (and/or maximization)
- Syntax A minimize statement is of the form

$$minimize\ \{\ w_1@p_1 : l_{1_1}, \dots, l_{m_1}; \dots; w_n@p_n : l_{1_n}, \dots, l_{m_n}\ \}.$$

where each l_{j_i} is a literal and w_i and p_i are integers for $1 \leq i \leq n$

priority levels, p_i, allow for representing lexicographically ordered minimization objectives

- Meaning A minimize statement is a directive that instructs the ASP solver to compute optimal stable models by minimizing a sum of weights (by descending levels)

Optimization statement

- Purpose Express (multiple) cost functions subject to minimization (and/or maximization)
- Syntax A minimize statement is of the form

$$\mathit{minimize}\ \{\ w_1@p_1 : l_{1_1}, \ldots, l_{m_1}; \ldots; w_n@p_n : l_{1_n}, \ldots, l_{m_n}\ \}.$$

  where each l_{j_i} is a literal and w_i and p_i are integers for $1 \leq i \leq n$

  priority levels, p_i, allow for representing lexicographically ordered minimization objectives

- Meaning A minimize statement is a directive that instructs the ASP solver to compute optimal stable models by minimizing a sum of weights (by descending levels)

Optimization statement

- A maximize statement of the form

$$maximize\ \{\ w_1@p_1 : l_1, \dots, w_n@p_n : l_n\ \}$$

  stands for $minimize\ \{\ -w_1@p_1 : l_1, \dots, -w_n@p_n : l_n\ \}$

- Example When configuring a computer, we may want to maximize hard disk capacity, while minimizing price

```
#maximize { 250@1:hd(1); 500@1:hd(2); 750@1:hd(3) }.
#minimize { 30@2:hd(1);  40@2:hd(2);  60@2:hd(3) }.
```

  The priority levels indicate that (minimizing) price is more important than (maximizing) capacity

Optimization statement

- A maximize statement of the form

$$maximize \ \{ \ w_1@p_1 : l_1, \dots, w_n@p_n : l_n \ \}$$

stands for $minimize \ \{ \ -w_1@p_1 : l_1, \dots, -w_n@p_n : l_n \ \}$

- Example When configuring a computer, we may want to maximize hard disk capacity, while minimizing price

```
#maximize { 250@1:hd(1); 500@1:hd(2); 750@1:hd(3) }.
#minimize { 30@2:hd(1);  40@2:hd(2);  60@2:hd(3) }.
```

The priority levels indicate that (minimizing) price is more important than (maximizing) capacity

Optimization statement

- A maximize statement of the form

$$maximize\ \{\ w_1@p_1 : l_1, \dots, w_n@p_n : l_n\ \}$$

stands for $minimize\ \{\ -w_1@p_1 : l_1, \dots, -w_n@p_n : l_n\ \}$

- Example When configuring a computer, we may want to maximize hard disk capacity, while minimizing price

```
#maximize { C@1:hd(I,P,C) }.
#minimize { P@2:hd(I,P,C) }.
```

The priority levels indicate that (minimizing) price is more important than (maximizing) capacity

Weak constraints

- Weak constraints are an alternative to minimize statements
- Syntax $\; \leftarrow\!\!\!\sim l_1, \dots, l_n \; [w@p]$

  where each l_i is a literal for $1 \leq i \leq n$; and w and p are integers
- Example

```
:~ hd(1). [30@2]
:~ hd(2). [40@2]
:~ hd(3). [60@2]
```

Potassco

Weak constraints

- Weak constraints are an alternative to minimize statements
- Syntax $\quad \leftsquigarrow l_1, \dots, l_n \; [w@p]$

  where each l_i is a literal for $1 \leq i \leq n$; and w and p are integers
- Example

```
:~ hd(1). [30@2]
:~ hd(2). [40@2]
:~ hd(3). [60@2]
```

Weak constraints

- Weak constraints are an alternative to minimize statements
- Syntax $\leftsquigarrow l_1, \ldots, l_n\ [w@p]$
  where each l_i is a literal for $1 \leq i \leq n$; and w and p are integers
- Example

```
:~ hd(1). [30@2]
:~ hd(2). [40@2]
:~ hd(3). [60@2]
```

Weak constraints

- Weak constraints are an alternative to minimize statements
- Syntax $\;\leftsquigarrow l_1, \dots, l_n \; [w@p]$

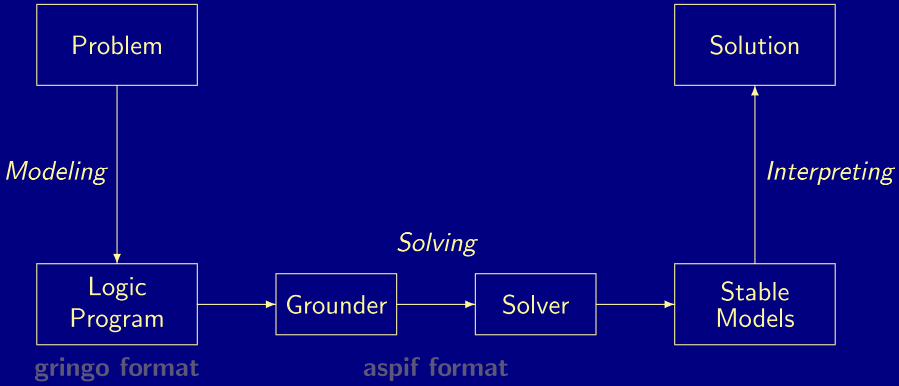
  where each l_i is a literal for $1 \leq i \leq n$; and w and p are integers
- Example

```
:~ hd(I,P,C). [P@2]
```

Potassco

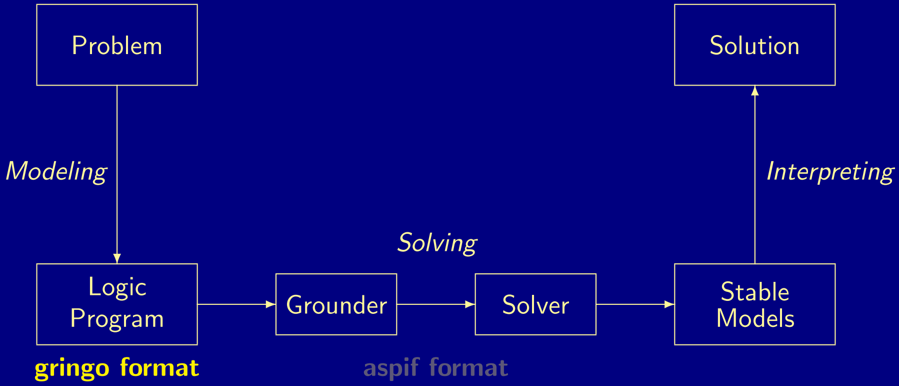
Outline

Gringo's input and output language



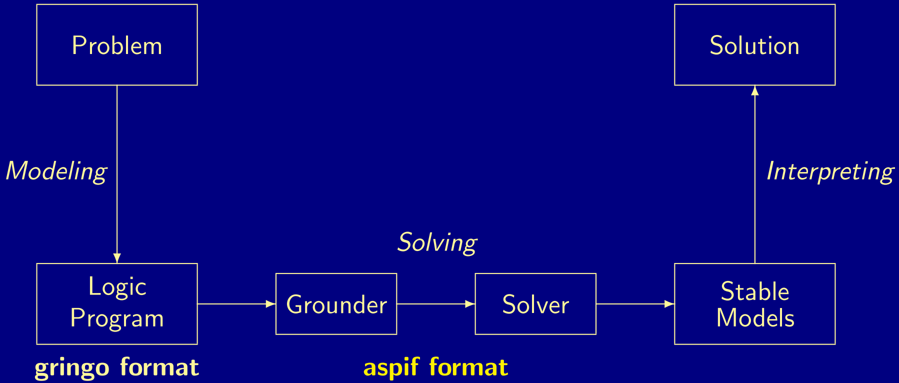
- gringo format is a user-oriented language for (non-ground) programs extending the ASP language standard *ASP-Core-2* [13]
- aspif format is a machine-oriented standard for ground programs

Gringo's input and output language



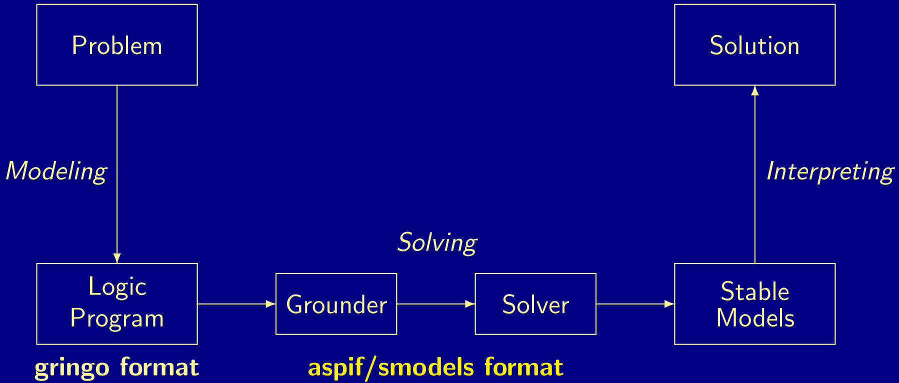
- gringo format is a user-oriented language for (non-ground) programs extending the ASP language standard *ASP-Core-2* [13]
- aspif format is a machine-oriented standard for ground programs

Gringo's input and output language



- gringo format is a user-oriented language for (non-ground) programs extending the ASP language standard *ASP-Core-2* [13]
- aspif format is a machine-oriented standard for ground programs

Gringo's input and output language



- gringo format is a user-oriented language for (non-ground) programs extending the ASP language standard *ASP-Core-2* [13]
- aspif format is a machine-oriented standard for ground programs

Outline

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $-a$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Potassco

Terms and literals

- Terms t are formed from
  - constant symbols, eg `c`, `d`, ...
  - function symbols, eg `f`, `g`, ...
  - numeral symbols, eg 1, 2, ...
  - variable symbols, eg `X`, `Y`, ..., `_`
  - parentheses (,)
  - tuple delimiters $\langle, \rangle$ (omitted whenever possible)
- Tuples $\boldsymbol{t}$
- Atoms a, $-a$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Terms and literals

- Terms t are formed from
  - constants, eg `c`, `d`, ...
  - functions, eg `f`, `g`, ...
  - numerals, eg 1, 2, ...
  - variables, eg `X`, `Y`, ..., `_`
  - parentheses (,)
  - tuple delimiters $\langle$, $\rangle$
- Tuples $\boldsymbol{t}$
- Atoms a, $-a$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Potassco

Terms and literals

- Terms t are formed from
  - constants, eg `c`, `d`, ...
  - functions, eg `f`, `g`, ...
  - numerals, eg 1, 2, ...
  - variables, eg `X`, `Y`, ..., `_`
  - parentheses (,)
  - tuple delimiters $\langle$, $\rangle$

  eg `f(3,c,Z)`, `g(42,_,_)`, or `f((3,c),X)`
- Tuples $\boldsymbol{t}$
- Atoms a, $-a$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Potassco

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$ of terms
- Atoms $a, -a$
- Symbolic literals $a, \neg a, \neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals $a, \neg a, \neg\neg a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- (Negated) Atoms a, $-a$ are formed from
  - predicate symbols, eg p, q, ...
  - parentheses (,)
  - tuples of terms
- Symbolic literals $a, \neg a, \neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals $a, \neg a, \neg\neg a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $-a$ are formed from
  - predicates, eg p, q, ...
  - parentheses (,)
  - tuples of terms
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Potassco

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $-a$ are formed from
  - predicates, eg `p`, `q`, ...
  - parentheses (,)
  - tuples of terms

  eg `-p(f(3,c,Z),g(42,_,_))` or `q()` written as `q`
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Potassco

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Potassco

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $-a$, $\bot$, $\top$
  viz `#false` and `#true`
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Potassco

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
  eg `p(a,X)`, '`not p(a,X)`', '`not not p(a,X)`'
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$ where
  - t_1 and t_2 are terms
  - $\prec$ is a comparison symbol
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Potassco

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$ where
  - t_1 and t_2 are terms
  - $\prec$ is a comparison symbol

  eg `3<1` or `f(42)=X`
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$ of literals
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$ where
  - l is a symbolic or arithmetic literal
  - $\boldsymbol{L}$ is a tuple of symbolic or arithmetic literals
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$ where
  - l is a symbolic or arithmetic literal
  - $\boldsymbol{L}$ is a tuple of symbolic or arithmetic literals
  - $l : \boldsymbol{L}$ is written as l whenever $\boldsymbol{L}$ is empty
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$ where
  - l is a symbolic or arithmetic literal
  - $\boldsymbol{L}$ is a tuple of symbolic or arithmetic literals

  eg '`p(X,Y):q(X),r(Y)`' or `p(42)` or '`#false:q`'
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$ where
  - α is an aggregate name
  - $\boldsymbol{t}_1 : \boldsymbol{L}_1, \dots, \boldsymbol{t_n} : \boldsymbol{L_n}$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - s_1 and s_2 are terms
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$ where
  - α is an aggregate name
  - $\boldsymbol{t}_1 : \boldsymbol{L}_1, \dots, \boldsymbol{t_n} : \boldsymbol{L_n}$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - s_1 and s_2 are terms
  - one (or both) of '$s_1 \prec_1$' and '$\prec_2 s_2$' can be omitted
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals

Potassco

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$ where
  - α is an aggregate name
  - $\boldsymbol{t}_1 : \boldsymbol{L}_1, \dots, \boldsymbol{t}_n : \boldsymbol{L}_n$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - s_1 and s_2 are terms
  - omitting $\prec_1$ or $\prec_2$ defaults to $\leq$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals are conditional or aggregate literals

Potassco

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$ where
  - α is an aggregate name
  - $\boldsymbol{t}_1 : \boldsymbol{L}_1, \dots, \boldsymbol{t_n} : \boldsymbol{L_n}$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - s_1 and s_2 are terms

  eg `10 <= #sum {6,C:course(C); 3,S:seminar(S)} <= 20`
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals are conditional or aggregate literals

Terms and literals

- Terms t
- Tuples $\boldsymbol{t}$, $\boldsymbol{L}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \boldsymbol{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_n : \boldsymbol{L}_n\} \prec_2 s_2$ where
  - α is an aggregate name
  - $\boldsymbol{t}_1 : \boldsymbol{L}_1, \dots, \boldsymbol{t_n} : \boldsymbol{L_n}$ are conditional literals
  - $\prec_1$ and $\prec_2$ are comparison symbols
  - s_1 and s_2 are terms

  eg `10 #sum {6,C:course(C); 3,S:seminar(S)} 20`
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals are conditional or aggregate literals

Terms and literals

- Terms t
- Tuples $\mathbf{t}$, $\mathbf{L}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \mathbf{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\mathbf{t}_1 : \mathbf{L}_1; \dots; \mathbf{t}_n : \mathbf{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$ where
  - a is an aggregate atom
- Literals are conditional or aggregate literals

Terms and literals

- Terms t
- Tuples $\mathbf{t}$, $\mathbf{L}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \mathbf{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\mathbf{t}_1 : \mathbf{L}_1; \dots; \mathbf{t}_n : \mathbf{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$ where
  - a is an aggregate atom

  eg `not 10 #sum {6,C:course(C); 3,S:seminar(S)} 20`
- Literals are conditional or aggregate literals

Terms and literals

- Terms t
- Tuples $\mathbf{t}$, $\mathbf{L}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \mathbf{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\mathbf{t}_1 : \mathbf{L}_1; \dots; \mathbf{t}_n : \mathbf{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals are conditional or aggregate literals

Terms and literals

- Terms t
- Tuples $\mathbf{t}$, $\mathbf{L}$
- Atoms a, $-a$, $\bot$, $\top$
- Symbolic literals a, $\neg a$, $\neg\neg a$
- Arithmetic literals $t_1 \prec t_2$
- Conditional literals $l : \mathbf{L}$
- Aggregate atoms $s_1 \prec_1 \alpha\{\mathbf{t}_1 : \mathbf{L}_1; \dots; \mathbf{t}_n : \mathbf{L}_n\} \prec_2 s_2$
- Aggregate literals a, $\neg a$, $\neg\neg a$
- Literals are conditional or aggregate literals

- For a detailed account please consult the user's guide!

Rules

- Rules are of the form

$$l_1\,;\ldots\,;l_m \leftarrow l_{m+1},\ldots,l_n \tag{1}$$

  where
  - l_i is a conditional literal for $1 \leq i \leq m$ and
  - l_i is a literal for $m+1 \leq i \leq n$

- Note Semicolons ';' must be used in (1) instead of commas ','
  whenever some l_i is a (genuine) conditional literal for $1 \leq i \leq n$

- Example `a(X) :- b(X) : c(X), d(X); e(x).`

- Note $l_1\,;\ldots\,;l_m \leftarrow l_{m+1},\ldots,l_n$ is the same as
  $l_1\,;\ldots\,;l_m \leftarrow l_{m+1};\ldots;l_n$

Rules

- Rules are of the form

$$l_1\,;\ldots\,;l_m \leftarrow l_{m+1},\ldots,l_n \tag{1}$$

where

  - l_i is a conditional literal for $1 \leq i \leq m$ and
  - l_i is a literal for $m+1 \leq i \leq n$

- Note Semicolons ';' must be used in (1) instead of commas ','
whenever some l_i is a (genuine) conditional literal for $1 \leq i \leq n$

- Example `a(X) :- b(X) : c(X), d(X); e(x).`

- Note $l_1\,;\ldots\,;l_m \leftarrow l_{m+1},\ldots,l_n$ is the same as
$l_1\,;\ldots\,;l_m \leftarrow l_{m+1};\ldots;l_n$

Potassco

Rules

- Rules are of the form

$$l_1 ; \dots ; l_m \leftarrow l_{m+1}, \dots, l_n \tag{1}$$

  where
  - l_i is a conditional literal for $1 \leq i \leq m$ and
  - l_i is a literal for $m+1 \leq i \leq n$
- Note Semicolons ';' must be used in (1) instead of commas ','
  whenever some l_i is a (genuine) conditional literal for $1 \leq i \leq n$
- Example `a(X) :- b(X) : c(X), d(X); e(x).`
- Note $l_1 ; \dots ; l_m \leftarrow l_{m+1}, \dots, l_n$ is the same as
  $l_1 ; \dots ; l_m \leftarrow l_{m+1}; \dots ; l_n$

Rules

- Rules are of the form

$$l_1 \,;\, \dots \,;\, l_m \leftarrow l_{m+1}, \dots, l_n \qquad (1)$$

  where
  - l_i is a conditional literal for $1 \leq i \leq m$ and
  - l_i is a literal for $m+1 \leq i \leq n$

- Note Semicolons ';' must be used in (1) instead of commas ','
  whenever some l_i is a (genuine) conditional literal for $1 \leq i \leq n$
- Example `a(X) :- b(X) : c(X), d(X); e(x).`
- Note $l_1 \,;\, \dots \,;\, l_m \leftarrow l_{m+1}, \dots, l_n$ is the same as
  $l_1 \,;\, \dots \,;\, l_m \leftarrow l_{m+1}; \dots; l_n$

Shortcuts

- A rule of the form

$$s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : l_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_k : l_k : \boldsymbol{L}_k\} \prec_2 s_2 \ \leftarrow \ l_{m+1}, \dots , l_n$$

where

  - α, $\prec_i$, s_i, $\boldsymbol{t}_j$ are as given above for $i = 1, 2$ and $1 \leq j \leq k$
  - $l_j : \boldsymbol{L}_j$ is a conditional literal for $1 \leq j \leq k$
  - l_i is a literal for $m + 1 \leq i \leq n$ (as in (1))

is a shorthand for the following $k + 1$ rules

$$\{l_j\} \leftarrow l_{m+1}, \dots , l_n, \boldsymbol{L}_j \qquad \text{for } 1 \leq j \leq k$$

$$\leftarrow l_{m+1}, \dots , l_n, \neg\, s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : l_1, \boldsymbol{L}_1; \dots ; \boldsymbol{t}_k : l_k, \boldsymbol{L}_k\} \prec_2 s_2$$

- Example `10 < #sum { C,X,Y : edge(X,Y) : cost(X,Y,C) }.`

Shortcuts

- A rule of the form

$$s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : l_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_k : l_k : \boldsymbol{L}_k\} \prec_2 s_2 \ \leftarrow \ l_{m+1}, \dots, l_n$$

where

  - α, $\prec_i$, s_i, $\boldsymbol{t}_j$ are as given above for $i = 1, 2$ and $1 \leq j \leq k$
  - $l_j : \boldsymbol{L}_j$ is a conditional literal for $1 \leq j \leq k$
  - l_i is a literal for $m + 1 \leq i \leq n$ (as in (1))

is a shorthand for the following $k + 1$ rules

$$\{l_j\} \leftarrow l_{m+1}, \dots, l_n, \boldsymbol{L}_j \qquad \text{for } 1 \leq j \leq k$$

$$\leftarrow l_{m+1}, \dots, l_n, \neg\, s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : l_1, \boldsymbol{L}_1; \dots ; \boldsymbol{t}_k : l_k, \boldsymbol{L}_k\} \prec_2 s_2$$

- Example `10 < #sum { C,X,Y : edge(X,Y) : cost(X,Y,C) }.`

Potassco

Shortcuts

- A rule of the form

$$s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : l_1 : \boldsymbol{L}_1; \dots ; \boldsymbol{t}_k : l_k : \boldsymbol{L}_k\} \prec_2 s_2 \ \leftarrow \ l_{m+1}, \dots , l_n$$

  where
  - α, $\prec_i$, s_i, $\boldsymbol{t}_j$ are as given above for $i = 1, 2$ and $1 \leq j \leq k$
  - $l_j : \boldsymbol{L}_j$ is a conditional literal for $1 \leq j \leq k$
  - l_i is a literal for $m + 1 \leq i \leq n$ (as in (1))

  is a shorthand for the following $k + 1$ rules

$$\{l_j\} \leftarrow l_{m+1}, \dots , l_n, \boldsymbol{L}_j \qquad \text{for } 1 \leq j \leq k$$

$$\leftarrow l_{m+1}, \dots , l_n, \neg\, s_1 \prec_1 \alpha\{\boldsymbol{t}_1 : l_1, \boldsymbol{L}_1; \dots ; \boldsymbol{t}_k : l_k, \boldsymbol{L}_k\} \prec_2 s_2$$

- Example `10 < #sum { C,X,Y : edge(X,Y) : cost(X,Y,C) }.`

Shortcuts

- The expression

$$s_1 \ \{l_1 : \boldsymbol{L}_1; \dots ; l_k : \boldsymbol{L}_k\} \ s_2$$

  is a shortcut for

  - $s_1 \leq count\{t_1 : l_1 : \boldsymbol{L}_1; \dots ; t_k : l_k : \boldsymbol{L}_k\} \leq s_2$
    if it appears in the head of a rule and
  - $s_1 \leq count\{t_1 : l_1, \boldsymbol{L}_1; \dots ; t_k : l_k, \boldsymbol{L}_k\} \leq s_2$
    if it appears in the body of a rule

  where $t_i \neq t_j$ whenever $l_i \neq l_j$ for $i \neq j$ and $1 \leq i, j \leq k$

- Note one (or both) of s_1 and s_2 can be omitted

Shortcuts

- The expression

$$s_1 \ \{l_1 : \boldsymbol{L}_1; \ldots; l_k : \boldsymbol{L}_k\} \ s_2$$

  is a shortcut for

  - $s_1 \leq count\{t_1 : l_1 : \boldsymbol{L}_1; \ldots; t_k : l_k : \boldsymbol{L}_k\} \leq s_2$
    if it appears in the head of a rule and
  - $s_1 \leq count\{t_1 : l_1, \boldsymbol{L}_1; \ldots; t_k : l_k, \boldsymbol{L}_k\} \leq s_2$
    if it appears in the body of a rule

  where $t_i \neq t_j$ whenever $l_i \neq l_j$ for $i \neq j$ and $1 \leq i, j \leq k$

- Note one (or both) of s_1 and s_2 can be omitted

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`

  Assume '`travel(a,b,3). travel(b,c,3).`'

  We get `sum(3)` !

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`
  Assume '`travel(a,b,3). travel(b,c,3).`'
  We get `sum(3)` !

Potassco

Aggregates on sets

- Example `sum(S) :- S = #sum { C : travel(X,Y,C) }.`
  Assume '`travel(a,b,3). travel(b,c,3).`'
  We get `sum(3)` !

Aggregates on sets

- Example `sum(S) :- S = #sum { C : travel(X,Y,C) }.`

  Assume '`travel(a,b,3). travel(b,c,3).`'

  We get `sum(3)` !

Aggregates on sets

- Example `sum(S) :- S = #sum { C : travel(X,Y,C) }.`
  Assume '`travel(a,b,3). travel(b,c,3).`'
  We get `sum(3)` !
- How come?

Aggregates on sets

- Example `sum(S) :- S = #sum { C : travel(X,Y,C) }.`

  Assume '`travel(a,b,3). travel(b,c,3).`'

  We get `sum(3)` !

- How come?

  `#sum { C : travel(X,Y,C) }`

Aggregates on sets

- Example `sum(S) :- S = #sum { C : travel(X,Y,C) }.`

  Assume '`travel(a,b,3). travel(b,c,3).`'

  We get `sum(3)`!

- How come?

  `#sum { 3 : travel(a,b,3); 3 : travel(b,c,3) }`

Aggregates on sets

- Example `sum(S) :- S = #sum { C : travel(X,Y,C) }.`

  Assume '`travel(a,b,3). travel(b,c,3).`'

  We get `sum(3)` !

- How come?

  `#sum { 3; 3 }`

Aggregates on sets

- Example `sum(S) :- S = #sum { C : travel(X,Y,C) }.`

  Assume '`travel(a,b,3). travel(b,c,3).`'

  We get `sum(3)`!

- How come?

  `#sum { 3 }`

Aggregates on sets

- Example `sum(S) :- S = #sum { C : travel(X,Y,C) }.`

  Assume '`travel(a,b,3). travel(b,c,3).`'

  We get `sum(3)` !

- How come?

  `sum(3) :- 3 = #sum { 3 }.`

Aggregates on sets

- Example `sum(S) :- S = #sum { C : travel(X,Y,C) }.`
  Assume '`travel(a,b,3). travel(b,c,3).`'
  We get `sum(3)`!

- How come?
  `sum(3).`

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`
  Assume '`travel(a,b,3). travel(b,c,3).`'
  We get `sum(3)` !

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`

  Assume '`travel(a,b,3). travel(b,c,3).`'

  We get `sum(6)` !

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`
  Assume '`travel(a,b,3). travel(b,c,3).`'
  We get `sum(6)`!

- How come?
  `#sum { C,X,Y : travel(X,Y,C) }`

Potassco

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`
  Assume '`travel(a,b,3). travel(b,c,3).`'
  We get `sum(6)` !

- How come?
  `#sum { 3,a,b : travel(a,b,3); 3,b,c : travel(b,c,3) }`

Potassco

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`
  Assume '`travel(a,b,3). travel(b,c,3).`'
  We get `sum(6)` !

- How come?
  `#sum { 3,a,b; 3,b,c }`

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`
  Assume '`travel(a,b,3). travel(b,c,3).`'
  We get `sum(6)` !

- How come?
  `#sum { (3,a,b); (3,b,c) }`

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`
  Assume '`travel(a,b,3). travel(b,c,3).`'
  We get `sum(6)` !

- How come?
  `sum(6) :- 6 = #sum { (3,a,b); (3,b,c) }.`

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`

  Assume ‘`travel(a,b,3). travel(b,c,3).`’

  We get `sum(6)`!

- How come?

  `sum(6).`

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`

  Assume '`travel(a,b,3). travel(b,c,3).`'

  We get `sum(6)` !

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`
  Assume '`travel(a,b,3). travel(b,c,3).`'
  We get `sum(6)` !

- Example

```
$ gringo --text <(echo "{a;b}.")
#count{ 1,0,a : a; 1,0,b : b }.
```

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`

  Assume '`travel(a,b,3). travel(b,c,3).`'

  We get `sum(6)` !

- Example

  ```
  $ gringo --text <(echo "{a;b}.")
  #count{ 1,0,a : a; 1,0,b : b }.
  ```

  ➥ *gringo* generates two distinct term tuples `(1,0,a)` and `(1,0,b)`

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`
  Assume '`travel(a,b,3). travel(b,c,3).`'
  We get `sum(6)`!

- Example

  ```
  $ gringo --text <(echo "{a;b}.")
  #count{ 1,0,a : a; 1,0,b : b }.
  ```

  ➥ *gringo* generates two distinct term tuples `(1,0,a)` and `(1,0,b)`

- Example Why ";"?

  `{ q(X,Y): p(X), p(Y), X < Y; q(X,X): p(X) } = 1`

Aggregates on sets

- Example `sum(S) :- S = #sum { C,X,Y : travel(X,Y,C) }.`

  Assume '`travel(a,b,3). travel(b,c,3).`'

  We get `sum(6)` !

- Example

  ```
  $ gringo --text <(echo "{a;b}.")
  #count{ 1,0,a : a; 1,0,b : b }.
  ```

  ➥ *gringo* generates two distinct term tuples `(1,0,a)` and `(1,0,b)`

- Example Why ";"?

  ```
  { q(X,Y): p(X), p(Y), X < Y; q(X,X): p(X) } = 1
  ```

Weak constraints

- Syntax A weak constraint is of the form

$$\leftarrow\!\sim l_1, \ldots, l_n \; [w@p, t_1, \ldots, t_m]$$

  where
  - $l_1, \ldots, l_n$ are literals
  - $t_1, \ldots, t_m$, w, and p are terms

  w and p stand for a weight and priority level
  ($p = 0$ if '$@p$' is omitted)

- Example The weak constraint

```
:~ hd(I,P,C). [P@42,I]
```

  amounts to the minimize statement

```
#minimize{ P@42,I : hd(I,P,C) }.
```

Potassco

Weak constraints

- Syntax A weak constraint is of the form

$$\leftsquigarrow l_1, \ldots, l_n \; [w@p, t_1, \ldots, t_m]$$

  where
  - $l_1, \ldots, l_n$ are literals
  - $t_1, \ldots, t_m$, w, and p are terms

  w and p stand for a weight and priority level
  ($p = 0$ if '$@p$' is omitted)

- Example The weak constraint

```
:~ hd(I,P,C). [P@42,I]
```

  amounts to the minimize statement

```
#minimize{ P@42,I : hd(I,P,C) }.
```

Weak constraints

- Syntax A weak constraint is of the form

$$\leftarrow\!\!\sim\ l_1, \dots, l_n\ [w@p, t_1, \dots, t_m]$$

  where
  - $l_1, \dots, l_n$ are literals
  - $t_1, \dots, t_m$, w, and p are terms

  w and p stand for a weight and priority level
  ($p = 0$ if '@p' is omitted)

- Example The weak constraint

```
:~ hd(I,P,C). [P@42,I]
```

  amounts to the minimize statement

```
#minimize{ P@42,I : hd(I,P,C) }.
```

Some more directives

- Output

$$\texttt{\#show.} \qquad \texttt{\#show}\ p/n. \qquad \texttt{\#show}\ t\ :\ l_1, \dots, l_n.$$

- Projection

$$\texttt{\#project}\ p/n. \qquad \texttt{\#project}\ a\ :\ l_1, \dots, l_n.$$

- Heuristics

$$\texttt{\#heuristic}\ a\ :\ l_1, \dots, l_n.\ [k@p, m]$$

- Acyclicity

$$\texttt{\#edge}\ (u, v)\ :\ l_1, \dots, l_n.$$

Some more directives

- Output

$$\texttt{\#show}. \qquad \texttt{\#show}\ p/n. \qquad \texttt{\#show}\ t\ :\ l_1, \dots, l_n.$$

- Projection

$$\texttt{\#project}\ p/n. \qquad \texttt{\#project}\ a\ :\ l_1, \dots, l_n.$$

- Heuristics

$$\texttt{\#heuristic}\ a\ :\ l_1, \dots, l_n.\ [k@p, m]$$

- Acyclicity

$$\texttt{\#edge}\ (u, v)\ :\ l_1, \dots, l_n.$$

Some more directives

- Output

  $\texttt{\#show.} \qquad \texttt{\#show}\ p/n. \qquad \texttt{\#show}\ t\ :\ l_1, \dots, l_n.$

- Projection

  $\texttt{\#project}\ p/n. \qquad \texttt{\#project}\ a\ :\ l_1, \dots, l_n.$

- Heuristics

  $\texttt{\#heuristic}\ a\ :\ l_1, \dots, l_n.\ [k@p, m]$

- Acyclicity

  $\texttt{\#edge}\ (u, v)\ :\ l_1, \dots, l_n.$

Some more directives

- Output

$$\#\texttt{show}. \qquad \#\texttt{show}\ p/n. \qquad \#\texttt{show}\ t\ :\ l_1,\dots,l_n.$$

- Projection

$$\#\texttt{project}\ p/n. \qquad \#\texttt{project}\ a\ :\ l_1,\dots,l_n.$$

- Heuristics

$$\#\texttt{heuristic}\ a\ :\ l_1,\dots,l_n.\ [k@p,m]$$

- Acyclicity

$$\#\texttt{edge}\ (u,v)\ :\ l_1,\dots,l_n.$$

Outline

smodels format

- The *smodels* format consists of
  - normal rules
  - choice rules
  - cardinality rules
  - weight rules
  - minimization statements
- Block-oriented format

smodels format in detail

Type/Format
Normal rule Slide 92
$1␣\iota(a_0)␣n␣n-m␣\iota(a_{m+1})␣\dots␣\iota(a_n)␣\iota(a_1)␣\dots␣\iota(a_m)$
Cardinality rule Slide 553
$2␣\iota(a_0)␣n␣n-m␣l␣\iota(a_{m+1})␣\dots␣\iota(a_n)␣\iota(a_1)␣\dots␣\iota(a_m)$
Choice rule Slide 541
$3␣m␣\iota(a_1)␣\dots␣\iota(a_m)␣o-m␣o-n␣\iota(a_{n+1})␣\dots␣\iota(a_o)␣\iota(a_{m+1})␣\dots␣\iota(a_n)$
Weight rule Slide 581
$5␣\iota(a_0)␣l␣n␣n-m␣\iota(a_{m+1})␣\dots␣\iota(a_n)␣\iota(a_1)␣\dots␣\iota(a_m)␣w_{m+1}␣\dots␣w_n␣w_1␣\dots␣w_m$
Minimize statement[2] Slide 593
$6␣0␣n␣n-m␣\iota(a_{m+1})␣\dots␣\iota(a_n)␣\iota(a_1)␣\dots␣\iota(a_m)␣w_{m+1}␣\dots␣w_n␣w_1␣\dots␣w_m$
Disjunctive rule Slide 723
$8␣m␣\iota(a_1)␣\dots␣\iota(a_m)␣o-m␣o-n␣\iota(a_{n+1})␣\dots␣\iota(a_o)␣\iota(a_{m+1})␣\dots␣\iota(a_n)$

- The function ι represents a mapping of atoms to numbers

Example

```
{a}.
b :- a.
c :- not a.
```

```
3 1 2 0 0
1 3 1 1 2
1 4 1 0 2
0
2 a
3 c
4 b
0
B+
0
B-
1
0
1
```

Potassco

Example

```
{a}.
b :- a.
c :- not a.
```

```
3 1 2 0 0
1 3 1 1 2
1 4 1 0 2
0
2 a
3 c
4 b
0
B+
0
B-
1
0
1
```

Potassco

aspif format

- The *aspif* format consists of
  - rule statements
  - minimize statements
  - projection statements
  - output statements
  - external statements
  - assumption statements
  - heuristic statements
  - edge statements
  - theory terms and atoms
  - comments
- Line-oriented format

Rule statements

Rule statements have the form $1␣H␣B$

- Head H has form $h␣m␣a_1␣\dots␣a_m$
  - $h \in \{0,1\}$ determines whether the head is a disjunction or choice,
  - $m \geq 0$ is the number of head elements, and
  - each a_i is a positive literal

Rule statements

Rule statements have the form $1␣H␣B$

- Head H has form $h␣m␣a_1␣\dots␣a_m$
  - $h \in \{0, 1\}$ determines whether the head is a disjunction or choice,
  - $m \geq 0$ is the number of head elements, and
  - each a_i is a positive literal

Rule statements

Rule statements have the form $1 \sqcup H \sqcup B$

- Head H has form $h \sqcup m \sqcup a_1 \sqcup \dots \sqcup a_m$
  - $h \in \{0, 1\}$ determines whether the head is a disjunction or choice,
  - $m \geq 0$ is the number of head elements, and
  - each a_i is a positive literal

- Note Heads are disjunctions or choices, including the special case of singular disjunctions for representing normal rules

Rule statements

Rule statements have the form $1 \sqcup H \sqcup B$

- Head H has form $h \sqcup m \sqcup a_1 \sqcup \dots \sqcup a_m$
  - $h \in \{0, 1\}$ determines whether the head is a disjunction or choice,
  - $m \geq 0$ is the number of head elements, and
  - each a_i is a positive literal
- Body B has one of two forms
  - normal bodies have form $0 \sqcup n \sqcup l_1 \sqcup \dots \sqcup l_n$
    - $n \geq 0$ is the length of the rule body, and
    - each l_i is a literal
  - weight bodies have form $1 \sqcup l \sqcup n \sqcup l_1 \sqcup w_1 \sqcup \dots \sqcup l_n \sqcup w_n$
    - l is a positive integer to denote the lower bound,
    - $n \geq 0$ is the number of literals in the rule body, and
    - each l_i and w_i are a literal and a positive integer

Potassco

Rule statements

Rule statements have the form $1␣H␣B$

- Head H has form $h␣m␣a_1␣\dots␣a_m$
  - $h \in \{0,1\}$ determines whether the head is a disjunction or choice,
  - $m \geq 0$ is the number of head elements, and
  - each a_i is a positive literal
- Body B has one of two forms
  - normal bodies have form $0␣n␣l_1␣\dots␣l_n$
    - $n \geq 0$ is the length of the rule body, and
    - each l_i is a literal
  - weight bodies have form $1␣l␣n␣l_1␣w_1␣\dots␣l_n␣w_n$
    - l is a positive integer to denote the lower bound,
    - $n \geq 0$ is the number of literals in the rule body, and
    - each l_i and w_i are a literal and a positive integer
- Note All types of rules are included in the above rule format

Example

```
{a}.
b :- a.
c :- not a.
```

```
asp 1 0 0
1 1 1 1 0 0
1 0 1 2 0 1 1
1 0 1 3 0 1 -1
4 1 a 1 1
4 1 b 1 2
4 1 c 1 3
0
```

Potassco

Example

```
{a}.
b :- a.
c :- not a.
```

```
asp 1 0 0
1 1 1 1 0 0
1 0 1 2 0 1 1
1 0 1 3 0 1 -1
4 1 a 1 1
4 1 b 1 2
4 1 c 1 3
0
```

Outline

Potassco

Things to remember

- integrity constraints
- choice rules
- cardinality rules, cardinality constraints
- weight rules, weight constraints
- conditional literals
- optimization statements and weak constraints
- input languages, *gringo*, ASP-Core-2
- intermediate languages, *smodels*, *aspif*

Language Extensions: Overview

Outline

Motivation

- Classical versus default negation
  - Symbol $-$ and $\neg$
  - Idea
    - $-a \approx -a \in X$
    - $\neg a \approx a \notin X$
  - Example
    - $cross \leftarrow -train$
    - $cross \leftarrow \neg train$

Potassco

Motivation

- Classical versus default negation
  - Symbol $-$ and $\neg$
  - Idea
    - $-a \;\approx\; -a \in X$
    - $\neg a \;\approx\; a \notin X$
  - Example
    - $cross \leftarrow -train$
    - $cross \leftarrow \neg train$

Potassco

Motivation

- Classical versus default negation
  - Symbol $-$ and $\neg$
  - Idea
    - $-a \;\approx\; -a \in X$
    - $\neg a \;\approx\; a \notin X$
  - Example
    - $cross \leftarrow -train$
    - $cross \leftarrow \neg train$

Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{-a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$
- Given a program P over $\mathcal{A}$, classical negation is encoded by adding

$$P^{-} = \{a \leftarrow b, -b \mid a \in (\mathcal{A} \cup \overline{\mathcal{A}}), b \in \mathcal{A}\}$$

- A set X of atoms is a stable model of a program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, if X is a stable model of $P \cup P^{-}$

Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{-a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$
- Given a program P over $\mathcal{A}$, classical negation is encoded by adding

$$P^{-} = \{a \leftarrow b, -b \mid a \in (\mathcal{A} \cup \overline{\mathcal{A}}), b \in \mathcal{A}\}$$

- A set X of atoms is a stable model of a program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, if X is a stable model of $P \cup P^{-}$

Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{-a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$
- Given a program P over $\mathcal{A}$, classical negation is encoded by adding

$$P^{-} = \{a \leftarrow b, -b \mid a \in (\mathcal{A} \cup \overline{\mathcal{A}}), b \in \mathcal{A}\}$$

- A set X of atoms is a stable model of a program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, if X is a stable model of $P \cup P^{-}$

Classical negation

- We consider logic programs in negation normal form
  - That is, classical negation is applied to atoms only
- Given an alphabet $\mathcal{A}$ of atoms, let $\overline{\mathcal{A}} = \{-a \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \overline{\mathcal{A}} = \emptyset$
- Given a program P over $\mathcal{A}$, classical negation is encoded by adding

$$P^{-} = \{a \leftarrow b, -b \mid a \in (\mathcal{A} \cup \overline{\mathcal{A}}), b \in \mathcal{A}\}$$

- A set X of atoms is a stable model of a program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, if X is a stable model of $P \cup P^{-}$

An example

- The program

$$P = \{a \leftarrow \neg b,\ b \leftarrow \neg a\} \cup \{c \leftarrow b,\ -c \leftarrow b\}$$

induces

$$P^- = \left\{ \begin{array}{rcl rcl rcl} a & \leftarrow & a, -a & a & \leftarrow & b, -b & a & \leftarrow & c, -c \\ -a & \leftarrow & a, -a & -a & \leftarrow & b, -b & -a & \leftarrow & c, -c \\ b & \leftarrow & a, -a & b & \leftarrow & b, -b & b & \leftarrow & c, -c \\ -b & \leftarrow & a, -a & -b & \leftarrow & b, -b & -b & \leftarrow & c, -c \\ c & \leftarrow & a, -a & c & \leftarrow & b, -b & c & \leftarrow & c, -c \\ -c & \leftarrow & a, -a & -c & \leftarrow & b, -b & -c & \leftarrow & c, -c \end{array} \right\}$$

- The stable models of P are given by the ones of $P \cup P^-$, viz $\{a\}$

An example

- The program

$$P \;=\; \{a \leftarrow \neg b,\; b \leftarrow \neg a\} \cup \{c \leftarrow b,\; -c \leftarrow b\}$$

induces

$$P^- = \left\{ \begin{array}{rclcrclcrcl} a & \leftarrow & a, -a & & a & \leftarrow & b, -b & & a & \leftarrow & c, -c \\ -a & \leftarrow & a, -a & & -a & \leftarrow & b, -b & & -a & \leftarrow & c, -c \\ b & \leftarrow & a, -a & & b & \leftarrow & b, -b & & b & \leftarrow & c, -c \\ -b & \leftarrow & a, -a & & -b & \leftarrow & b, -b & & -b & \leftarrow & c, -c \\ c & \leftarrow & a, -a & & c & \leftarrow & b, -b & & c & \leftarrow & c, -c \\ -c & \leftarrow & a, -a & & -c & \leftarrow & b, -b & & -c & \leftarrow & c, -c \end{array} \right\}$$

- The stable models of P are given by the ones of $P \cup P^-$, viz $\{a\}$

Potassco

An example

- The program

$$P = \{a \leftarrow \neg b,\ b \leftarrow \neg a\} \cup \{c \leftarrow b,\ -c \leftarrow b\}$$

induces

$$P^- = \left\{ \begin{array}{rcl rcl rcl} a & \leftarrow & a, -a & a & \leftarrow & b, -b & a & \leftarrow & c, -c \\ -a & \leftarrow & a, -a & -a & \leftarrow & b, -b & -a & \leftarrow & c, -c \\ b & \leftarrow & a, -a & b & \leftarrow & b, -b & b & \leftarrow & c, -c \\ -b & \leftarrow & a, -a & -b & \leftarrow & b, -b & -b & \leftarrow & c, -c \\ c & \leftarrow & a, -a & c & \leftarrow & b, -b & c & \leftarrow & c, -c \\ -c & \leftarrow & a, -a & -c & \leftarrow & b, -b & -c & \leftarrow & c, -c \end{array} \right\}$$

- The stable models of P are given by the ones of $P \cup P^-$, viz $\{a\}$

Properties

- The only inconsistent stable “model” is $X = \mathcal{A} \cup \overline{\mathcal{A}}$

  Strictly speaking, an inconsistemt set like $\mathcal{A} \cup \overline{\mathcal{A}}$ is not a model

- For a logic program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, exactly one of the following two cases applies:
  1. All stable models of P are consistent or
  2. $X = \mathcal{A} \cup \overline{\mathcal{A}}$ is the only stable model of P

Potassco

Properties

- The only inconsistent stable "model" is $X = \mathcal{A} \cup \overline{\mathcal{A}}$
  Strictly speaking, an inconsistemt set like $\mathcal{A} \cup \overline{\mathcal{A}}$ is not a model
- For a logic program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, exactly one of the following two cases applies:
  1. All stable models of P are consistent or
  2. $X = \mathcal{A} \cup \overline{\mathcal{A}}$ is the only stable model of P

Properties

- The only inconsistent stable "model" is $X = \mathcal{A} \cup \overline{\mathcal{A}}$

  Strictly speaking, an inconsistemt set like $\mathcal{A} \cup \overline{\mathcal{A}}$ is not a model
- For a logic program P over $\mathcal{A} \cup \overline{\mathcal{A}}$, exactly one of the following two cases applies:
  1. All stable models of P are consistent or
  2. $X = \mathcal{A} \cup \overline{\mathcal{A}}$ is the only stable model of P

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train,\ -train \leftarrow\}$
  - stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train,\ -train \leftarrow,\ -cross \leftarrow\}$
  - stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train,\ -train \leftarrow \neg train\}$
  - stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train,\ -train \leftarrow \neg train,\ -cross \leftarrow\}$
  - no stable model

Potassco

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train,\ -train \leftarrow\}$
  - stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train,\ -train \leftarrow,\ -cross \leftarrow\}$
  - stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train,\ -train \leftarrow \neg train\}$
  - stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train,\ -train \leftarrow \neg train,\ -cross \leftarrow\}$
  - no stable model

Potassco

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train, \ -train \leftarrow\}$
  - stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train, \ -train \leftarrow, \ -cross \leftarrow\}$
  - stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train, \ -train \leftarrow \neg train\}$
  - stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train, \ -train \leftarrow \neg train, \ -cross \leftarrow\}$
  - no stable model

Potassco

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train, \ -train \leftarrow\}$
  - stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train, \ -train \leftarrow, \ -cross \leftarrow\}$
  - stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train, \ -train \leftarrow \neg train\}$
  - stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train, \ -train \leftarrow \neg train, \ -cross \leftarrow\}$
  - no stable model

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train, \; -train \leftarrow\}$
  - stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train, \; -train \leftarrow, \; -cross \leftarrow\}$
  stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train, \; -train \leftarrow \neg train\}$
  stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train, \; -train \leftarrow \neg train, \; -cross \leftarrow\}$
  no stable model

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train,\ -train \leftarrow\}$
  - stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train,\ -train \leftarrow,\ -cross \leftarrow\}$
  - stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train,\ -train \leftarrow \neg train\}$
  - stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train,\ -train \leftarrow \neg train,\ -cross \leftarrow\}$
  - no stable model

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train,\ -train \leftarrow\}$
  stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train,\ -train \leftarrow,\ -cross \leftarrow\}$
  - stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train,\ -train \leftarrow \neg train\}$
  stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train,\ -train \leftarrow \neg train,\ -cross \leftarrow\}$
  no stable model

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train,\ -train \leftarrow\}$
  - stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train,\ -train \leftarrow,\ -cross \leftarrow\}$
  - stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train,\ -train \leftarrow \neg train\}$
  - stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train,\ -train \leftarrow \neg train,\ -cross \leftarrow\}$
  - no stable model

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train, \ -train \leftarrow\}$
  stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train, \ -train \leftarrow, \ -cross \leftarrow\}$
  stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train, \ -train \leftarrow \neg train\}$
  - stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train, \ -train \leftarrow \neg train, \ -cross \leftarrow\}$
  no stable model

Potassco

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train, \ -train \leftarrow\}$
  stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train, \ -train \leftarrow, \ -cross \leftarrow\}$
  stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train, \ -train \leftarrow \neg train\}$
  - stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train, \ -train \leftarrow \neg train, \ -cross \leftarrow\}$
  no stable model

Potassco

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train, \ -train \leftarrow\}$
  stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train, \ -train \leftarrow, \ -cross \leftarrow\}$
  stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train, \ -train \leftarrow \neg train\}$
  stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train, \ -train \leftarrow \neg train, \ -cross \leftarrow\}$
  - no stable model

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train,\ -train \leftarrow\}$
  stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train,\ -train \leftarrow,\ -cross \leftarrow\}$
  stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train,\ -train \leftarrow \neg train\}$
  stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train,\ -train \leftarrow \neg train,\ -cross \leftarrow\}$
  - no stable model

Train spotting

- $P_1 = \{cross \leftarrow \neg train\}$
  - stable model: $\{cross\}$
- $P_2 = \{cross \leftarrow -train\}$
  - stable model: $\emptyset$
- $P_3 = \{cross \leftarrow -train, \ -train \leftarrow\}$
  - stable model: $\{cross, -train\}$
- $P_4 = \{cross \leftarrow -train, \ -train \leftarrow, \ -cross \leftarrow\}$
  - stable model: $\{cross, -cross, train, -train\}$
- $P_5 = \{cross \leftarrow -train, \ -train \leftarrow \neg train\}$
  - stable model: $\{cross, -train\}$
- $P_6 = \{cross \leftarrow -train, \ -train \leftarrow \neg train, \ -cross \leftarrow\}$
  - no stable model

Default negation in rule heads

- We consider logic programs with default negation in rule heads
- Given an alphabet $\mathcal{A}$ of atoms, let $\widetilde{\mathcal{A}} = \{\widetilde{a} \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \widetilde{\mathcal{A}} = \emptyset$
- Given a program P over $\mathcal{A}$, consider the program

$$\begin{array}{rcl} \widetilde{P} & = & \{r \in P \mid h(r) \neq \neg a\} \\ & & \cup\ \{\leftarrow B(r) \cup \{\neg\widetilde{a}\} \mid r \in P \text{ and } h(r) = \neg a\} \\ & & \cup\ \{\widetilde{a} \leftarrow \neg a \mid r \in P \text{ and } h(r) = \neg a\} \end{array}$$

- A set X of atoms is a stable model of a program P (with default negation in rule heads) over $\mathcal{A}$, if $X = Y \cap \mathcal{A}$ for some stable model Y of $\widetilde{P}$ over $\mathcal{A} \cup \widetilde{\mathcal{A}}$

Default negation in rule heads

- We consider logic programs with default negation in rule heads
- Given an alphabet $\mathcal{A}$ of atoms, let $\widetilde{\mathcal{A}} = \{\widetilde{a} \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \widetilde{\mathcal{A}} = \emptyset$
- Given a program P over $\mathcal{A}$, consider the program

$$
\begin{aligned}
\widetilde{P} \quad = \quad & \{r \in P \mid h(r) \neq \neg a\} \\
\cup\ & \{\leftarrow B(r) \cup \{\neg \widetilde{a}\} \mid r \in P \text{ and } h(r) = \neg a\} \\
\cup\ & \{\widetilde{a} \leftarrow \neg a \mid r \in P \text{ and } h(r) = \neg a\}
\end{aligned}
$$

- A set X of atoms is a stable model of a program P (with default negation in rule heads) over $\mathcal{A}$,
if $X = Y \cap \mathcal{A}$ for some stable model Y of $\widetilde{P}$ over $\mathcal{A} \cup \widetilde{\mathcal{A}}$

Default negation in rule heads

- We consider logic programs with default negation in rule heads
- Given an alphabet $\mathcal{A}$ of atoms, let $\widetilde{\mathcal{A}} = \{\widetilde{a} \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \widetilde{\mathcal{A}} = \emptyset$
- Given a program P over $\mathcal{A}$, consider the program

$$\begin{aligned}\widetilde{P} \quad = \quad & \{r \in P \mid h(r) \neq \neg a\} \\ \cup \; & \{\leftarrow B(r) \cup \{\neg\widetilde{a}\} \mid r \in P \text{ and } h(r) = \neg a\} \\ \cup \; & \{\widetilde{a} \leftarrow \neg a \mid r \in P \text{ and } h(r) = \neg a\}\end{aligned}$$

- A set X of atoms is a stable model of a program P (with default negation in rule heads) over $\mathcal{A}$,
if $X = Y \cap \mathcal{A}$ for some stable model Y of $\widetilde{P}$ over $\mathcal{A} \cup \widetilde{\mathcal{A}}$

Potassco

Default negation in rule heads

- We consider logic programs with default negation in rule heads
- Given an alphabet $\mathcal{A}$ of atoms, let $\widetilde{\mathcal{A}} = \{\widetilde{a} \mid a \in \mathcal{A}\}$ such that $\mathcal{A} \cap \widetilde{\mathcal{A}} = \emptyset$
- Given a program P over $\mathcal{A}$, consider the program

$$\begin{array}{rcl}\widetilde{P} & = & \{r \in P \mid h(r) \neq \neg a\} \\ & & \cup\ \{\leftarrow B(r) \cup \{\neg\widetilde{a}\} \mid r \in P \text{ and } h(r) = \neg a\} \\ & & \cup\ \{\widetilde{a} \leftarrow \neg a \mid r \in P \text{ and } h(r) = \neg a\}\end{array}$$

- A set X of atoms is a stable model of a program P (with default negation in rule heads) over $\mathcal{A}$,
  if $X = Y \cap \mathcal{A}$ for some stable model Y of $\widetilde{P}$ over $\mathcal{A} \cup \widetilde{\mathcal{A}}$

Potassco

Outline

Disjunctive logic programs

- A disjunctive rule, r, is of the form

$$a_1 \,;\dots;\, a_m \leftarrow a_{m+1},\dots,a_n,\neg a_{n+1},\dots,\neg a_o$$

where $0 \leq m \leq n \leq o$ and each a_i is an atom for $0 \leq i \leq o$

- A disjunctive logic program is a finite set of disjunctive rules
- Notation

$$\begin{array}{rcl} H(r) & = & \{a_1,\dots,a_m\} \\ B(r) & = & \{a_{m+1},\dots,a_n,\neg a_{n+1},\dots,\neg a_o\} \\ B(r)^+ & = & \{a_{m+1},\dots,a_n\} \\ B(r)^- & = & \{a_{n+1},\dots,a_o\} \\ A(P) & = & \bigcup_{r\in P}\left(H(r)\cup B(r)^+\cup B(r)^-\right) \\ B(P) & = & \{B(r) \mid r \in P\} \end{array}$$

- A program is called positive if $B(r)^- = \emptyset$ for all its rules

Disjunctive logic programs

- A disjunctive rule, r, is of the form

$$a_1 \,;\ldots\,; a_m \leftarrow a_{m+1},\ldots,a_n,\neg a_{n+1},\ldots,\neg a_o$$

  where $0 \leq m \leq n \leq o$ and each a_i is an atom for $0 \leq i \leq o$
- A disjunctive logic program is a finite set of disjunctive rules
- Notation

$$\begin{array}{rcl} H(r) & = & \{a_1,\ldots,a_m\} \\ B(r) & = & \{a_{m+1},\ldots,a_n,\neg a_{n+1},\ldots,\neg a_o\} \\ B(r)^+ & = & \{a_{m+1},\ldots,a_n\} \\ B(r)^- & = & \{a_{n+1},\ldots,a_o\} \\ A(P) & = & \bigcup_{r\in P}\left(H(r)\cup B(r)^+\cup B(r)^-\right) \\ B(P) & = & \{B(r) \mid r \in P\} \end{array}$$

- A program is called positive if $B(r)^- = \emptyset$ for all its rules

Disjunctive logic programs

- A disjunctive rule, r, is of the form

$$a_1 ; \dots ; a_m \leftarrow a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o$$

where $0 \leq m \leq n \leq o$ and each a_i is an atom for $0 \leq i \leq o$

- A disjunctive logic program is a finite set of disjunctive rules
- Notation

$$\begin{array}{rcl}
H(r) & = & \{a_1, \dots, a_m\} \\
B(r) & = & \{a_{m+1}, \dots, a_n, \neg a_{n+1}, \dots, \neg a_o\} \\
B(r)^+ & = & \{a_{m+1}, \dots, a_n\} \\
B(r)^- & = & \{a_{n+1}, \dots, a_o\} \\
A(P) & = & \bigcup_{r \in P} \left(H(r) \cup B(r)^+ \cup B(r)^-\right) \\
B(P) & = & \{B(r) \mid r \in P\}
\end{array}$$

- A program is called positive if $B(r)^- = \emptyset$ for all its rules

Stable models

- Positive programs
  - A set X of atoms is closed under a positive program P iff for any $r \in P$, $H(r) \cap X \neq \emptyset$ whenever $B(r)^+ \subseteq X$
    - X corresponds to a model of P (seen as a formula)
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program P is denoted by $\min_\subseteq(P)$
    - $\min_\subseteq(P)$ corresponds to the $\subseteq$-minimal models of P (ditto)
- Disjunctive programs
  - The reduct, P^X, of a disjunctive program P relative to a set X of atoms is defined by

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

  - A set X of atoms is a stable model of a disjunctive program P, if $X \in \min_\subseteq(P^X)$

Stable models

- Positive programs
  - A set X of atoms is closed under a positive program P iff for any $r \in P$, $H(r) \cap X \neq \emptyset$ whenever $B(r)^+ \subseteq X$
    - X corresponds to a model of P (seen as a formula)
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program P is denoted by $\min_\subseteq(P)$
    - $\min_\subseteq(P)$ corresponds to the $\subseteq$-minimal models of P (ditto)
- Disjunctive programs
  - The reduct, P^X, of a disjunctive program P relative to a set X of atoms is defined by

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

  - A set X of atoms is a stable model of a disjunctive program P, if $X \in \min_\subseteq(P^X)$

Stable models

- Positive programs
  - A set X of atoms is closed under a positive program P iff for any $r \in P$, $H(r) \cap X \neq \emptyset$ whenever $B(r)^+ \subseteq X$
    - X corresponds to a model of P (seen as a formula)
  - The set of all $\subseteq$-minimal sets of atoms being closed under a positive program P is denoted by $\min_\subseteq(P)$
    - $\min_\subseteq(P)$ corresponds to the $\subseteq$-minimal models of P (ditto)
- Disjunctive programs
  - The reduct, P^X, of a disjunctive program P relative to a set X of atoms is defined by

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

  - A set X of atoms is a stable model of a disjunctive program P, if $X \in \min_\subseteq(P^X)$

A "positive" example

$$P = \left\{ \begin{array}{lcr} a & \leftarrow & \\ b\,;c & \leftarrow & a \end{array} \right\}$$

- The sets $\{a, b\}$, $\{a, c\}$, and $\{a, b, c\}$ are closed under P
- We have $\min_{\subseteq}(P) = \{\{a, b\}, \{a, c\}\}$

Potassco

A "positive" example

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

- The sets $\{a, b\}$, $\{a, c\}$, and $\{a, b, c\}$ are closed under P
- We have $\min_{\subseteq}(P) = \{\{a, b\}, \{a, c\}\}$

A “positive” example

$$P = \left\{ \begin{array}{lcr} a & \leftarrow & \\ b\,;c & \leftarrow & a \end{array} \right\}$$

- The sets $\{a,b\}$, $\{a,c\}$, and $\{a,b,c\}$ are closed under P
- We have $\min_{\subseteq}(P) = \{\{a,b\},\{a,c\}\}$

Graph coloring (reloaded)

```
node(1..6).

edge(1,(2;3;4)).  edge(2,(4;5;6)).  edge(3,(1;4;5)).
edge(4,(1;2)).    edge(5,(3;4;6)).  edge(6,(2;3;5)).


assign(X,r) ; assign(X,b) ; assign(X,g) :- node(X).


:- edge(X,Y), assign(X,C), assign(Y,C).
```

Graph coloring (reloaded)

```
node(1..6).

edge(1,(2;3;4)).  edge(2,(4;5;6)).  edge(3,(1;4;5)).
edge(4,(1;2)).    edge(5,(3;4;6)).  edge(6,(2;3;5)).

color(r).   color(b).   color(g).

assign(X,C) : color(C) :- node(X).

:- edge(X,Y), assign(X,C), assign(Y,C).
```

More Examples

- $P_1 = \{a ; b ; c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a ; b ; c \leftarrow \ , \ \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a ; b ; c \leftarrow \ , \ \leftarrow a \ , \ b \leftarrow c \ , \ c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a ; b \leftarrow c \ , \ b \leftarrow \neg a, \neg c \ , \ a ; c \leftarrow \neg b\}$
  - stable models $\{a\}$ and $\{b\}$

Potassco

More Examples

- $P_1 = \{a \,;\, b \,;\, c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a \,;\, b \,;\, c \leftarrow \;,\; \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a \,;\, b \,;\, c \leftarrow \;,\; \leftarrow a \;,\; b \leftarrow c \;,\; c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a \,;\, b \leftarrow c \;,\; b \leftarrow \neg a, \neg c \;,\; a \,;\, c \leftarrow \neg b\}$
  - stable models $\{a\}$ and $\{b\}$

More Examples

- $P_1 = \{a \,;\, b \,;\, c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a \,;\, b \,;\, c \leftarrow \;,\; \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a \,;\, b \,;\, c \leftarrow \;,\; \leftarrow a \;,\; b \leftarrow c \;,\; c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a \,;\, b \leftarrow c \;,\; b \leftarrow \neg a, \neg c \;,\; a \,;\, c \leftarrow \neg b\}$
  - stable models $\{a\}$ and $\{b\}$

Potassco

More Examples

- $P_1 = \{a\,;b\,;c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow \neg a, \neg c\ ,\ a\,;c \leftarrow \neg b\}$
  - stable models $\{a\}$ and $\{b\}$

Potassco

More Examples

- $P_1 = \{a\,;b\,;c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow \neg a, \neg c\ ,\ a\,;c \leftarrow \neg b\}$
  - stable models $\{a\}$ and $\{b\}$

Potassco

More Examples

- $P_1 = \{a \,;\, b \,;\, c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a \,;\, b \,;\, c \leftarrow \;,\; \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a \,;\, b \,;\, c \leftarrow \;,\; \leftarrow a \;,\; b \leftarrow c \;,\; c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a \,;\, b \leftarrow c \;,\; b \leftarrow \neg a, \neg c \;,\; a \,;\, c \leftarrow \neg b\}$
  - stable models $\{a\}$ and $\{b\}$

More Examples

- $P_1 = \{a\,;b\,;c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow \neg a, \neg c\ ,\ a\,;c \leftarrow \neg b\}$
  - stable models $\{a\}$ and $\{b\}$

Potassco

More Examples

- $P_1 = \{a \,;\, b \,;\, c \leftarrow\}$
    - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a \,;\, b \,;\, c \leftarrow \;,\; \leftarrow a\}$
    - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a \,;\, b \,;\, c \leftarrow \;,\; \leftarrow a \;,\; b \leftarrow c \;,\; c \leftarrow b\}$
    - stable model $\{b, c\}$
- $P_4 = \{a \,;\, b \leftarrow c \;,\; b \leftarrow \neg a, \neg c \;,\; a \,;\, c \leftarrow \neg b\}$
    - stable models $\{a\}$ and $\{b\}$

More Examples

- $P_1 = \{a\,;b\,;c \leftarrow\}$
  - stable models $\{a\}$, $\{b\}$, and $\{c\}$
- $P_2 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\}$
  - stable models $\{b\}$ and $\{c\}$
- $P_3 = \{a\,;b\,;c \leftarrow\ ,\ \leftarrow a\ ,\ b \leftarrow c\ ,\ c \leftarrow b\}$
  - stable model $\{b, c\}$
- $P_4 = \{a\,;b \leftarrow c\ ,\ b \leftarrow \neg a, \neg c\ ,\ a\,;c \leftarrow \neg b\}$
  - stable models $\{a\}$ and $\{b\}$

Some properties

- A disjunctive logic program may have zero, one, or multiple stable models
- If X is a stable model of a disjunctive logic program P, then X is a model of P (seen as a formula)
- If X and Y are stable models of a disjunctive logic program P, then $X \not\subset Y$
- If $a \in X$ for some stable model X of a disjunctive logic program P, then there is a rule $r \in P$ such that $B(r)^+ \subseteq X$, $B(r)^- \cap X = \emptyset$, and $H(r) \cap X = \{a\}$

Some properties

- A disjunctive logic program may have zero, one, or multiple stable models
- If X is a stable model of a disjunctive logic program P, then X is a model of P (seen as a formula)
- If X and Y are stable models of a disjunctive logic program P, then $X \not\subset Y$
- If $a \in X$ for some stable model X of a disjunctive logic program P, then there is a rule $r \in P$ such that $B(r)^+ \subseteq X$, $B(r)^- \cap X = \emptyset$, and $H(r) \cap X = \{a\}$

Potassco

An example with variables

$$P = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(X) \,;\, c(Y) & \leftarrow & a(X,Y), \neg c(Y) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1) \,;\, c(1) & \leftarrow & a(1,1), \neg c(1) \\ b(1) \,;\, c(2) & \leftarrow & a(1,2), \neg c(2) \\ b(2) \,;\, c(1) & \leftarrow & a(2,1), \neg c(1) \\ b(2) \,;\, c(2) & \leftarrow & a(2,2), \neg c(2) \end{array} \right\}$$

For every stable model X of P, we have

- $a(1,2) \in X$ and
- $\{a(1,1), a(2,1), a(2,2)\} \cap X = \emptyset$

An example with variables

$$P = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(X)\,;c(Y) & \leftarrow & a(X,Y), \neg c(Y) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), \neg c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), \neg c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), \neg c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), \neg c(2) \end{array} \right\}$$

For every stable model X of P, we have

- $a(1,2) \in X$ and
- $\{a(1,1), a(2,1), a(2,2)\} \cap X = \emptyset$

An example with variables

$$P = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(X) \,;\, c(Y) & \leftarrow & a(X,Y), \neg c(Y) \end{array} \right\}$$

$$ground(P) = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1) \,;\, c(1) & \leftarrow & a(1,1), \neg c(1) \\ b(1) \,;\, c(2) & \leftarrow & a(1,2), \neg c(2) \\ b(2) \,;\, c(1) & \leftarrow & a(2,1), \neg c(1) \\ b(2) \,;\, c(2) & \leftarrow & a(2,2), \neg c(2) \end{array} \right\}$$

For every stable model X of P, we have

- $a(1,2) \in X$ and
- $\{a(1,1), a(2,1), a(2,2)\} \cap X = \emptyset$

Potassco

An example with variables

$$ground(P)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), \neg c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), \neg c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), \neg c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), \neg c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2), b(1)\}$
- We get $\min_{\subseteq}(ground(P)^X) = \{\ \{a(1,2), b(1)\},\ \{a(1,2), c(2)\}\ \}$
- X is a stable model of P because $X \in \min_{\subseteq}(ground(P)^X)$

Potassco

An example with variables

$$ground(P)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), \neg c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), \neg c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), \neg c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), \neg c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2), b(1)\}$
- We get $\min_{\subseteq}(ground(P)^X) = \{\ \{a(1,2), b(1)\},\ \{a(1,2), c(2)\}\ \}$
- X is a stable model of P because $X \in \min_{\subseteq}(ground(P)^X)$

An example with variables

$$
ground(P)^X = \left\{ \begin{array}{lcl}
a(1,2) & \leftarrow & \\
b(1) \,;\, c(1) & \leftarrow & a(1,1), \neg c(1) \\
b(1) \,;\, c(2) & \leftarrow & a(1,2), \neg c(2) \\
b(2) \,;\, c(1) & \leftarrow & a(2,1), \neg c(1) \\
b(2) \,;\, c(2) & \leftarrow & a(2,2), \neg c(2)
\end{array} \right\}
$$

- Consider $X = \{a(1,2), b(1)\}$
- We get $\min_{\subseteq}(ground(P)^X) = \{\ \{a(1,2), b(1)\},\ \{a(1,2), c(2)\}\ \}$
- X is a stable model of P because $X \in \min_{\subseteq}(ground(P)^X)$

Potassco

An example with variables

$$ground(P)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), \neg c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), \neg c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), \neg c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), \neg c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2), b(1)\}$
- We get $\min_{\subseteq}(ground(P)^X) = \{\ \{a(1,2), b(1)\},\ \{a(1,2), c(2)\}\ \}$
- X is a stable model of P because $X \in \min_{\subseteq}(ground(P)^X)$

An example with variables

$$
ground(P)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1) ; c(1) & \leftarrow & a(1,1), \neg c(1) \\ b(1) ; c(2) & \leftarrow & a(1,2), \neg c(2) \\ b(2) ; c(1) & \leftarrow & a(2,1), \neg c(1) \\ b(2) ; c(2) & \leftarrow & a(2,2), \neg c(2) \end{array} \right\}
$$

- Consider $X = \{a(1,2), b(1)\}$
- We get $\min_{\subseteq}(ground(P)^X) = \{\ \{a(1,2), b(1)\},\ \{a(1,2), c(2)\}\ \}$
- X is a stable model of P because $X \in \min_{\subseteq}(ground(P)^X)$

An example with variables

$$ground(P)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), \neg c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), \neg c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), \neg c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), \neg c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2), c(2)\}$
- We get $\min_{\subseteq}(ground(P)^X) = \{\ \{a(1,2)\}\ \}$
- X is no stable model of P because $X \notin \min_{\subseteq}(ground(P)^X)$

An example with variables

$$ground(P)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), \neg c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), \neg c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), \neg c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), \neg c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2), c(2)\}$
- We get $\min_\subseteq(ground(P)^X) = \{\ \{a(1,2)\}\ \}$
- X is no stable model of P because $X \notin \min_\subseteq(ground(P)^X)$

An example with variables

$$ground(P)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1) ; c(1) & \leftarrow & a(1,1), \neg c(1) \\ b(1) ; c(2) & \leftarrow & a(1,2), \neg c(2) \\ b(2) ; c(1) & \leftarrow & a(2,1), \neg c(1) \\ b(2) ; c(2) & \leftarrow & a(2,2), \neg c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2), c(2)\}$
- We get $\min_{\subseteq}(ground(P)^X) = \{ \{a(1,2)\} \}$
- X is no stable model of P because $X \notin \min_{\subseteq}(ground(P)^X)$

An example with variables

$$ground(P)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), \neg c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), \neg c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), \neg c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), \neg c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2), c(2)\}$
- We get $\min_\subseteq(ground(P)^X) = \{\ \{a(1,2)\}\ \}$
- X is no stable model of P because $X \notin \min_\subseteq(ground(P)^X)$

Potassco

An example with variables

$$ground(P)^X = \left\{ \begin{array}{lcl} a(1,2) & \leftarrow & \\ b(1)\,;c(1) & \leftarrow & a(1,1), \neg c(1) \\ b(1)\,;c(2) & \leftarrow & a(1,2), \neg c(2) \\ b(2)\,;c(1) & \leftarrow & a(2,1), \neg c(1) \\ b(2)\,;c(2) & \leftarrow & a(2,2), \neg c(2) \end{array} \right\}$$

- Consider $X = \{a(1,2), c(2)\}$
- We get $\min_{\subseteq}(ground(P)^X) = \{\ \{a(1,2)\}\ \}$
- X is no stable model of P because $X \notin \min_{\subseteq}(ground(P)^X)$

Potassco

Default negation in rule heads

- Consider disjunctive rules of the form

$$a_1 \,;\ldots\,; a_m \,; \neg a_{m+1} \,;\ldots\,; \neg a_n \leftarrow a_{n+1}, \ldots, a_o, \neg a_{o+1}, \ldots, \neg a_p$$

where $0 \leq m \leq n \leq o \leq p$ and each a_i is an atom for $0 \leq i \leq p$

- Given a program P over $\mathcal{A}$, consider the program

$$\begin{aligned}\widetilde{P} \quad = \quad & \{H(r)^+ \leftarrow B(r) \cup \{\neg\widetilde{a} \mid a \in H(r)^-\} \mid r \in P\} \\ & \cup\, \{\widetilde{a} \leftarrow \neg a \mid r \in P \text{ and } a \in H(r)^-\}\end{aligned}$$

- A set X of atoms is a stable model of a disjunctive program P (with default negation in rule heads) over $\mathcal{A}$, if $X = Y \cap \mathcal{A}$ for some stable model Y of $\widetilde{P}$ over $\mathcal{A} \cup \widetilde{\mathcal{A}}$

Default negation in rule heads

- Consider disjunctive rules of the form

$$a_1 ; \dots ; a_m ; \neg a_{m+1} ; \dots ; \neg a_n \leftarrow a_{n+1}, \dots, a_o, \neg a_{o+1}, \dots, \neg a_p$$

  where $0 \leq m \leq n \leq o \leq p$ and each a_i is an atom for $0 \leq i \leq p$
- Given a program P over $\mathcal{A}$, consider the program

$$\begin{aligned} \widetilde{P} \quad = \quad & \{H(r)^+ \leftarrow B(r) \cup \{\neg \widetilde{a} \mid a \in H(r)^-\} \mid r \in P\} \\ & \cup \{\widetilde{a} \leftarrow \neg a \mid r \in P \text{ and } a \in H(r)^-\} \end{aligned}$$

- A set X of atoms is a stable model of a disjunctive program P (with default negation in rule heads) over $\mathcal{A}$, if $X = Y \cap \mathcal{A}$ for some stable model Y of $\widetilde{P}$ over $\mathcal{A} \cup \widetilde{\mathcal{A}}$

Default negation in rule heads

- Consider disjunctive rules of the form

$$a_1 \,;\ldots;\, a_m \,;\, \neg a_{m+1} \,;\ldots;\, \neg a_n \leftarrow a_{n+1},\ldots,a_o,\neg a_{o+1},\ldots,\neg a_p$$

  where $0 \leq m \leq n \leq o \leq p$ and each a_i is an atom for $0 \leq i \leq p$
- Given a program P over $\mathcal{A}$, consider the program

$$\begin{aligned}\widetilde{P} \quad = \quad & \{H(r)^+ \leftarrow B(r) \cup \{\neg\widetilde{a} \mid a \in H(r)^-\} \mid r \in P\} \\ & \cup \{\widetilde{a} \leftarrow \neg a \mid r \in P \text{ and } a \in H(r)^-\}\end{aligned}$$

- A set X of atoms is a stable model of a disjunctive program P (with default negation in rule heads) over $\mathcal{A}$,
  if $X = Y \cap \mathcal{A}$ for some stable model Y of $\widetilde{P}$ over $\mathcal{A} \cup \widetilde{\mathcal{A}}$

Potassco

An example

- The program
$$P = \{a \,;\, \neg a \leftarrow\}$$
yields
$$\widetilde{P} = \{a \leftarrow \neg\widetilde{a}\} \cup \{\widetilde{a} \leftarrow \neg a\}$$
- $\widetilde{P}$ has two stable models, $\{a\}$ and $\{\widetilde{a}\}$
- This induces the stable models $\{a\}$ and $\emptyset$ of P

An example

- The program

$$P = \{a \,;\, \neg a \leftarrow\}$$

  yields

$$\widetilde{P} = \{a \leftarrow \neg\widetilde{a}\} \cup \{\widetilde{a} \leftarrow \neg a\}$$

- $\widetilde{P}$ has two stable models, $\{a\}$ and $\{\widetilde{a}\}$
- This induces the stable models $\{a\}$ and $\emptyset$ of P

Potassco

An example

- The program
$$P = \{a \,;\, \neg a \leftarrow\}$$
yields
$$\widetilde{P} = \{a \leftarrow \neg\widetilde{a}\} \cup \{\widetilde{a} \leftarrow \neg a\}$$
- $\widetilde{P}$ has two stable models, $\{a\}$ and $\{\widetilde{a}\}$
- This induces the stable models $\{a\}$ and $\emptyset$ of P

An example

- The program

$$P = \{a ; \neg a \leftarrow\}$$

  yields

$$\widetilde{P} = \{a \leftarrow \neg\widetilde{a}\} \cup \{\widetilde{a} \leftarrow \neg a\}$$

- $\widetilde{P}$ has two stable models, $\{a\}$ and $\{\widetilde{a}\}$
- This induces the stable models $\{a\}$ and $\emptyset$ of P

Outline

Propositional theories

- Formulas are formed from
  - atoms in $\mathcal{A}$
  - $\bot$

  using
  - conjunction ($\wedge$)
  - disjunction ($\vee$)
  - implication ($\rightarrow$)
- Notation

$$\begin{array}{rcl} \top & = & (\bot \rightarrow \bot) \\ \neg\phi & = & (\phi \rightarrow \bot) \end{array}$$

A propositional theory is a finite set of formulas

Potassco

Propositional theories

- Formulas are formed from
  - atoms in $\mathcal{A}$
  - $\bot$

  using
  - conjunction ($\wedge$)
  - disjunction ($\vee$)
  - implication ($\rightarrow$)
- Notation

$$\begin{array}{rcl} \top & = & (\bot \rightarrow \bot) \\ \neg\phi & = & (\phi \rightarrow \bot) \end{array}$$

- A propositional theory is a finite set of formulas

Propositional theories

- Formulas are formed from
  - atoms in $\mathcal{A}$
  - $\bot$

  using
  - conjunction ($\wedge$)
  - disjunction ($\vee$)
  - implication ($\rightarrow$)
- Notation

$$
\begin{array}{rcl}
\top & = & (\bot \rightarrow \bot) \\
\neg\phi & = & (\phi \rightarrow \bot)
\end{array}
$$

- A propositional theory is a finite set of formulas

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:

$$\begin{array}{ll} \phi^X = \bot & \text{if } X \not\models \phi \\ \phi^X = \phi & \text{if } \phi \in X \\ \phi^X = (\psi^X \circ \varphi^X) & \text{if } X \models \phi \text{ and } \phi = (\psi \circ \varphi) \text{ for } \circ \in \{\wedge, \vee, \rightarrow\} \end{array}$$

If $\phi = \neg\psi = (\psi \rightarrow \bot)$,
then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise

The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:

$$\begin{array}{ll} \phi^X = \bot & \text{if } X \not\models \phi \\ \phi^X = \phi & \text{if } \phi \in X \\ \phi^X = (\psi^X \circ \varphi^X) & \text{if } X \models \phi \text{ and } \phi = (\psi \circ \varphi) \text{ for } \circ \in \{\wedge, \vee, \rightarrow\} \end{array}$$

If $\phi = \neg\psi = (\psi \rightarrow \bot)$,
then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise

The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:
  - $\phi^X = \bot$ if $X \not\models \phi$
  - $\phi^X = \phi$ if $\phi \in X$
  - $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
  - If $\phi = \neg\psi = (\psi \rightarrow \bot)$,
    then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise
- The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:
  - $\phi^X = \bot$ if $X \not\models \phi$
  - $\phi^X = \phi$ if $\phi \in X$
  - $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
  - If $\phi = \neg\psi = (\psi \rightarrow \bot)$,
    then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise
- The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:
  - $\phi^X = \bot$ if $X \not\models \phi$
  - $\phi^X = \phi$ if $\phi \in X$
  - $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
  - If $\phi = \neg\psi = (\psi \rightarrow \bot)$,
    then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise
- The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:
  - $\phi^X = \bot$ if $X \not\models \phi$
  - $\phi^X = \phi$ if $\phi \in X$
  - $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
  - If $\phi = \neg\psi = (\psi \rightarrow \bot)$, then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise
- The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Reduct

- The satisfaction relation $X \models \phi$ between a set X of atoms and a (set of) formula(s) ϕ is defined as in propositional logic
- The reduct, ϕ^X, of a formula ϕ relative to a set X of atoms is defined recursively as follows:
  - $\phi^X = \bot$ if $X \not\models \phi$
  - $\phi^X = \phi$ if $\phi \in X$
  - $\phi^X = (\psi^X \circ \varphi^X)$ if $X \models \phi$ and $\phi = (\psi \circ \varphi)$ for $\circ \in \{\wedge, \vee, \rightarrow\}$
  - If $\phi = \neg\psi = (\psi \rightarrow \bot)$,
    then $\phi^X = (\bot \rightarrow \bot) = \top$, if $X \not\models \psi$, and $\phi^X = \bot$, otherwise
- The reduct, Φ^X, of a propositional theory Φ relative to a set X of atoms is defined as $\Phi^X = \{\phi^X \mid \phi \in \Phi\}$

Potassco

Stable models

- A set X of atoms satisfies a propositional theory Φ, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$
- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory Φ is denoted by $\min_{\subseteq}(\Phi)$
- A set X of atoms is a stable model of a propositional theory Φ, if $X \in \min_{\subseteq}(\Phi^X)$
- If X is a stable model of Φ, then
  - $X \models \Phi$ and
  - $\min_{\subseteq}(\Phi^X) = \{X\}$
- Note In general, this does not imply $X \in \min_{\subseteq}(\Phi)$!

Potassco

Stable models

- A set X of atoms satisfies a propositional theory Φ, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$
- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory Φ is denoted by $\min_{\subseteq}(\Phi)$
- A set X of atoms is a stable model of a propositional theory Φ, if $X \in \min_{\subseteq}(\Phi^X)$
- If X is a stable model of Φ, then
  - $X \models \Phi$ and
  - $\min_{\subseteq}(\Phi^X) = \{X\}$
- Note In general, this does not imply $X \in \min_{\subseteq}(\Phi)$!

Stable models

- A set X of atoms satisfies a propositional theory Φ, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$
- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory Φ is denoted by $\min_{\subseteq}(\Phi)$
- A set X of atoms is a stable model of a propositional theory Φ, if $X \in \min_{\subseteq}(\Phi^X)$
- If X is a stable model of Φ, then
  - $X \models \Phi$ and
  - $\min_{\subseteq}(\Phi^X) = \{X\}$
- Note In general, this does not imply $X \in \min_{\subseteq}(\Phi)$!

Stable models

- A set X of atoms satisfies a propositional theory Φ, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$
- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory Φ is denoted by $\min_\subseteq(\Phi)$
- A set X of atoms is a stable model of a propositional theory Φ, if $X \in \min_\subseteq(\Phi^X)$
- If X is a stable model of Φ, then
  - $X \models \Phi$ and
  - $\min_\subseteq(\Phi^X) = \{X\}$
- Note In general, this does not imply $X \in \min_\subseteq(\Phi)$!

Stable models

- A set X of atoms satisfies a propositional theory Φ, written $X \models \Phi$, if $X \models \phi$ for each $\phi \in \Phi$
- The set of all $\subseteq$-minimal sets of atoms satisfying a propositional theory Φ is denoted by $\min_{\subseteq}(\Phi)$
- A set X of atoms is a stable model of a propositional theory Φ, if $X \in \min_{\subseteq}(\Phi^X)$
- If X is a stable model of Φ, then
  - $X \models \Phi$ and
  - $\min_{\subseteq}(\Phi^X) = \{X\}$
- Note In general, this does not imply $X \in \min_{\subseteq}(\Phi)$!

Two examples

- $\Phi_1 = \{p \vee (p \to (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \to (q \wedge r))\}$ and $\min_\subseteq(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$
    For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \to \bot)\}$ and $\min_\subseteq(\Phi_1^{\emptyset}) = \{\emptyset\}$

- $\Phi_2 = \{p \vee (\neg p \to (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_\subseteq(\Phi_2^{\emptyset}) = \emptyset$
    For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \to \bot)\}$ and $\min_\subseteq(\Phi_2^{\{p\}}) = \{\emptyset\}$
    For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \to (q \wedge r))\}$ and $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$

- $\Phi_2 = \{p \vee (\neg p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✗
  - For $X = \emptyset$, we get $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_1^{\emptyset}) = \{\emptyset\}$
- $\Phi_2 = \{p \vee (\neg p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_\subseteq(\Phi_2^{\emptyset}) = \emptyset$
  - For $X = \{p\}$, we get $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_2^{\{p\}}) = \{\emptyset\}$
  - For $X = \{q, r\}$, we get $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Potassco

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔
- $\Phi_2 = \{p \vee (\neg p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Potassco

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔

- $\Phi_2 = \{p \vee (\neg p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_\subseteq(\Phi_2^{\emptyset}) = \emptyset$
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_2^{\{p\}}) = \{\emptyset\}$
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔
- $\Phi_2 = \{p \vee (\neg p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_\subseteq(\Phi_2^{\emptyset}) = \emptyset$
    For $X = \{p\}$, we get $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_2^{\{p\}}) = \{\emptyset\}$
    For $X = \{q, r\}$, we get $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔

- $\Phi_2 = \{p \vee (\neg p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$ ✘
    For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$
    For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔

- $\Phi_2 = \{p \vee (\neg p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_\subseteq(\Phi_2^{\emptyset}) = \emptyset$ ✘
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_2^{\{p\}}) = \{\emptyset\}$
    For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔

- $\Phi_2 = \{p \vee (\neg p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_\subseteq(\Phi_2^{\emptyset}) = \emptyset$ ✘
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_\subseteq(\Phi_2^{\{p\}}) = \{\emptyset\}$ ✘
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_\subseteq(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔

- $\Phi_2 = \{p \vee (\neg p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$ ✘
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$ ✘
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔
- $\Phi_2 = \{p \vee (\neg p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$ ✘
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$ ✘
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$ ✔

Two examples

- $\Phi_1 = \{p \vee (p \rightarrow (q \wedge r))\}$
  - For $X = \{p, q, r\}$, we get
    $\Phi_1^{\{p,q,r\}} = \{p \vee (p \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_1^{\{p,q,r\}}) = \{\emptyset\}$ ✘
  - For $X = \emptyset$, we get
    $\Phi_1^{\emptyset} = \{\bot \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_1^{\emptyset}) = \{\emptyset\}$ ✔

- $\Phi_2 = \{p \vee (\neg p \rightarrow (q \wedge r))\}$
  - For $X = \emptyset$, we get
    $\Phi_2^{\emptyset} = \{\bot\}$ and $\min_{\subseteq}(\Phi_2^{\emptyset}) = \emptyset$ ✘
  - For $X = \{p\}$, we get
    $\Phi_2^{\{p\}} = \{p \vee (\bot \rightarrow \bot)\}$ and $\min_{\subseteq}(\Phi_2^{\{p\}}) = \{\emptyset\}$ ✘
  - For $X = \{q, r\}$, we get
    $\Phi_2^{\{q,r\}} = \{\bot \vee (\top \rightarrow (q \wedge r))\}$ and $\min_{\subseteq}(\Phi_2^{\{q,r\}}) = \{\{q, r\}\}$ ✔

Relationship to logic programs

- The translation, $\tau[(\phi \leftarrow \psi)]$, of a rule $(\phi \leftarrow \psi)$ is defined as follows:
  - $\tau[(\phi \leftarrow \psi)] = (\tau[\psi] \rightarrow \tau[\phi])$
  - $\tau[\bot] = \bot$
  - $\tau[\top] = \top$
  - $\tau[\phi] = \phi$ if ϕ is an atom
  - $\tau[\neg\phi] = \neg\tau[\phi]$
  - $\tau[(\phi, \psi)] = (\tau[\phi] \wedge \tau[\psi])$
  - $\tau[(\phi; \psi)] = (\tau[\phi] \vee \tau[\psi])$

- The translation of a logic program P is $\tau[P] = \{\tau[r] \mid r \in P\}$

- Given a logic program P and a set X of atoms,
  X is a stable model of P iff X is a stable model of $\tau[P]$

Relationship to logic programs

- The translation, $\tau[(\phi \leftarrow \psi)]$, of a rule $(\phi \leftarrow \psi)$ is defined as follows:
  - $\tau[(\phi \leftarrow \psi)] = (\tau[\psi] \rightarrow \tau[\phi])$
  - $\tau[\bot] = \bot$
  - $\tau[\top] = \top$
  - $\tau[\phi] = \phi$ if ϕ is an atom
  - $\tau[\neg\phi] = \neg\tau[\phi]$
  - $\tau[(\phi, \psi)] = (\tau[\phi] \wedge \tau[\psi])$
  - $\tau[(\phi; \psi)] = (\tau[\phi] \vee \tau[\psi])$
- The translation of a logic program P is $\tau[P] = \{\tau[r] \mid r \in P\}$
- Given a logic program P and a set X of atoms,
  X is a stable model of P iff X is a stable model of $\tau[P]$

Potassco

Relationship to logic programs

- The translation, $\tau[(\phi \leftarrow \psi)]$, of a rule $(\phi \leftarrow \psi)$ is defined as follows:
  - $\tau[(\phi \leftarrow \psi)] = (\tau[\psi] \rightarrow \tau[\phi])$
  - $\tau[\bot] = \bot$
  - $\tau[\top] = \top$
  - $\tau[\phi] = \phi$ if ϕ is an atom
  - $\tau[\neg\phi] = \neg\tau[\phi]$
  - $\tau[(\phi, \psi)] = (\tau[\phi] \wedge \tau[\psi])$
  - $\tau[(\phi; \psi)] = (\tau[\phi] \vee \tau[\psi])$
- The translation of a logic program P is $\tau[P] = \{\tau[r] \mid r \in P\}$
- Given a logic program P and a set X of atoms,
  X is a stable model of P iff X is a stable model of $\tau[P]$

Relationship to logic programs

- The translation, $\tau[(\phi \leftarrow \psi)]$, of a rule $(\phi \leftarrow \psi)$ is defined as follows:
  - $\tau[(\phi \leftarrow \psi)] = (\tau[\psi] \rightarrow \tau[\phi])$
  - $\tau[\bot] = \bot$
  - $\tau[\top] = \top$
  - $\tau[\phi] = \phi$ if ϕ is an atom
  - $\tau[\neg\phi] = \neg\tau[\phi]$
  - $\tau[(\phi, \psi)] = (\tau[\phi] \wedge \tau[\psi])$
  - $\tau[(\phi; \psi)] = (\tau[\phi] \vee \tau[\psi])$
- The translation of a logic program P is $\tau[P] = \{\tau[r] \mid r \in P\}$
- Given a logic program P and a set X of atoms, X is a stable model of P iff X is a stable model of $\tau[P]$

Logic programs as propositional theories

- The normal logic program $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$ corresponds to $\tau[P] = \{\neg q \rightarrow p,\ \neg p \rightarrow q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The disjunctive logic program $P = \{p\,; q \leftarrow\}$ corresponds to $\tau[P] = \{\top \rightarrow p \vee q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The nested logic program $P = \{p \leftarrow \neg\neg p\}$ corresponds to $\tau[P] = \{\neg\neg p \rightarrow p\}$
  - stable models: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$ corresponds to $\tau[P] = \{\neg q \rightarrow p,\ \neg p \rightarrow q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The disjunctive logic program $P = \{p \,;\, q \leftarrow\}$ corresponds to $\tau[P] = \{\top \rightarrow p \vee q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The nested logic program $P = \{p \leftarrow \neg\neg p\}$ corresponds to $\tau[P] = \{\neg\neg p \rightarrow p\}$
  - stable models: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$ corresponds to $\tau[P] = \{\neg q \rightarrow p,\ \neg p \rightarrow q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The disjunctive logic program $P = \{p \,;\, q \leftarrow\}$ corresponds to $\tau[P] = \{\top \rightarrow p \vee q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The nested logic program $P = \{p \leftarrow \neg\neg p\}$ corresponds to $\tau[P] = \{\neg\neg p \rightarrow p\}$
  - stable models: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$ corresponds to $\tau[P] = \{\neg q \rightarrow p,\ \neg p \rightarrow q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The disjunctive logic program $P = \{p\,;q \leftarrow\}$ corresponds to $\tau[P] = \{\top \rightarrow p \vee q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The nested logic program $P = \{p \leftarrow \neg\neg p\}$ corresponds to $\tau[P] = \{\neg\neg p \rightarrow p\}$
  - stable models: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$ corresponds to $\tau[P] = \{\neg q \rightarrow p,\ \neg p \rightarrow q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The disjunctive logic program $P = \{p\, ;q \leftarrow\}$ corresponds to $\tau[P] = \{\top \rightarrow p \vee q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The nested logic program $P = \{p \leftarrow \neg\neg p\}$ corresponds to $\tau[P] = \{\neg\neg p \rightarrow p\}$
  - stable models: $\emptyset$ and $\{p\}$

Logic programs as propositional theories

- The normal logic program $P = \{p \leftarrow \neg q,\ q \leftarrow \neg p\}$ corresponds to $\tau[P] = \{\neg q \rightarrow p,\ \neg p \rightarrow q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The disjunctive logic program $P = \{p\,;q \leftarrow\}$ corresponds to $\tau[P] = \{\top \rightarrow p \vee q\}$
  - stable models: $\{p\}$ and $\{q\}$
- The nested logic program $P = \{p \leftarrow \neg\neg p\}$ corresponds to $\tau[P] = \{\neg\neg p \rightarrow p\}$
  - stable models: $\emptyset$ and $\{p\}$

Outline

Potassco

Motivation

- Aggregates provide a general way to obtain a single value from a collection of input values
- Popular aggregate (functions)
  - average
  - count
  - maximum
  - minimum
  - sum
- Cardinality and weight constraints rely on count and sum aggregates

Syntax

- An aggregate has the form:

$$\alpha \ \{w_1 : a_1, \ldots, w_m : a_m, w_{m+1} : \neg a_{m+1}, \ldots, w_n : \neg a_n\} \prec k$$

  where for $1 \leq i \leq n$
  - α stands for a function mapping multisets over $\mathbb{Z}$ to $\mathbb{Z} \cup \{+\infty, -\infty\}$
  - $\prec$ stands for a relation between $\mathbb{Z} \cup \{+\infty, -\infty\}$ and $\mathbb{Z}$
  - $k \in \mathbb{Z}$
  - a_i are atoms and
  - w_i are integers
- Example $sum\ \{30 : hd(a), \ldots, 50 : hd(m)\} \leq 300$

Semantics

- A (positive) aggregate $\alpha\ \{w_1 : a_1, \ldots, w_n : a_n\} \prec k$ can be represented by the formula:

$$\bigwedge_{I \subseteq \{1,\ldots,n\}, \alpha\{w_i \mid i \in I\} \not\prec k} \left(\bigwedge_{i \in I} a_i \rightarrow \bigvee_{i \in \overline{I}} a_i \right)$$

  where $\overline{I} = \{1, \ldots, n\} \setminus I$ and $\not\prec$ is the complement of $\prec$
- Then, $\alpha\ \{w_1 : a_1, \ldots, w_n : a_n\} \prec k$ is true in X iff the above formula is true in X

Example

- Consider $sum\{1 : p, 1 : q\} \neq 1$
  That is, $a_1 = p$, $a_2 = q$ and $w_1 = 1$, $w_2 = 1$
- Calculemus!

I	$\{w_i \mid i \in I\}$	$\sum\{w_i \mid i \in I\}$	$\sum\{w_i \mid i \in I\} = 1$
$\emptyset$	$\{\}$	0	*false*
$\{1\}$	$\{1\}$	1	*true*
$\{2\}$	$\{1\}$	1	*true*
$\{1, 2\}$	$\{1, 1\}$	2	*false*

- We get $(p \rightarrow q) \wedge (q \rightarrow p)$
- Analogously, we obtain $(p \vee q) \wedge \neg(p \wedge q)$ for $sum\{1 : p, 1 : q\} = 1$

Potassco

Example

- Consider $sum\{1 : p, 1 : q\} \neq 1$
  That is, $a_1 = p$, $a_2 = q$ and $w_1 = 1$, $w_2 = 1$
- Calculemus!

I	$\{w_i \mid i \in I\}$	$\sum\{w_i \mid i \in I\}$	$\sum\{w_i \mid i \in I\} = 1$
$\emptyset$	$\{\}$	0	*false*
$\{1\}$	$\{1\}$	1	*true*
$\{2\}$	$\{1\}$	1	*true*
$\{1,2\}$	$\{1,1\}$	2	*false*

- We get $(p \rightarrow q) \wedge (q \rightarrow p)$
- Analogously, we obtain $(p \vee q) \wedge \neg(p \wedge q)$ for $sum\{1 : p, 1 : q\} = 1$

Potassco

Example

- Consider $sum\{1 : p, 1 : q\} \neq 1$
  That is, $a_1 = p$, $a_2 = q$ and $w_1 = 1$, $w_2 = 1$
- Calculemus!

I	$\{w_i \mid i \in I\}$	$\sum\{w_i \mid i \in I\}$	$\sum\{w_i \mid i \in I\} = 1$
$\emptyset$	$\{\}$	0	*false*
$\{1\}$	$\{1\}$	1	*true*
$\{2\}$	$\{1\}$	1	*true*
$\{1, 2\}$	$\{1, 1\}$	2	*false*

- We get $(p \rightarrow q) \wedge (q \rightarrow p)$
- Analogously, we obtain $(p \vee q) \wedge \neg(p \wedge q)$ for $sum\{1 : p, 1 : q\} = 1$

Potassco

Monotonicity

- Monotone aggregates
  - For instance,
    - $B(r)^+$
    - $sum\{1:p,1:q\}>1$ amounts to $p \wedge q$
  - We get a simpler characterization: $\bigwedge_{I\subseteq\{1,\dots,n\},\alpha\{w_i\mid i\in I\}\not\prec k}\bigvee_{i\in\bar{I}} a_i$
- Anti-monotone aggregates
  - For instance,
    - $B(r)^-$
    - $sum\{1:p,1:q\}<1$ amounts to $\neg p \wedge \neg q$
  - We get a simpler characterization: $\bigwedge_{I\subseteq\{1,\dots,n\},\alpha\{w_i\mid i\in I\}\not\prec k}\neg\bigwedge_{i\in I} a_i$
- Non-monotone aggregates
  - For instance, $sum\{1:p,1:q\}\neq 1$ is non-monotone.

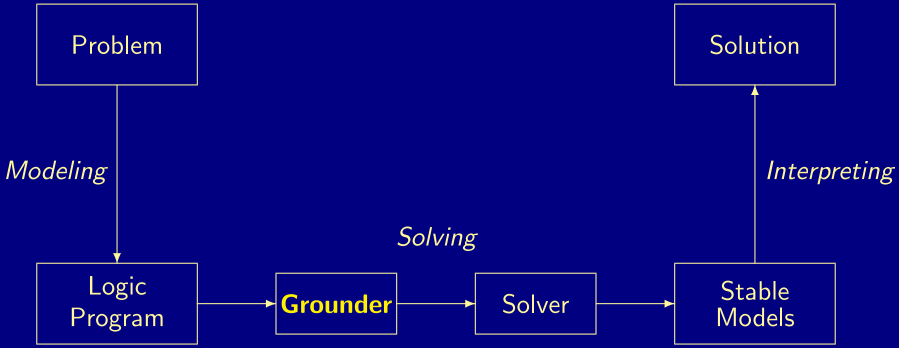
Potassco

Monotonicity

- Monotone aggregates
  - For instance,
    - $B(r)^+$
    - $sum\{1:p,1:q\}>1$ amounts to $p \wedge q$
  - We get a simpler characterization: $\bigwedge_{I\subseteq\{1,\dots,n\},\alpha\{w_i|i\in I\}\not\prec k}\bigvee_{i\in\bar{I}} a_i$
- Anti-monotone aggregates
  - For instance,
    - $B(r)^-$
    - $sum\{1:p,1:q\}<1$ amounts to $\neg p \wedge \neg q$
  - We get a simpler characterization: $\bigwedge_{I\subseteq\{1,\dots,n\},\alpha\{w_i|i\in I\}\not\prec k}\neg\bigwedge_{i\in I} a_i$
- Non-monotone aggregates
  - For instance, $sum\{1:p,1:q\}\neq 1$ is non-monotone.

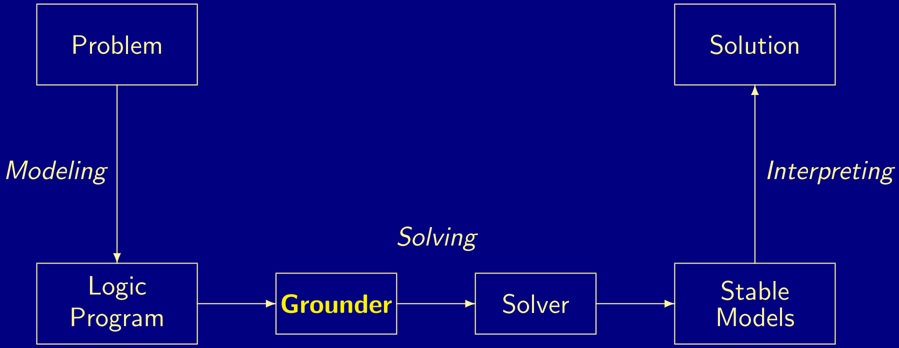
Grounding: Overview

Grounding



- Disclaimer Grounding algorithms for normal logic programs

Grounding



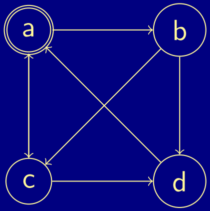
- Disclaimer Grounding algorithms for normal logic programs

Outline

Hamiltonian cycle

Instance

```
% vertices
node(a).  node(b).
node(c).  node(d).

% edges
edge(a,b).  edge(a,c).
edge(b,c).  edge(b,d).
edge(c,a).  edge(c,d).
edge(d,a).

% starting point
start(a).
```



Hamiltonian cycle

Encoding

```
% generate path
path(X,Y) :- not omit(X,Y), edge(X,Y).
omit(X,Y) :- not path(X,Y), edge(X,Y).

% at most one incoming/outgoing edge
:- path(X,Y), path(X',Y), X < X'.
:- path(X,Y), path(X,Y'), Y < Y'.

% at least one incoming/outgoing edge
on_path(Y) :- path(X,Y), path(Y,Z).
:- node(X), not on_path(X).

% connectedness
reach(X) :- start(X).
reach(Y) :- reach(X), path(X,Y).
:- node(X), not reach(X).
```

Potassco

Essential concepts

- Safety of a rule
  - each variable must occur in a positive body literal
- Term universe of a program
  - all constants in the program
- Atom base of a program
  - all ground atoms over predicates in program
- Ground instance of a rule
  - all variables replaced with elements from term universe
- Ground instantiation of a logic program
  - $\mathit{ground}(P)$ is the union of all instances of rules in P
- Unifier of two expressions
  - a (variable) substitution making two expressions equal once applied to both

Potassco

Size of grounding

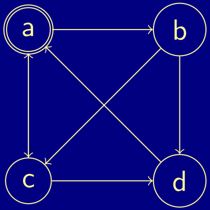
Hamiltonian cycle

```
% term universe: {a,b,c,d}
12 facts from instance
% path(X,Y) :- not omit(X,Y), edge(X,Y).
% omit(X,Y) :- not path(X,Y), edge(X,Y).
% reach(Y) :- reach(X), path(X,Y).
16 rules + 16 rules + 16 rules
% on_path(Y) :- path(X,Y), path(Y,Z).
% :- path(X,Y), path(X',Y), X < X'.
% :- path(X,Y), path(X,Y'), Y < Y'.
64 rules + 64 rules + 64 rules
% reach(X) :- start(X).
% :- node(X), not on_path(X).
% :- node(X), not reach(X).
4 rules + 4 rules + 4 rules
```

Unnecessary rules, I

Hamiltonian cycle

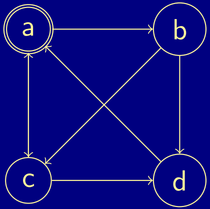
```
% path(X,Y) :- not omit(X,Y), edge(X,Y).
path(a,a) :- not omit(a,a), edge(a,a).
path(a,b) :- not omit(a,b), edge(a,b).
path(a,c) :- not omit(a,c), edge(a,c).
path(a,d) :- not omit(a,d), edge(a,d).
             ⋮
path(d,a) :- not omit(d,a), edge(d,a).
path(d,b) :- not omit(d,b), edge(d,b).
path(d,c) :- not omit(d,c), edge(d,c).
path(d,d) :- not omit(d,d), edge(d,d).
```



Unnecessary rules, II

Hamiltonian cycle

```
% :- path(X,Y), path(X',Y), X < X'.
:- path(a,a), path(a,a), a < a.
:- path(a,b), path(a,b), a < a.
:- path(a,c), path(a,c), a < a.
:- path(a,d), path(a,d), a < a.

:- path(a,a), path(b,a), a < b.
:- path(a,b), path(b,b), a < b.
:- path(a,c), path(b,c), a < b.
:- path(a,d), path(b,d), a < b.
          ⋮
:- path(d,d), path(d,d), d < d.
```



Outline

Bottom up grounding

- Idea Ground relevant rules by gradually extending the atom base
- Relative ground instantiation of a logic program P wrt a set of ground atoms D

$$ground_D(P) = \{r \in ground(P) \mid B(r)^+ \subseteq D\}$$

Potassco

Bottom up grounding

- Idea Ground relevant rules by gradually extending the atom base
- Relative ground instantiation of a logic program P wrt a set of ground atoms D

$$ground_D(P) = \{r \in ground(P) \mid B(r)^+ \subseteq D\}$$

Potassco

Bottom up grounding

- Idea Ground relevant rules by gradually extending the atom base
- Relative ground instantiation of a logic program P wrt a set of ground atoms D

$$ground_D(P) = \{r \in ground(P) \mid B(r)^+ \subseteq D\}$$

- Algorithm

---

function `ground_bottom_up`(P, D)
  $G \leftarrow ground_D(P)$;
  if $H(G) \not\subseteq D$ **then**
    return `ground_bottom_up`$(P, D \cup H(G))$;
  return G;

Bottom up grounding

- Idea Ground relevant rules by gradually extending the atom base
- Relative ground instantiation of a logic program P wrt a set of ground atoms D

$$ground_D(P) = \{r \in ground(P) \mid B(r)^+ \subseteq D\}$$

- Algorithm

---

function `ground_bottom_up`(P, D)
  $G \leftarrow ground_D(P)$;
  if $H(G) \not\subseteq D$ **then**
    return `ground_bottom_up`$(P, D \cup H(G))$;
  return G;

- Property Given a safe normal program P and set of ground facts I, $P \cup I$ is equivalent to `ground_bottom_up`$(P, H(I)) \cup I$

Bottom up grounding, step 1

Hamiltonian cycle

```
% Step 1
path(a,b) :- not omit(a,b), edge(a,b).
          ⋮ % 7 rules total
path(d,a) :- not omit(d,a), edge(d,a).

omit(a,b) :- not path(a,b), edge(a,b).
          ⋮ % 7 rules total
omit(d,a) :- not path(d,a), edge(d,a).

:- node(a), not on_path(a).  :- node(b), not on_path(b).
:- node(c), not on_path(c).  :- node(d), not on_path(d).

:- node(a), not reach(a).  :- node(b), not reach(b).
:- node(c), not reach(c).  :- node(d), not reach(d).

reach(a) :- start(a).
```

Bottom up grounding, step 2

Hamiltonian cycle

```
% Step 2 and rules of Step 1
:- path(a,c), path(b,c), a < b.
:- path(b,d), path(c,d), b < c.
:- path(c,a), path(d,a), c < d.

:- path(a,b), path(a,c), b < c.
:- path(c,a), path(c,d), a < d.
:- path(b,c), path(b,d), c < d.

on_path(a) :- path(c,a), path(a,c).
            ⋮ % 12 rules total
on_path(d) :- path(c,d), path(d,a).

reach(b) :- reach(a), path(a,b).
reach(c) :- reach(a), path(a,c).
```

Bottom up grounding, step 3 and 4

Hamiltonian cycle

```
% Step 3 and rules of Step 2
reach(c) :- reach(b), path(b,c).
reach(d) :- reach(b), path(b,d).
reach(a) :- reach(c), path(c,a).
reach(d) :- reach(c), path(c,d).

% Step 4 and rules of Step 3
reach(a) :- reach(d), path(d,a).
```

Potassco

Properties of bottom-up grounding

- Grounds only relevant rules
  - each positive body literal has a non-cyclic derivation (ignoring negative literals)
- Re-grounds rules from previous steps
- Performs no simplifications

Improving bottom-up grounding

- Use dependencies to focus grounding
  - begin with partial atom base given by facts
  - use rule dependency graph of program to obtain components that can be grounded successively
- Adapt semi-naive evaluation put forward in the database field
  - avoids redundancies when grounding
- Perform simplifications during grounding
  - remove literals from rule bodies if possible
  - omit rules if body cannot be satisfied

Program dependencies

- Dependency graph of program P
  - rule r_2 depends on rule r_1
    if $b \in B(r_2)^+ \cup B(r_2)^-$ unifies with $h \in h(r_1)$
  - $G_P = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ depends on } r_1\}$
- Positive dependency graph of program P
  - rule r_2 positively depends on rule r_1
    if $b \in B(r_2)^+$ unifies with $h \in H(r_1)$
  - $G_P^+ = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ positively depends on } r_1\}$

Program dependencies

- Dependency graph of program P
  - rule r_2 depends on rule r_1
    if $b \in B(r_2)^+ \cup B(r_2)^-$ unifies with $h \in h(r_1)$
  - $G_P = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ depends on } r_1\}$
- Positive dependency graph of program P
  - rule r_2 positively depends on rule r_1
    if $b \in B(r_2)^+$ unifies with $h \in H(r_1)$
  - $G_P^+ = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ positively depends on } r_1\}$

Program dependencies

- Dependency graph of program P
  - rule r_2 depends on rule r_1 if $b \in B(r_2)^+ \cup B(r_2)^-$ unifies with $h \in h(r_1)$
  - $G_P = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ depends on } r_1\}$
- Positive dependency graph of program P
  - rule r_2 positively depends on rule r_1 if $b \in B(r_2)^+$ unifies with $h \in H(r_1)$
  - $G_P^+ = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ positively depends on } r_1\}$

Potassco

Program dependencies

- Dependency graph of program P
  - $G_P = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ depends on } r_1\}$
- Positive dependency graph of program P
  - $G_P^+ = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ positively depends on } r_1\}$

Program dependencies

- Dependency graph of program P
  - $G_P = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ depends on } r_1\}$
- Positive dependency graph of program P
  - $G_P^+ = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ positively depends on } r_1\}$
- Strongly connected components of a directed graph form a partition into sub-graphs in which each node is reachable from any other node

Program dependencies

- Dependency graph of program P
  - $G_P = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ depends on } r_1\}$
- Positive dependency graph of program P
  - $G_P^+ = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ positively depends on } r_1\}$
- Strongly connected components of a directed graph form a partition into sub-graphs in which each node is reachable from any other node
- Topological ordering of strongly connected components
  - $(C_1, \ldots, C_n)$ is a topological ordering of G_P

Potassco

Program dependencies

- Dependency graph of program P
  - $G_P = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ depends on } r_1\}$
- Positive dependency graph of program P
  - $G_P^+ = (P, E)$ where $E = \{(r_1, r_2) \mid r_2 \text{ positively depends on } r_1\}$
- Strongly connected components of a directed graph form a partition into sub-graphs in which each node is reachable from any other node
- Topological ordering of strongly connected components
  - $(C_1, \dots, C_n)$ is a topological ordering of G_P,
    that is, $(r_1, r_2) \in E, r_1 \in C_i, r_2 \in C_j$, implies $i \leq j$

Program dependencies

- Dependency graph of program P
  - $G_P = (P, E)$ where $E = \{(r_1, r_2) \mid r_2$ depends on $r_1\}$
- Positive dependency graph of program P
  - $G_P^+ = (P, E)$ where $E = \{(r_1, r_2) \mid r_2$ positively depends on $r_1\}$
- Strongly connected components of a directed graph form a partition into sub-graphs in which each node is reachable from any other node
- Topological ordering of strongly connected components
  - $(C_1, \dots, C_n)$ is a topological ordering of G_P
  - $(C_{i,1}, \dots, C_{i,m_i})$ is a topological ordering of each $G_{C_i}^+$

Potassco

Program dependencies

- Dependency graph of program P
  - $G_P = (P, E)$ where $E = \{(r_1, r_2) \mid r_2$ depends on $r_1\}$
- Positive dependency graph of program P
  - $G_P^+ = (P, E)$ where $E = \{(r_1, r_2) \mid r_2$ positively depends on $r_1\}$
- Strongly connected components of a directed graph form a partition into sub-graphs in which each node is reachable from any other node
- Topological ordering of strongly connected components
  - $(C_1, \dots, C_n)$ is a topological ordering of G_P
  - $(C_{i,1}, \dots, C_{i,m_i})$ is a topological ordering of each $G_{C_i}^+$

➥ $L_P = (C_{1,1}, \dots, C_{1,m_1}, \dots, C_{n,1}, \dots, C_{n,m_n})$

Potassco

Dependencies

Hamiltonian cycle

$C_{1,1}$	`omit(X,Y) :- not path(X,Y), edge(X,Y).`
$C_{1,2}$	`path(X,Y) :- not omit(X,Y), edge(X,Y).`
$C_{2,1}$	`:- path(X,Y), path(X',Y), X < X'.`
$C_{3,1}$	`:- path(X,Y), path(X,Y'), Y < Y'.`
$C_{4,1}$	`on_path(Y) :- path(X,Y), path(Y,Z).`
$C_{5,1}$	`:- node(X), not on_path(X).`
$C_{6,1}$	`reach(X) :- start(X).`
$C_{7,1}$	`reach(Y) :- reach(X), path(X,Y).`
$C_{8,1}$	`:- node(X), not reach(X).`

Grounding with dependencies

---

function ground_with_dependencies(P, D)
  $G \leftarrow \emptyset$;
  foreach C **in** L_P **do**
    $G' \leftarrow$ ground_bottom_up(C, D);
    $(G, D) \leftarrow (G \cup G', D \cup H(G'))$;
  return G;

- Property Given a safe normal program P and set of facts I, $P \cup I$ is equivalent to ground_with_dependencies$(P, H(I)) \cup I$

Potassco

Grounding with dependencies

---

function $\texttt{ground\_with\_dependencies}(P, D)$
  $G \leftarrow \emptyset$;
  foreach C **in** L_P **do**
    $G' \leftarrow \texttt{ground\_bottom\_up}(C, D)$;
    $(G, D) \leftarrow (G \cup G', D \cup H(G'))$;
  return G;

- Property Given a safe normal program P and set of facts I, $P \cup I$ is equivalent to $\texttt{ground\_with\_dependencies}(P, H(I)) \cup I$

Grounding with dependencies

Hamiltonian cycle

```
% component C_{1,1}
omit(a,b) :- not path(a,b), edge(a,b).
           ⋮ % 7 rules total
omit(d,a) :- not path(d,a), edge(d,a).

% component C_{1,2}
path(a,b) :- not omit(a,b), edge(a,b).
           ⋮ % 7 rules total
path(d,a) :- not omit(d,a), edge(d,a).

...
```

- No re-grounding if there is no positive recursion in a component

Potassco

Grounding with dependencies

Hamiltonian cycle

```
% component C_{1,1}
omit(a,b) :- not path(a,b), edge(a,b).
            ⋮ % 7 rules total
omit(d,a) :- not path(d,a), edge(d,a).

% component C_{1,2}
path(a,b) :- not omit(a,b), edge(a,b).
            ⋮ % 7 rules total
path(d,a) :- not omit(d,a), edge(d,a).

...
```

- No re-grounding if there is no positive recursion in a component

Grounding component $C_{7,1}$

Hamiltonian cycle

```
% Step 1
reach(b) :- reach(a), path(a,b).
reach(c) :- reach(a), path(a,c).

% Step 2 and rules of Step 1
reach(c) :- reach(b), path(b,c).
reach(d) :- reach(b), path(b,d).
reach(a) :- reach(c), path(c,a).
reach(d) :- reach(c), path(c,d).

% Step 3 and rules of Step 2
reach(a) :- reach(d), path(d,a).

% less re-grounding but still...
```

Outline

Semi-naive grounding

- Idea (originates from database systems [1])

  To avoid recomputing the same atoms at each level, semi-naive grounding focuses on newly generated atoms:

  Any new atom at step i relies on at least one atom newly derived at step $i-1$

- Recursive atoms Given $L_P = (C_1, \ldots, C_n)$, an atom a_1 is recursive in component C_i if a_1 unifies a_2 such that
  - $r_1 \in C_i$ and $r_2 \in C_j$ with $i \leq j$
  - $a_1 \in B(r_1)^+ \cup B(r_1)^-$
  - $a_2 \in h(r_2)$

Potassco

Semi-naive grounding

- Idea (originates from database systems [1])

  To avoid recomputing the same atoms at each level, semi-naive grounding focuses on newly generated atoms:

  Any new atom at step i relies on at least one atom newly derived at step $i-1$

- Recursive atoms Given $L_P = (C_1, \dots, C_n)$, an atom a_1 is recursive in component C_i if a_1 unifies a_2 such that
  - $r_1 \in C_i$ and $r_2 \in C_j$ with $i \leq j$
  - $a_1 \in B(r_1)^+ \cup B(r_1)^-$
  - $a_2 \in h(r_2)$

Recursive atoms

Hamiltonian cycle

$C_{1,1}$ `omit(X,Y) :- not path(X,Y), edge(X,Y).`

$C_{1,2}$ `path(X,Y) :- not omit(X,Y), edge(X,Y).`

$C_{2,1}$ `:- path(X,Y), path(X',Y), X < X'.`

$C_{3,1}$ `:- path(X,Y), path(X,Y'), Y < Y'.`

$C_{4,1}$ `on_path(Y) :- path(X,Y), path(Y,Z).`

$C_{5,1}$ `:- node(X), not on_path(X).`

$C_{6,1}$ `reach(X) :- start(X).`

$C_{7,1}$ `reach(Y) :- reach(X), path(X,Y).`

$C_{8,1}$ `:- node(X), not reach(X).`

Preparing components

- The set of prepared rules for $r \in C$ is

$$\left\{\begin{array}{llllll} h \leftarrow n(b_1), & a(b_2), & \dots, & a(b_{i-1}), & a(b_i), & B \\ h \leftarrow o(b_1), & n(b_2), & \dots, & a(b_{i-1}), & a(b_i), & B \\ \vdots & \vdots & \ddots & \vdots & \vdots & \\ h \leftarrow o(b_1), & o(b_2), & \dots, & n(b_{i-1}), & a(b_i), & B \\ h \leftarrow o(b_1), & o(b_2), & \dots, & o(b_{i-1}), & n(b_i), & B \end{array}\right\}$$

or $\quad \{\ h \leftarrow n(b_{i+1}), \dots, n(b_j), b_{j+1}, \dots, b_n\ \} \quad$ if $i = 0$

where

  - $B(r) = \{b_1, \dots, b_i, b_{i+1} \dots, b_j, b_{j+1}, \dots, b_n\}$
  - $b_k \in B(r)^+$ for $1 \leq k \leq i$ is recursive
  - $b_k \in B(r)^+$ for $i < k \leq j$ is not recursive
  - $B = a(b_{i+1}), \dots, a(b_j), b_{j+1}, \dots, b_n$

- A prepared component is the union of all its prepared rules

Preparing components

- The set of prepared rules for $r \in C$ is

$$\left\{\begin{array}{llllll} h \leftarrow n(b_1), & a(b_2), & \dots, & a(b_{i-1}), & a(b_i), & B \\ h \leftarrow o(b_1), & n(b_2), & \dots, & a(b_{i-1}), & a(b_i), & B \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ h \leftarrow o(b_1), & o(b_2), & \dots, & n(b_{i-1}), & a(b_i), & B \\ h \leftarrow o(b_1), & o(b_2), & \dots, & o(b_{i-1}), & n(b_i), & B \end{array}\right\}$$

  or $\{\ h \leftarrow n(b_{i+1}), \dots, n(b_j), b_{j+1}, \dots, b_n\ \}$ if $i = 0$

  where
  - $B(r) = \{b_1, \dots, b_i, b_{i+1} \dots, b_j, b_{j+1}, \dots, b_n\}$
  - $b_k \in B(r)^+$ for $1 \leq k \leq i$ is recursive
  - $b_k \in B(r)^+$ for $i < k \leq j$ is not recursive
  - $B = a(b_{i+1}), \dots, a(b_j), b_{j+1}, \dots, b_n$
- A prepared component is the union of all its prepared rules

Potassco

Preparing components

```
% prepared component C_{1,1}
omit(X,Y) :- n(edge(X,Y)), not path(X,Y).
% prepared component C_{1,2}
path(X,Y) :- n(edge(X,Y)), not omit(X,Y).
% prepared component C_{2,1}
:- n(path(X,Y)), n(path(X',Y)), X < X'.
...
% prepared component C_{7,1}
reach(Y) :- n(reach(X)), a(path(X,Y)).
...
```

Semi-naive evaluation-based grounding

```
function ground_semi_naive(P, A)
    G ← ∅;
    foreach C in L_P do
        (O, N) ← (∅, A);
        repeat
            let D_p = {p(a) | a ∈ D} for set D of atoms;
            G' ← ground_{O_o ∪ N_n ∪ A_a}(prepared C);
            N ← H(G') \ A;
            (G, O, A) ← (G ∪ G', A, N ∪ A);
        until N = ∅;
    return G with o/1, n/1, a/1 stripped from positive bodies;
```

- **Property** Given a safe normal program P and set of facts I, $P \cup I$ is equivalent to $\texttt{ground\_semi\_naive}(P, H(I)) \cup I$

Potassco

Semi-naive evaluation-based grounding

function `ground_semi_naive`(P, A)
- $G \leftarrow \emptyset$;
- **foreach** C **in** L_P **do**
  - $(O, N) \leftarrow (\emptyset, A)$;
  - **repeat**
    - **let** $D_p = \{p(a) \mid a \in D\}$ for set D of atoms;
    - $G' \leftarrow \mathit{ground}_{O_o \cup N_n \cup A_a}(\text{prepared } C)$;
    - $N \leftarrow H(G') \setminus A$;
    - $(G, O, A) \leftarrow (G \cup G', A, N \cup A)$;
  - **until** $N = \emptyset$;
- **return** G with $o/1$, $n/1$, $a/1$ stripped from positive bodies;

- Property Given a safe normal program P and set of facts I, $P \cup I$ is equivalent to `ground_semi_naive`$(P, H(I)) \cup I$

Grounding component $C_{7,1}$

Hamiltonian cycle

```
% grounding of
% reach(Y) :- n(reach(X)), a(path(X,Y)).

% Step 1 with N = A from previous step (reach(a) ∈ A)
reach(b) :- n(reach(a)), a(path(a,b)).
reach(c) :- n(reach(a)), a(path(a,c)).

% Step 2 with N = { reach(b), reach(c) }
reach(c) :- n(reach(b)), a(path(b,c)).
reach(d) :- n(reach(b)), a(path(b,d)).
reach(a) :- n(reach(c)), a(path(c,a)).
reach(d) :- n(reach(c)), a(path(c,d)).

% Step 3 with N = { reach(d) }
reach(a) :- n(reach(d)), a(path(d,a)).
```

Grounding component $C_{7,1}$

Hamiltonian cycle

```
% grounding of
% reach(Y) :- n(reach(X)), a(path(X,Y)).

% Step 1 with N = A from previous step (reach(a) ∈ A)
reach(b) :- reach(a), path(a,b).
reach(c) :- reach(a), path(a,c).

% Step 2 with N = { reach(b), reach(c) }
reach(c) :- reach(b), path(b,c).
reach(d) :- reach(b), path(b,d).
reach(a) :- reach(c), path(c,a).
reach(d) :- reach(c), path(c,d).

% Step 3 with N = { reach(d) }
reach(a) :- reach(d), path(d,a).

% without n/1 and a/1 of course
```

Potassco

Nonlinear programs

Example

```
trans(U,V) :- edge(U,V).
trans(U,W) :- trans(U,V), trans(V,W).
```

```
% prepared Component 1:
trans(U,V) :- n(edge(U,V)).

% prepared Component 2:
trans(U,W) :- n(trans(U,V)), a(trans(V,W)).
trans(U,W) :- o(trans(U,V)), n(trans(V,W)).
```

Nonlinear programs

Example

```
trans(U,V) :- edge(U,V).
% trans(U,W) :- trans(U,V), trans(V,W).
% better written as:
trans(U,W) :- trans(U,V), edge(V,W).

% prepared Component 1:
trans(U,V) :- n(edge(U,V)).

% prepared Component 2:
trans(U,W) :- n(trans(U,V)), a(edge(V,W)).
```

Potassco

Outline

Propagation of atoms

- Simplifications are performed on-the-fly relative to the atom base
  - ➥ rules are printed immediately but not stored in *gringo*
- Distinguish a set of true atoms among the atom base
  - ➥ each true atom amounts to a fact
- Simplifications
  - Remove facts from positive body
  - Discard rules with negative literals over a fact
  - Discard rules whenever the head is a fact
  - Gather new facts whenever a rule body is empty

Potassco

Propagation of true atoms

- Simplifications are performed on-the-fly relative to the atom base
  - ➥ rules are printed immediately but not stored in *gringo*
- Distinguish a set of true atoms among the atom base
  - ➥ each true atom amounts to a fact
- Simplifications
  - Remove facts from positive body
  - Discard rules with negative literals over a fact
  - Discard rules whenever the head is a fact
  - Gather new facts whenever a rule body is empty

Propagation of true atoms

- Simplifications are performed on-the-fly relative to the atom base
  - ➥ rules are printed immediately but not stored in *gringo*
- Distinguish a set of true atoms among the atom base
  - ➥ each true atom amounts to a fact
- Simplifications
  - Remove facts from positive body
  - Discard rules with negative literals over a fact
  - Discard rules whenever the head is a fact
  - Gather new facts whenever a rule body is empty

Propagation of true atoms

Hamiltonian cycle

```
...
path(a,b) :- not omit(a,b), edge(a,b).
...
reach(a) :- start(a).
```

Propagation of true atoms

Hamiltonian cycle

```
...
path(a,b) :- not omit(a,b).
...
reach(a). % reach(a) is added as fact
```

Potassco

Propagation of true atoms

Hamiltonian cycle

```
...
path(a,b) :- not omit(a,b).
...
reach(a). % reach(a) is added as fact

...

:- node(a), not reach(a).
...
```

Propagation of true atoms

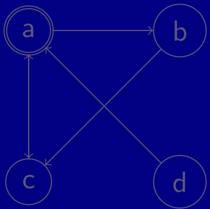
Hamiltonian cycle

```
...
path(a,b) :- not omit(a,b).
...
reach(a). % reach(a) is added as fact

...

~~:- node(a), not reach(a).~~ % rule is discarded
...
```

Propagation of false atoms

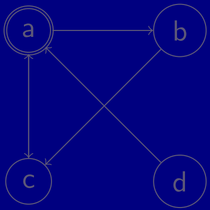
- Non-recursive body negative literals whose atom is not in the current atom base can be removed from rule bodies

- Example Consider the instance where node `d` is not reachable



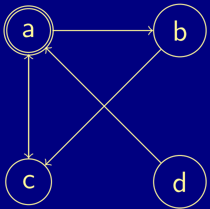
Propagation of false atoms

- Non-recursive body negative literals whose atom is not in the current atom base can be removed from rule bodies
- Example Consider the instance where node `d` is not reachable



Propagation of false atoms

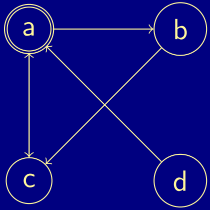
- Non-recursive body negative literals whose atom is not in the current atom base can be removed from rule bodies
- Example Consider the instance where node `d` is not reachable



Propagation of negative literals

Hamiltonian cycle

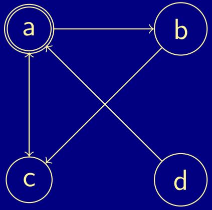
```
path(a,b) :- not omit(a,b).
path(a,c) :- not omit(a,c).
path(b,c) :- not omit(b,c).
path(c,a) :- not omit(c,a).
path(d,a) :- not omit(d,a).
...
reach(a).
reach(b) :- path(a,b).
reach(c) :- path(a,c).
reach(c) :- path(b,c), reach(b).
...
% reach(d) ∉ D  (atom base)
:- not reach(b).
:- not reach(c).
:- not reach(d). % remove not reach(d) from body
```



Propagation of negative literals

Hamiltonian cycle

```
path(a,b) :- not omit(a,b).
path(a,c) :- not omit(a,c).
path(b,c) :- not omit(b,c).
path(c,a) :- not omit(c,a).
path(d,a) :- not omit(d,a).
...
reach(a).
reach(b) :- path(a,b).
reach(c) :- path(a,c).
reach(c) :- path(b,c), reach(b).
...
% reach(d) ∉ D (atom base)
:- not reach(b).
:- not reach(c).
:- . % inconsistency detected during grounding
```



Stratified logic programs

- A logic program P is stratified, if it has a partition $(P_i)_{0\leq i\leq n}$ such that for each predicate p
  - $def_P(p) \subseteq P_i$ for some $1 \leq i \leq n$

  and, for each $1 \leq i \leq n$,
  - if p occurs in $B(r)^+$ for some $r \in P_i$, then $def_P(p) \subseteq \bigcup_{j\leq i} P_j$
  - if p occurs in $B(r)^-$ for some $r \in P_i$, then $def_P(p) \subseteq \bigcup_{j<i} P_j$

- Property Stratified logic programs
  - are completely evaluated during grounding
  - have a single stable model

Stratified logic programs

- The definition of a predicate p in a program P, written $def_P(p)$, is the subset of P consisting of all rules with p in the head

- A logic program P is stratified, if it has a partition $(P_i)_{0 \leq i \leq n}$ such that for each predicate p
  - $def_P(p) \subseteq P_i$ for some $1 \leq i \leq n$

  and, for each $1 \leq i \leq n$,
  - if p occurs in $B(r)^+$ for some $r \in P_i$, then $def_P(p) \subseteq \bigcup_{j \leq i} P_j$
  - if p occurs in $B(r)^-$ for some $r \in P_i$, then $def_P(p) \subseteq \bigcup_{j < i} P_j$

- Property Stratified logic programs
  - are completely evaluated during grounding
  - have a single stable model

Potassco

Stratified logic programs

- The definition of a predicate p in a program P, written $\mathit{def}_P(p)$, is the subset of P consisting of all rules with p in the head
- A logic program P is stratified, if it has a partition $(P_i)_{0\leq i\leq n}$ such that for each predicate p
  - $\mathit{def}_P(p) \subseteq P_i$ for some $1 \leq i \leq n$

  and, for each $1 \leq i \leq n$,
  - if p occurs in $B(r)^+$ for some $r \in P_i$, then $\mathit{def}_P(p) \subseteq \bigcup_{j\leq i} P_j$
  - if p occurs in $B(r)^-$ for some $r \in P_i$, then $\mathit{def}_P(p) \subseteq \bigcup_{j<i} P_j$
- Property Stratified logic programs
  - are completely evaluated during grounding
  - have a single stable model

Stratified logic programs

- The definition of a predicate p in a program P, written $\mathit{def}_P(p)$, is the subset of P consisting of all rules with p in the head
- A logic program P is stratified, if it has a partition $(P_i)_{0\leq i\leq n}$ such that for each predicate p
  - $\mathit{def}_P(p) \subseteq P_i$ for some $1 \leq i \leq n$

  and, for each $1 \leq i \leq n$,
  - if p occurs in $B(r)^+$ for some $r \in P_i$, then $\mathit{def}_P(p) \subseteq \bigcup_{j\leq i} P_j$
  - if p occurs in $B(r)^-$ for some $r \in P_i$, then $\mathit{def}_P(p) \subseteq \bigcup_{j<i} P_j$
- Property Stratified logic programs
  - are completely evaluated during grounding
  - have a single stable model

Potassco

Stratified logic programs

- The definition of a predicate p in a program P, written $\mathit{def}_P(p)$, is the subset of P consisting of all rules with p in the head
- A logic program P is stratified, if it has a partition $(P_i)_{0 \leq i \leq n}$ such that for each predicate p
  - $\mathit{def}_P(p) \subseteq P_i$ for some $1 \leq i \leq n$

  and, for each $1 \leq i \leq n$,
  - if p occurs in $B(r)^+$ for some $r \in P_i$, then $\mathit{def}_P(p) \subseteq \bigcup_{j \leq i} P_j$
  - if p occurs in $B(r)^-$ for some $r \in P_i$, then $\mathit{def}_P(p) \subseteq \bigcup_{j < i} P_j$
- Property Stratified logic programs
  - are completely evaluated during grounding
  - have a single stable model

Stratified logic programs

- A logic program P is stratified, if its dependency graph has no cycles containing negative edges
- Property Stratified logic programs
  - are completely evaluated during grounding
  - have a single stable model

Outline

Safe body order

- Given safe rule r, the tuple $(b_1, \dots, b_n)$ is a safe body order if
  - $\{b_1, \dots, b_n\} = B(r)$
  - the body $\{b_1, \dots, b_i\}$ is safe for each i

- Example Given rule '`:- node(X), not reach(X).`'
  - (`node(X)`, `not reach(X)`) is a safe body order
  - (`not reach(X)`, `node(X)`) is not a safe body order

Safe body order

- Given safe rule r, the tuple $(b_1, \dots, b_n)$ is a safe body order if
  - $\{b_1, \dots, b_n\} = B(r)$
  - the body $\{b_1, \dots, b_i\}$ is safe for each i

- Example Given rule ‘`:- node(X), not reach(X).`’
  - $(\texttt{node(X)}, \texttt{not reach(X)})$ is a safe body order
  - $(\texttt{not reach(X)}, \texttt{node(X)})$ is not a safe body order

Potassco

Safe body order

- Given safe rule r, the tuple $(b_1, \ldots, b_n)$ is a safe body order if
  - $\{b_1, \ldots, b_n\} = B(r)$
  - the body $\{b_1, \ldots, b_i\}$ is safe for each i
- Example Given rule '`:- node(X), not reach(X).`'
  - (`node(X)`, `not reach(X)`) is a safe body order
  - (`not reach(X)`, `node(X)`) is not a safe body order

Matching body literals

- $match_{F,D}(\sigma, b)$ is the set of all matches for literal b
  - σ is a (ground) substitution
  - F are true atoms (set of ground atoms)
  - D is the atom base (set of ground atoms)

  $\sigma' \in match_{F,D}(\sigma, b)$ if

  $\sigma \subseteq \sigma'$ and $vars(b) \subseteq vars(\sigma') \subseteq vars(b) \cup vars(\sigma)$

  $b\sigma'$ holds if b is a comparison literal

  $b\sigma' \in D$ if b is an atom

  $a\sigma' \notin F$ if b is a negative literal of form `not` a

Matching[3] body literals

- $match_{F,D}(\sigma, b)$ is the set of all matches for literal b
  - σ is a (ground) substitution
  - F are true atoms (set of ground atoms)
  - D is the atom base (set of ground atoms)

  $\sigma' \in match_{F,D}(\sigma, b)$ if

  $\sigma \subseteq \sigma'$ and $vars(b) \subseteq vars(\sigma') \subseteq vars(b) \cup vars(\sigma)$

  $b\sigma'$ holds if b is a comparison literal

  $b\sigma' \in D$ if b is an atom

  $a\sigma' \notin F$ if b is a negative literal of form `not` a

[3] A match is a substitution σ such that $a\sigma = b$ for a non-ground atom a and a ground atom b.

Matching[3] body literals

- $match_{F,D}(\sigma, b)$ is the set of all matches for literal b
  - σ is a (ground) substitution
  - F are true atoms (set of ground atoms)
  - D is the atom base (set of ground atoms)

  $\sigma' \in match_{F,D}(\sigma, b)$ if
  - $\sigma \subseteq \sigma'$ and $vars(b) \subseteq vars(\sigma') \subseteq vars(b) \cup vars(\sigma)$
  - $b\sigma'$ holds if b is a comparison literal
  - $b\sigma' \in D$ if b is an atom
  - $a\sigma' \notin F$ if b is a negative literal of form `not` a

[3] A match is a substitution σ such that $a\sigma = b$ for a non-ground atom a and a ground atom b; matching is also referred to as one-sided unification.

Matching body literals

- $match_{F,D}(\sigma, b)$ is the set of all matches for literal b
  - σ is a (ground) substitution
  - F are true atoms (set of ground atoms)
  - D is the atom base (set of ground atoms)
  - $\sigma' \in match_{F,D}(\sigma, b)$ if
    - $\sigma \subseteq \sigma'$ and $vars(b) \subseteq vars(\sigma') \subseteq vars(b) \cup vars(\sigma)$
    - $b\sigma'$ holds if b is a comparison literal
    - $b\sigma' \in D$ if b is an atom
    - $a\sigma' \notin F$ if b is a negative literal of form `not` a

Matching body literals

- $match_{F,D}(\sigma, b)$ is the set of all matches for literal b
  - σ is a (ground) substitution
  - F are true atoms (set of ground atoms)
  - D is the atom base (set of ground atoms)
  - $\sigma' \in match_{F,D}(\sigma, b)$ if
    - $\sigma \subseteq \sigma'$ and $vars(b) \subseteq vars(\sigma') \subseteq vars(b) \cup vars(\sigma)$
    - $b\sigma'$ holds if b is a comparison literal
    - $b\sigma' \in D$ if b is an atom
    - $a\sigma' \notin F$ if b is a negative literal of form `not` a

Matching body literals

- $match_{F,D}(\sigma, b)$ is the set of all matches for literal b
  - σ is a (ground) substitution
  - F are true atoms (set of ground atoms)
  - D is the atom base (set of ground atoms)
  - $\sigma' \in match_{F,D}(\sigma, b)$ if [3]
    - $\sigma \subseteq \sigma'$ and $vars(b) \subseteq vars(\sigma') \subseteq vars(b) \cup vars(\sigma)$
    - $b\sigma'$ holds if b is a comparison literal
    - $b\sigma' \in D$ if b is an atom
    - $a\sigma' \notin F$ if b is a negative literal of form `not` a

[3] $vars(a)$ gives all variables occurring in atom a.

Matching body literals

- $match_{F,D}(\sigma, b)$ is the set of all matches for literal b
  - σ is a (ground) substitution
  - F are true atoms (set of ground atoms)
  - D is the atom base (set of ground atoms)
  - $\sigma' \in match_{F,D}(\sigma, b)$ if[3]
    - $\sigma \subseteq \sigma'$ and $vars(b) \subseteq vars(\sigma') \subseteq vars(b) \cup vars(\sigma)$
    - $b\sigma'$ holds if b is a comparison literal
    - $b\sigma' \in D$ if b is an atom
    - $a\sigma' \notin F$ if b is a negative literal of form `not` a

[3] $vars(\sigma)$ gives all variables mapped by σ on ground terms.

Potassco

Matching body literals

- $match_{F,D}(\sigma, b)$ is the set of all matches for literal b
  - σ is a (ground) substitution
  - F are true atoms (set of ground atoms)
  - D is the atom base (set of ground atoms)
  - $\sigma' \in match_{F,D}(\sigma, b)$ if
    - $\sigma \subseteq \sigma'$ and $vars(b) \subseteq vars(\sigma') \subseteq vars(b) \cup vars(\sigma)$
    - $b\sigma'$ holds if b is a comparison literal
    - $b\sigma' \in D$ if b is an atom
    - $a\sigma' \notin F$ if b is a negative literal of form `not` a

An Example

- Consider body order $(\mathtt{p(X)}, \mathtt{q(X,Y)}, \mathtt{not\ r(Y)})$
  - $F = \{\mathtt{r}(3)\}$
  - $D = \{\mathtt{p}(1), \mathtt{q}(1,2), \mathtt{q}(1,3), \mathtt{r}(3)\}$

- $match_{F,D}(\emptyset, \mathtt{p(X)}) = \{\{\mathtt{X} \mapsto 1\}\}$
- $match_{F,D}(\{\mathtt{X} \mapsto 1\}, \mathtt{q(X,Y)}) = \{\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 2\}, \{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 3\}\}$
- $match_{F,D}(\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 2\}, \mathtt{not\ r(Y)}) = \{\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 2\}\}$
- $match_{F,D}(\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 3\}, \mathtt{not\ r(Y)}) = \emptyset$

An Example

- Consider body order $(\texttt{p(X)}, \texttt{q(X,Y)}, \texttt{not r(Y)})$
  - $F = \{\texttt{r}(3)\}$
  - $D = \{\texttt{p}(1), \texttt{q}(1,2), \texttt{q}(1,3), \texttt{r}(3)\}$

- $match_{F,D}(\emptyset, \texttt{p(X)}) = \{\{\texttt{X} \mapsto 1\}\}$
- $match_{F,D}(\{\texttt{X} \mapsto 1\}, \texttt{q(X,Y)}) = \{\{\texttt{X} \mapsto 1, \texttt{Y} \mapsto 2\}, \{\texttt{X} \mapsto 1, \texttt{Y} \mapsto 3\}\}$
- $match_{F,D}(\{\texttt{X} \mapsto 1, \texttt{Y} \mapsto 2\}, \texttt{not r(Y)}) = \{\{\texttt{X} \mapsto 1, \texttt{Y} \mapsto 2\}\}$
- $match_{F,D}(\{\texttt{X} \mapsto 1, \texttt{Y} \mapsto 3\}, \texttt{not r(Y)}) = \emptyset$

Potassco

An Example

- Consider body order `(p(X), q(X,Y), not r(Y))`
  - $F = \{\mathtt{r}(3)\}$
  - $D = \{\mathtt{p}(1), \mathtt{q}(1,2), \mathtt{q}(1,3), \mathtt{r}(3)\}$

- $match_{F,D}(\emptyset, \mathtt{p(X)}) = \{\{\mathtt{X} \mapsto 1\}\}$
- $match_{F,D}(\{\mathtt{X} \mapsto 1\}, \mathtt{q(X,Y)}) = \{\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 2\}, \{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 3\}\}$
- $match_{F,D}(\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 2\}, \mathtt{not\ r(Y)}) = \{\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 2\}\}$
- $match_{F,D}(\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 3\}, \mathtt{not\ r(Y)}) = \emptyset$

An Example

- Consider body order $(\texttt{p(X)}, \texttt{q(X,Y)}, \texttt{not r(Y)})$
  - $F = \{\mathtt{r}(3)\}$
  - $D = \{\mathtt{p}(1), \mathtt{q}(1,2), \mathtt{q}(1,3), \mathtt{r}(3)\}$

- $match_{F,D}(\emptyset, \texttt{p(X)}) = \{\{\mathtt{X} \mapsto 1\}\}$
- $match_{F,D}(\{\mathtt{X} \mapsto 1\}, \texttt{q(X,Y)}) = \{\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 2\}, \{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 3\}\}$
- $match_{F,D}(\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 2\}, \texttt{not r(Y)}) = \{\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 2\}\}$
- $match_{F,D}(\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 3\}, \texttt{not r(Y)}) = \emptyset$

Potassco

An Example

- Consider body order $(\mathtt{p(X)}, \mathtt{q(X,Y)}, \mathtt{not\ r(Y)})$
  - $F = \{\mathtt{r}(3)\}$
  - $D = \{\mathtt{p}(1), \mathtt{q}(1,2), \mathtt{q}(1,3), \mathtt{r}(3)\}$

- $\mathit{match}_{F,D}(\emptyset, \mathtt{p(X)}) = \{\{\mathtt{X} \mapsto 1\}\}$
- $\mathit{match}_{F,D}(\{\mathtt{X} \mapsto 1\}, \mathtt{q(X,Y)}) = \{\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 2\}, \{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 3\}\}$
- $\mathit{match}_{F,D}(\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 2\}, \mathtt{not\ r(Y)}) = \{\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 2\}\}$
- $\mathit{match}_{F,D}(\{\mathtt{X} \mapsto 1, \mathtt{Y} \mapsto 3\}, \mathtt{not\ r(Y)}) = \emptyset$

Rule grounding by backtracking

function $\texttt{ground\_backtrack}_{r,R,D}(\sigma, F, (b_1, \dots, b_n))$

- **if** $n = 0$ **then**
  - **let** $H = H(r\sigma)$;
    - $B = B(r\sigma)^+ \setminus F \cup ;$
      - $\{\texttt{not}\ a\sigma \mid a \in B(r)^- \setminus R, a\sigma \in D\} \cup ;$
      - $\{\texttt{not}\ a\sigma \mid a \in B(r)^- \cap R\};$
  - **if** $B = \emptyset$ **then** $F \leftarrow F \cup H$;
  - **return** $(\{H \leftarrow B \mid B^- \cap F = \emptyset, H \cap F = \emptyset\}, F)$;
- **else**
  - $G \leftarrow \emptyset$;
  - **foreach** $\sigma' \in match_{F,D}(\sigma, b_1)$ **do**
    - $(G, F) \leftarrow (G, F) \sqcup \texttt{ground\_backtrack}_{r,R,D}(\sigma', F, (b_2, \dots, b_n))$;
  - **return** (G, F);

Recursive atoms

Hamiltonian cycle

$C_{1,1}$ `omit(X,Y) :- not path(X,Y), edge(X,Y).`

$C_{1,2}$ `path(X,Y) :- not omit(X,Y), edge(X,Y).`

$C_{2,1}$ `:- path(X,Y), path(X',Y), X < X'.`

$C_{3,1}$ `:- path(X,Y), path(X,Y'), Y < Y'.`

$C_{4,1}$ `on_path(Y) :- path(X,Y), path(Y,Z).`

$C_{5,1}$ `:- node(X), not on_path(X).`

$C_{6,1}$ `reach(X) :- start(X).`

$C_{7,1}$ `reach(Y) :- reach(X), path(X,Y).`

$C_{8,1}$ `:- node(X), not reach(X).`

Ground rule one

Hamiltonian cycle

- r = 'omit(X,Y) :- not path(X,Y), edge(X,Y).'
- $R = \{\, \mathtt{path}(\mathtt{X}, \mathtt{Y}) \,\}$
- $F = \left\{ \begin{array}{l} \mathtt{edge}(\mathtt{a}, \mathtt{b}), \mathtt{edge}(\mathtt{a}, \mathtt{c}), \mathtt{edge}(\mathtt{b}, \mathtt{c}), \mathtt{edge}(\mathtt{b}, \mathtt{d}), \\ \mathtt{edge}(\mathtt{c}, \mathtt{a}), \mathtt{edge}(\mathtt{c}, \mathtt{d}), \mathtt{edge}(\mathtt{d}, \mathtt{a}), \ldots \end{array} \right\}$

Ground rule one

Hamiltonian cycle

- r = '`omit(X,Y) :- not path(X,Y), edge(X,Y).`'
- $R = \{\, \mathtt{path}(\mathtt{X}, \mathtt{Y}) \,\}$
- $F = \left\{ \begin{array}{l} \mathtt{edge}(\mathtt{a}, \mathtt{b}), \mathtt{edge}(\mathtt{a}, \mathtt{c}), \mathtt{edge}(\mathtt{b}, \mathtt{c}), \mathtt{edge}(\mathtt{b}, \mathtt{d}), \\ \mathtt{edge}(\mathtt{c}, \mathtt{a}), \mathtt{edge}(\mathtt{c}, \mathtt{d}), \mathtt{edge}(\mathtt{d}, \mathtt{a}), \dots \end{array} \right\}$

1. $\mathtt{ground\_backtrack}_{r,R,D}(\emptyset, F, (\mathtt{edge(X,Y),\ not\ path(X,Y)}))$

Ground rule one

Hamiltonian cycle

- r = '`omit(X,Y) :- not path(X,Y), edge(X,Y).`'
- $R = \{\,\mathtt{path}(\mathtt{X},\mathtt{Y})\,\}$
- $F = \left\{\begin{array}{l}\mathtt{edge}(\mathtt{a},\mathtt{b}), \mathtt{edge}(\mathtt{a},\mathtt{c}), \mathtt{edge}(\mathtt{b},\mathtt{c}), \mathtt{edge}(\mathtt{b},\mathtt{d}),\\ \mathtt{edge}(\mathtt{c},\mathtt{a}), \mathtt{edge}(\mathtt{c},\mathtt{d}), \mathtt{edge}(\mathtt{d},\mathtt{a}), \dots\end{array}\right\}$

1. $\mathtt{ground\_backtrack}_{r,R,D}(\emptyset, F, (\mathtt{edge(X,Y),\ not\ path(X,Y)}))$
2. $\mathtt{ground\_backtrack}_{r,R,D}(\{\mathtt{X} \mapsto a, \mathtt{Y} \mapsto b\}, F, (\mathtt{not\ path(X,Y)}))$

Ground rule one

Hamiltonian cycle

- r = '`omit(X,Y) :- not path(X,Y), edge(X,Y).`'
- $R = \{\,\mathtt{path}(\mathtt{X},\mathtt{Y})\,\}$
- $F = \left\{\begin{array}{l}\mathtt{edge}(\mathtt{a},\mathtt{b}),\mathtt{edge}(\mathtt{a},\mathtt{c}),\mathtt{edge}(\mathtt{b},\mathtt{c}),\mathtt{edge}(\mathtt{b},\mathtt{d}),\\ \mathtt{edge}(\mathtt{c},\mathtt{a}),\mathtt{edge}(\mathtt{c},\mathtt{d}),\mathtt{edge}(\mathtt{d},\mathtt{a}),\dots\end{array}\right\}$

1. $\mathtt{ground\_backtrack}_{r,R,D}(\emptyset, F, (\mathtt{edge(X,Y),\ not\ path(X,Y)}))$
2. $\mathtt{ground\_backtrack}_{r,R,D}(\{\mathtt{X}\mapsto a, \mathtt{Y}\mapsto b\}, F, (\mathtt{not\ path(X,Y)}))$
3. $\mathtt{ground\_backtrack}_{r,R,D}(\{\mathtt{X}\mapsto a, \mathtt{Y}\mapsto b\}, F, ())$

Ground rule one

Hamiltonian cycle

- r = '`omit(X,Y) :- not path(X,Y), edge(X,Y).`'
- $R = \{\, \mathtt{path}(\mathtt{X}, \mathtt{Y}) \,\}$
- $F = \left\{ \begin{array}{l} \mathtt{edge}(\mathtt{a}, \mathtt{b}), \mathtt{edge}(\mathtt{a}, \mathtt{c}), \mathtt{edge}(\mathtt{b}, \mathtt{c}), \mathtt{edge}(\mathtt{b}, \mathtt{d}), \\ \mathtt{edge}(\mathtt{c}, \mathtt{a}), \mathtt{edge}(\mathtt{c}, \mathtt{d}), \mathtt{edge}(\mathtt{d}, \mathtt{a}), \ldots \end{array} \right\}$

1. $\mathtt{ground\_backtrack}_{r,R,D}(\emptyset, F, (\mathtt{edge(X,Y)},\ \mathtt{not\ path(X,Y)}))$
2. $\mathtt{ground\_backtrack}_{r,R,D}(\{\mathtt{X} \mapsto a, \mathtt{Y} \mapsto b\}, F, (\mathtt{not\ path(X,Y)}))$
3. $\mathtt{ground\_backtrack}_{r,R,D}(\{\mathtt{X} \mapsto a, \mathtt{Y} \mapsto b\}, F, ())$
   ➥ $(\{\, \mathtt{omit(a,b)\ :-\ not\ path(a,b)} \,\}, F)$

Potassco

Ground rule one

Hamiltonian cycle

- r = 'omit(X,Y) :- not path(X,Y), edge(X,Y).'
- $R = \{\,\mathtt{path}(\mathtt{X},\mathtt{Y})\,\}$
- $F = \left\{\begin{array}{l}\mathtt{edge}(\mathtt{a},\mathtt{b}),\mathtt{edge}(\mathtt{a},\mathtt{c}),\mathtt{edge}(\mathtt{b},\mathtt{c}),\mathtt{edge}(\mathtt{b},\mathtt{d}),\\ \mathtt{edge}(\mathtt{c},\mathtt{a}),\mathtt{edge}(\mathtt{c},\mathtt{d}),\mathtt{edge}(\mathtt{d},\mathtt{a}),\ldots\end{array}\right\}$

1. $\mathtt{ground\_backtrack}_{r,R,D}(\emptyset, F, (\mathtt{edge(X,Y),\ not\ path(X,Y)}))$
2. $\mathtt{ground\_backtrack}_{r,R,D}(\{\mathtt{X}\mapsto a, \mathtt{Y}\mapsto b\}, F, (\mathtt{not\ path(X,Y)}))$
3. $\mathtt{ground\_backtrack}_{r,R,D}(\{\mathtt{X}\mapsto a, \mathtt{Y}\mapsto b\}, F, ())$
   ➥ $(\{\,\mathtt{omit(a,b)\ :\!-\ not\ path(a,b)}\,\}, F)$

- Summary
  edge(a,b) $\rightarrow$ not path(a,b) $\Rightarrow$ omit(a,b) :- not path(a,b)

Potassco

Ground rule one

Hamiltonian cycle

- r = '`omit(X,Y) :- not path(X,Y), edge(X,Y).`'
- $R = \{\, \mathtt{path}(\mathtt{X},\mathtt{Y}) \,\}$
- $F = \left\{ \begin{array}{l} \mathtt{edge}(\mathtt{a},\mathtt{b}), \mathtt{edge}(\mathtt{a},\mathtt{c}), \mathtt{edge}(\mathtt{b},\mathtt{c}), \mathtt{edge}(\mathtt{b},\mathtt{d}), \\ \mathtt{edge}(\mathtt{c},\mathtt{a}), \mathtt{edge}(\mathtt{c},\mathtt{d}), \mathtt{edge}(\mathtt{d},\mathtt{a}), \dots \end{array} \right\}$

- Summary
  `edge(a,b)` $\rightarrow$ `not path(a,b)` $\Rightarrow$ `omit(a,b) :- not path(a,b)`

Potassco

Ground rule one

Hamiltonian cycle

- r = '`omit(X,Y) :- not path(X,Y), edge(X,Y).`'
- $R = \{ \texttt{path}(\texttt{X}, \texttt{Y}) \}$
- $F = \left\{ \begin{array}{l} \texttt{edge}(\texttt{a}, \texttt{b}), \texttt{edge}(\texttt{a}, \texttt{c}), \texttt{edge}(\texttt{b}, \texttt{c}), \texttt{edge}(\texttt{b}, \texttt{d}), \\ \texttt{edge}(\texttt{c}, \texttt{a}), \texttt{edge}(\texttt{c}, \texttt{d}), \texttt{edge}(\texttt{d}, \texttt{a}), \dots \end{array} \right\}$

- Summary

`edge(a,b)`	$\rightarrow$	`not path(a,b)`	$\Rightarrow$	`omit(a,b) :- not path(a,b)`
`edge(a,c)`	$\rightarrow$	`not path(a,c)`	$\Rightarrow$	`omit(a,c) :- not path(a,c)`
`edge(b,c)`	$\rightarrow$	`not path(b,c)`	$\Rightarrow$	`omit(b,c) :- not path(b,c)`
`edge(b,d)`	$\rightarrow$	`not path(b,d)`	$\Rightarrow$	`omit(b,d) :- not path(b,d)`
`edge(c,a)`	$\rightarrow$	`not path(c,a)`	$\Rightarrow$	`omit(c,a) :- not path(c,a)`
`edge(c,d)`	$\rightarrow$	`not path(c,d)`	$\Rightarrow$	`omit(c,d) :- not path(c,d)`
`edge(d,a)`	$\rightarrow$	`not path(d,a)`	$\Rightarrow$	`omit(d,a) :- not path(d,a)`

Special termination cases

- `a(42) :- not b(52)` $R = \emptyset \quad F = \emptyset \quad D = \emptyset$
  - ➥ $(\{$`a(42).`$\}, F \cup \{\mathtt{a(42)}\})$
- `a(42) :- not a(42)` $R = \{a(42)\} \quad F = \emptyset \quad D = \emptyset$
  - ➥ $(\{$`a(42) :- not a(42)`$\}, F)$

Special termination cases

- `a(42) :- not b(52)` $R = \emptyset \quad F = \emptyset \quad D = \emptyset$
  - ➥ $(\{$`a(42).`$\}, F \cup \{\mathtt{a}(42)\})$
- `a(42) :- not a(42)` $R = \{a(42)\} \quad F = \emptyset \quad D = \emptyset$
  - ➥ $(\{$`a(42) :- not a(42)`$\}, F)$

Special termination cases

- `a(42) :- not b(52)` $R = \emptyset \quad F = \emptyset \quad D = \{b(52)\}$
  - ➥ $(\{$`a(42) :- not b(52).`$\}, F)$
- `a(42) :- not a(42)` $R = \{a(42)\} \quad F = \emptyset \quad D = \emptyset$
  - ➥ $(\{$`a(42) :- not a(42)`$\}, F)$

Special termination cases

- `a(42) :- not b(52)` $R = \emptyset \quad F = \{b(52)\} \quad D = \{b(52)\}$
  - ➥ $(\emptyset, F)$
- `a(42) :- not a(42)` $R = \{a(42)\} \quad F = \emptyset \quad D = \emptyset$
  - ➥ $(\{$`a(42) :- not a(42)`$\}, F)$

Potassco

Special termination cases

- `a(42) :- not b(52)` $R = \emptyset \quad F = \{b(52)\} \quad D = \{b(52)\}$

  ➥ $(\emptyset, F)$

- `a(42) :- not a(42)` $R = \{a(42)\} \quad F = \emptyset \quad D = \emptyset$

  ➥ $(\{\texttt{a(42) :- not a(42)}\}, F)$

Outline

Potassco

Things we ignored

- (Recursive) aggregates
- Conditional literals
- Optimization statements
- Disjunctions
- Arithmetic functions
- Syntactic sugar to write more compact encodings
- Safety of $=$ relation (for aggregates and terms)
- Python/Lua integration

Things to remember

- Safety and bottom-up grounding
- Grounding along topological ordering of sets of rule
- Semi-naive evaluation
- Simplifications
- Rule instantiation
- Impact of
  - atoms F found to be true
  - atoms D found to be possible
  - atoms $\mathcal{A} \setminus D$ found to be false

Things to remember

- Safety and bottom-up grounding
- Grounding along topological ordering of sets of rule
- Semi-naive evaluation
- Simplifications
- Rule instantiation
- Impact of
  - atoms F found to be true
  - atoms D found to be possible
  - atoms $\mathcal{A} \setminus D$ found to be false

Computational aspects: Overview

Outline

Potassco

Consequence operator

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

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

  - Iterated applications of T_P are written as T_P^j for $j \geq 0$
    - $T_P^0 X = X$
    - $T_P^i X = T_P T_P^{i-1} X$ for $i \geq 1$
- Properties For any positive program P
  - $Cn(P) = \bigcup_{i \geq 0} T_P^i \emptyset$
  - $X \subseteq Y$ implies $T_P X \subseteq T_P Y$
  - $Cn(P)$ is the smallest fixpoint of T_P

Potassco

Consequence operator

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

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

  - Iterated applications of T_P are written as T_P^j for $j \geq 0$
    - $T_P^0 X = X$
    - $T_P^i X = T_P T_P^{i-1} X$ for $i \geq 1$

- Properties For any positive program P
  - $Cn(P) = \bigcup_{i \geq 0} T_P^i \emptyset$
  - $X \subseteq Y$ implies $T_P X \subseteq T_P Y$
  - $Cn(P)$ is the smallest fixpoint of T_P

Consequence operator

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

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

  - Iterated applications of T_P are written as T_P^j for $j \geq 0$
    - $T_P^0 X = X$
    - $T_P^i X = T_P T_P^{i-1} X$ for $i \geq 1$
- Properties For any positive program P
  - $Cn(P) = \bigcup_{i \geq 0} T_P^i \emptyset$
  - $X \subseteq Y$ implies $T_P X \subseteq T_P Y$
  - $Cn(P)$ is the smallest fixpoint of T_P

Potassco

An example

- Consider the positive program

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

- We get

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

- $Cn(P) = \{p,q,r,t,s\}$ is the smallest fixpoint of T_P because
  - $T_P\{p,q,r,t,s\} = \{p,q,r,t,s\}$ and
  - $T_P X \neq X$ for each $X \subset \{p,q,r,t,s\}$

Potassco

An example

- Consider the positive program

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

- We get

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

- $Cn(P) = \{p,q,r,t,s\}$ is the smallest fixpoint of T_P because
  - $T_P\{p,q,r,t,s\} = \{p,q,r,t,s\}$ and
  - $T_P X \neq X$ for each $X \subset \{p,q,r,t,s\}$

Potassco

An example

- Consider the positive program

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

- We get

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

- $Cn(P) = \{p,q,r,t,s\}$ is the smallest fixpoint of T_P because
  - $T_P\{p,q,r,t,s\} = \{p,q,r,t,s\}$ and
  - $T_P X \neq X$ for each $X \subset \{p,q,r,t,s\}$

Outline

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Potassco

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
    If $X \subseteq U$, then $Cn(P^U) \subseteq X$
    If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Potassco

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$

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

Potassco

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Approximating stable models

- First Idea Approximate a stable model X by two sets of atoms L and U such that $L \subseteq X \subseteq U$
  - L and U constitute lower and upper bounds on X
  - L and $(\mathcal{A} \setminus U)$ describe a three-valued model of the program
- Observation

$$X \subseteq Y \text{ implies } P^Y \subseteq P^X \text{ implies } Cn(P^Y) \subseteq Cn(P^X)$$

- Properties Let X be a stable model of normal logic program P
  - If $L \subseteq X$, then $X \subseteq Cn(P^L)$
  - If $X \subseteq U$, then $Cn(P^U) \subseteq X$
  - If $L \subseteq X \subseteq U$, then $L \cup Cn(P^U) \subseteq X \subseteq U \cap Cn(P^L)$

Approximating stable models

- Second Idea

  repeat

  replace L **by** $L \cup Cn(P^U)$
  replace U **by** $U \cap Cn(P^L)$

  until L and U do not change anymore

- Observations
  - At each iteration step
    - L becomes larger (or equal)
    - U becomes smaller (or equal)
  - $L \subseteq X \subseteq U$ is invariant for every stable model X of P

    If $L \not\subseteq U$, then P has no stable model
    If $L = U$, then L is a stable model of P

Potassco

Approximating stable models

- Second Idea

  repeat

   replace L **by** $L \cup Cn(P^U)$
   replace U **by** $U \cap Cn(P^L)$

  until L and U do not change anymore

- Observations
  - At each iteration step
    - L becomes larger (or equal)
    - U becomes smaller (or equal)
  - $L \subseteq X \subseteq U$ is invariant for every stable model X of P
  - If $L \not\subseteq U$, then P has no stable model
  - If $L = U$, then L is a stable model of P

Approximating stable models

- Second Idea

  repeat

   replace L **by** $L \cup Cn(P^U)$
   replace U **by** $U \cap Cn(P^L)$

  until L and U do not change anymore

- Observations
  - At each iteration step
    - L becomes larger (or equal)
    - U becomes smaller (or equal)
  - $L \subseteq X \subseteq U$ is invariant for every stable model X of P
  - If $L \not\subseteq U$, then P has no stable model
  - If $L = U$, then L is a stable model of P

Potassco

Approximating stable models

- Second Idea

  repeat

  replace L **by** $L \cup Cn(P^U)$
  replace U **by** $U \cap Cn(P^L)$

  until L and U do not change anymore

- Observations
  - At each iteration step
    - L becomes larger (or equal)
    - U becomes smaller (or equal)
  - $L \subseteq X \subseteq U$ is invariant for every stable model X of P
  - If $L \not\subseteq U$, then P has no stable model
  - If $L = U$, then L is a stable model of P

The simplistic expand algorithm

expand$_P(L, U)$

repeat

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

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

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

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

An example

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

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

$$\{a, b\} \subseteq X \quad \text{and} \quad (\mathcal{A} \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset$$

for every stable model X of P

Potassco

An example

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

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

$$\{a, b\} \subseteq X \quad \text{and} \quad (\mathcal{A} \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset$$

for every stable model X of P

Potassco

An example

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

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

$$\{a, b\} \subseteq X \quad \text{and} \quad (\mathcal{A} \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset$$

for every stable model X of P

An example

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

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

$$\{a,b\} \subseteq X \quad \text{and} \quad (\mathcal{A} \setminus \{a,b,d,e\}) \cap X = (\{c\} \cap X) = \emptyset$$

for every stable model X of P

Potassco

An example

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

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

$$\{a, b\} \subseteq X \quad \text{and} \quad (\mathcal{A} \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset$$

for every stable model X of P

An example

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

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

$$\{a, b\} \subseteq X \quad \text{and} \quad (\mathcal{A} \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset$$

for every stable model X of P

Potassco

An example

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

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

$$\{a, b\} \subseteq X \quad \text{and} \quad (\mathcal{A} \setminus \{a, b, d, e\}) \cap X = (\{c\} \cap X) = \emptyset$$

for every stable model X of P

An example

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

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

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

An example

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

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

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

The simplistic expand algorithm

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

Let's expand with d !

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

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

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

Potassco

Let's expand with d !

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

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

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

Let's expand with d !

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

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

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

Potassco

Let's expand with d !

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

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

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

Potassco

Let's expand with d !

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

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

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

Let's expand with $\neg d$!

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

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

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

Potassco

Let's expand with $\neg d$!

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

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

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

Potassco

Let's expand with $\neg d$!

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

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

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

Potassco

Let's expand with $\neg d$!

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

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

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

Let's expand with $\neg d$!

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

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

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

A simplistic solving algorithm

$solve_P(L, U)$

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

A simplistic solving algorithm

- Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - backtracking search building a binary search tree
  - a node in the search tree corresponds to a three-valued interpretation
  - the search space is pruned by
    - deriving deterministic consequences and detecting conflicts (**expand**)
    - making one choice at a time by appeal to a heuristic (**choose**)
  - heuristic choices are made on atoms

A simplistic solving algorithm

- Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - backtracking search building a binary search tree
  - a node in the search tree corresponds to a three-valued interpretation
  - the search space is pruned by
    - deriving deterministic consequences and detecting conflicts (**expand**)
    - making one choice at a time by appeal to a heuristic (**choose**)
  - heuristic choices are made on atoms

A simplistic solving algorithm

- Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - backtracking search building a binary search tree
  - a node in the search tree corresponds to a three-valued interpretation
  - the search space is pruned by
    - deriving deterministic consequences and detecting conflicts (**expand**)
    - making one choice at a time by appeal to a heuristic (**choose**)
  - heuristic choices are made on atoms

A simplistic solving algorithm

- Close to the approach taken by the ASP solver `smodels`, inspired by the Davis-Putman-Logemann-Loveland (DPLL) procedure
  - backtracking search building a binary search tree
  - a node in the search tree corresponds to a three-valued interpretation
  - the search space is pruned by
    - deriving deterministic consequences and detecting conflicts (**expand**)
    - making one choice at a time by appeal to a heuristic (**choose**)
  - heuristic choices are made on atoms

Outline

Complexity

Let a be an atom and X be a set of atoms

- For a positive normal logic program P
  - deciding whether X is the stable model of P is P-complete
  - deciding whether a is in the stable model of P is P-complete
- For a normal logic program P
  - deciding whether X is a stable model of P is P-complete
  - deciding whether a is in a stable model of P is NP-complete
- For a normal logic program P with optimization statements
  - deciding whether X is an optimal stable model of P is $co\text{-}NP$-complete
  - deciding whether a is in an optimal stable model of P is Δ_2^p-complete

Complexity

Let a be an atom and X be a set of atoms

- For a positive normal logic program P
  - deciding whether X is the stable model of P is P-complete
  - deciding whether a is in the stable model of P is P-complete
- For a normal logic program P
  - deciding whether X is a stable model of P is P-complete
  - deciding whether a is in a stable model of P is NP-complete
- For a normal logic program P with optimization statements
  - deciding whether X is an optimal stable model of P is $co\text{-}NP$-complete
  - deciding whether a is in an optimal stable model of P is Δ_2^p-complete

Complexity

Let a be an atom and X be a set of atoms

- For a positive normal logic program P
  - deciding whether X is the stable model of P is P-complete
  - deciding whether a is in the stable model of P is P-complete
- For a normal logic program P
  - deciding whether X is a stable model of P is P-complete
  - deciding whether a is in a stable model of P is NP-complete
- For a normal logic program P with optimization statements
  - deciding whether X is an optimal stable model of P is $co\text{-}NP$-complete
  - deciding whether a is in an optimal stable model of P is Δ_2^p-complete

Complexity

Let a be an atom and X be a set of atoms

- For a positive normal logic program P
  - deciding whether X is the stable model of P is P-complete
  - deciding whether a is in the stable model of P is P-complete
- For a normal logic program P
  - deciding whether X is a stable model of P is P-complete
  - deciding whether a is in a stable model of P is NP-complete
- For a normal logic program P with optimization statements
  - deciding whether X is an optimal stable model of P is $co\text{-}NP$-complete
  - deciding whether a is in an optimal stable model of P is Δ_2^p-complete

Potassco

Complexity

Let a be an atom and X be a set of atoms

- For a positive disjunctive logic program P
  - deciding whether X is a stable model of P is *co-NP*-complete
  - deciding whether a is in a stable model of P is NP^{NP}-complete
- For a disjunctive logic program P
  - deciding whether X is a stable model of P is *co-NP*-complete
  - deciding whether a is in a stable model of P is NP^{NP}-complete
- For a disjunctive logic program P with optimization statements
  - deciding whether X is an optimal stable model of P is *co*-NP^{NP}-complete
  - deciding whether a is in an optimal stable model of P is Δ_3^p-complete
- For a propositional theory Φ
  - deciding whether X is a stable model of Φ is *co-NP*-complete
  - deciding whether a is in a stable model of Φ is NP^{NP}-complete

Complexity

Let a be an atom and X be a set of atoms

- For a positive disjunctive logic program P
  - deciding whether X is a stable model of P is *co-NP*-complete
  - deciding whether a is in a stable model of P is NP^{NP}-complete
- For a disjunctive logic program P
  - deciding whether X is a stable model of P is *co-NP*-complete
  - deciding whether a is in a stable model of P is NP^{NP}-complete
- For a disjunctive logic program P with optimization statements
  - deciding whether X is an optimal stable model of P is $co\text{-}NP^{NP}$-complete
  - deciding whether a is in an optimal stable model of P is Δ_3^p-complete
- For a propositional theory Φ
  - deciding whether X is a stable model of Φ is *co-NP*-complete
  - deciding whether a is in a stable model of Φ is NP^{NP}-complete

Auf Wiedersehen!

"Tomorrow isn't staying out
I'll be back, without a doubt!"

Pink panther

Bibliography

- The following list of references is compiled from the open source bibliography available at

  `https://github.com/krr-up/bibliography`

- Feel free to submit corrections via pull requests!

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] M. Alviano et al. "The ASP System DLV2". In: *Proceedings of the Fourteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17)*. Ed. by M. Balduccini and T. Janhunen. Vol. 10377. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2017, pp. 215–221.

[3] M. Alviano et al. "The Fourth Answer Set Programming Competition: Preliminary Report". In: *Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*. Ed. by P. Cabalar and T. Son. Vol. 8148. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2013, pp. 42–53.

[4] M. Alviano et al. "WASP: A Native ASP Solver Based on Constraint Learning". In: *Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*.

Ed. by P. Cabalar and T. Son. Vol. 8148. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2013, pp. 54–66.

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

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

[7] C. Baral and M. Gelfond. "Logic Programming and Knowledge Representation". In: *Journal of Logic Programming* 12 (1994), pp. 1–80.

[8] P. Borchert et al. "Towards Systematic Benchmarking in Answer Set Programming: The Dagstuhl Initiative". In: *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*. Ed. by V. Lifschitz and I. Niemelä. Vol. 2923. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2004, pp. 3–7.

[9] G. Brewka, T. Eiter, and M. Truszczynski. "Answer Set Programming: An Introduction to the Special Issue". In: *AI Magazine* 37.3 (2016), pp. 5–6.

[10] G. Brewka, T. Eiter, and M. Truszczyński. "Answer set programming at a glance". In: *Communications of the ACM* 54.12 (2011), pp. 92–103.

[11] P. Cabalar and T. Son, eds. *Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*. Vol. 8148. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2013.

[12] F. Calimeri et al. "ASP-Core-2 Input Language Format". In: *Theory and Practice of Logic Programming* 20.2 (2019), pp. 294–309.

[13] F. Calimeri et al. "ASP-Core-2 Input Language Format". In: *Theory and Practice of Logic Programming* 20.2 (2020), pp. 294–309.

[14] F. Calimeri et al. "I-DLV: The new intelligent grounder of DLV". In: *Intelligenza Artificiale* 11.1 (2017), pp. 5–20.

[15] F. Calimeri et al. "The Design of the Fifth Answer Set Programming Competition". In: *Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP'14)*. Ed. by M. Leuschel and T. Schrijvers. Vol. arXiv:1405.3710v4. Theory and Practice of Logic Programming, Online Supplement. 2014. URL: http://arxiv.org/abs/1405.3710v4.

[16] F. Calimeri et al. "The Third Answer Set Programming Competition: Preliminary Report of the System Competition Track". In: *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*. Ed. by J. Delgrande and W. Faber. Vol. 6645. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2011, pp. 388–403.

[17] J. Delgrande and W. Faber, eds. *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*. Vol. 6645. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2011.

[18] M. Denecker et al. "The Second Answer Set Programming Competition". In: *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*. Ed. by E. Erdem, F. Lin, and T. Schaub. Vol. 5753. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2009, pp. 637–654.

[19] T. Eiter, G. Ianni, and T. Krennwallner. "Answer Set Programming: A Primer". In: *Fifth International Reasoning Web Summer School (RW'09)*. Ed. by S. Tessaris et al. Vol. 5689. Lecture Notes in Computer Science. Slides at http://www.kr.tuwien.ac.at/staff/tkren/pub/2009/rw2009-lecture.zip. Springer-Verlag, 2009, pp. 40–110. URL: `http://www.kr.tuwien.ac.at/staff/tkren/pub/2009/rw2009-asp.pdf`.

[20] E. Erdem, F. Lin, and T. Schaub, eds. *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*. Vol. 5753. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2009.

[21] M. Gebser, B. Kaufmann, and T. Schaub. "Conflict-Driven Answer Set Solving: From Theory to Practice". In: *Artificial Intelligence* 187-188 (2012), pp. 52–89.

[22] M. Gebser, T. Schaub, and S. Thiele. "Gringo: A New Grounder for Answer Set Programming". In: *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*. Ed. by C. Baral, G. Brewka, and J. Schlipf. Vol. 4483. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2007, pp. 266–271.

[23] M. Gebser et al. "Abstract Gringo". In: *Theory and Practice of Logic Programming* 15.4-5 (2015), pp. 449–463.

[24] M. Gebser et al. "Advances in gringo Series 3". In: *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*. Ed. by J. Delgrande and W. Faber. Vol. 6645. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2011, pp. 345–351.

[25] M. Gebser et al. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

[26] M. Gebser et al. "Conflict-Driven Answer Set Solving". In: *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*. Ed. by M. Veloso. AAAI/MIT Press, 2007, pp. 386–392.

[27] M. Gebser et al. "Multi-shot ASP solving with clingo". In: *Theory and Practice of Logic Programming* 19.1 (2019), pp. 27–82.

[28] M. Gebser et al. "On the Input Language of ASP Grounder Gringo". In: *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*. Ed. by E. Erdem, F. Lin, and T. Schaub. Vol. 5753. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2009, pp. 502–508.

[29] M. Gebser et al. *Potassco User Guide*. 2nd ed. University of Potsdam. 2015. URL: http://potassco.org.

[30] M. Gebser et al. "The First Answer Set Programming System Competition". In: *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*. Ed. by C. Baral, G. Brewka, and J. Schlipf. Vol. 4483. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2007, pp. 3–17.

[31] M. Gelfond. "Answer Sets". In: *Handbook of Knowledge Representation*. Ed. by V. Lifschitz, F. van Harmelen, and B. Porter. Elsevier Science, 2008. Chap. 7, pp. 285–316.

[32] M. Gelfond and Y. Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, 2014.

[33] M. Gelfond and N. Leone. "Logic programming and knowledge representation — the A-Prolog perspective". In: *Artificial Intelligence* 138.1-2 (2002), pp. 3–38.

[34] M. Gelfond and V. Lifschitz. "Logic Programs with Classical Negation". In: *Proceedings of the Seventh International Conference*

on Logic Programming (ICLP'90). Ed. by D. Warren and P. Szeredi. MIT Press, 1990, pp. 579–597.

[35] M. Gelfond and V. Lifschitz. "The Stable Model Semantics for Logic Programming". In: *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*. Ed. by R. Kowalski and K. Bowen. MIT Press, 1988, pp. 1070–1080.

[36] R. Kaminski, T. Schaub, and P. Wanko. "A Tutorial on Hybrid Answer Set Solving with clingo". In: *Proceedings of the Thirteenth International Summer School of the Reasoning Web*. Ed. by G. Ianni et al. Vol. 10370. Lecture Notes in Computer Science. Springer-Verlag, 2017, pp. 167–203.

[37] N. Leone et al. "The DLV System for Knowledge Representation and Reasoning". In: *ACM Transactions on Computational Logic* 7.3 (2006), pp. 499–562.

[38] V. Lifschitz. *Answer Set Programming*. Springer-Verlag, 2019.

[39] V. Lifschitz. "Answer set programming and plan generation". In: *Artificial Intelligence* 138.1-2 (2002), pp. 39–54.

[40] V. Lifschitz. "Introduction to answer set programming". Unpublished draft. 2004. URL: http://www.cs.utexas.edu/users/vl/papers/esslli.ps.

[41] V. Lifschitz. "Thirteen Definitions of a Stable Model". In: *Fields of Logic and Computation, Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*. Ed. by A. Blass, N. Dershowitz, and W. Reisig. Vol. 6300. Lecture Notes in Computer Science. Springer-Verlag, 2010, pp. 488–503.

[42] V. Lifschitz. "Twelve Definitions of a Stable Model". In: *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*. Ed. by M. Garcia de la Banda and E. Pontelli. Vol. 5366. Lecture Notes in Computer Science. Springer-Verlag, 2008, pp. 37–51.

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

[44] V. Marek and M. Truszczyński. "Stable models and an alternative logic programming paradigm". In: *The Logic Programming Paradigm: a 25-Year Perspective*. Ed. by K. Apt et al. Springer-Verlag, 1999, pp. 375–398.

[45] I. Niemelä. "Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm". In: *Annals of Mathematics and Artificial Intelligence* 25.3-4 (1999), pp. 241–273.

[46] I. Niemelä and P. Simons. "Efficient Implementation of the Well-founded and Stable Model Semantics". In: *Proceedings of the Joint International Conference and Symposium on Logic Programming*. Ed. by M. Maher. MIT Press, 1996, pp. 289–303.

[47] T. Schaub and S. Woltran. "Answer set programming unleashed!" In: *Künstliche Intelligenz* 32.2-3 (2018), pp. 105–108.

[48] T. Schaub and S. Woltran. "Special Issue on Answer Set Programming". In: *Künstliche Intelligenz* 32.2-3 (2018), pp. 101–103.

[49] P. Simons, I. Niemelä, and T. Soininen. "Extending and implementing the stable model semantics". In: *Artificial Intelligence* 138.1-2 (2002), pp. 181–234.

[50] T. Syrjänen. "Omega-Restricted Logic Programs". In: *Proceedings of the Sixth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*. Ed. by T. Eiter, W. Faber, and M. Truszczyński. Vol. 2173. Lecture Notes in Computer Science. Springer-Verlag, 2001, pp. 267–279.