ASP in Industry, here and there

Torsten Schaub

University of Potsdam & Potassco Solutions GmbH

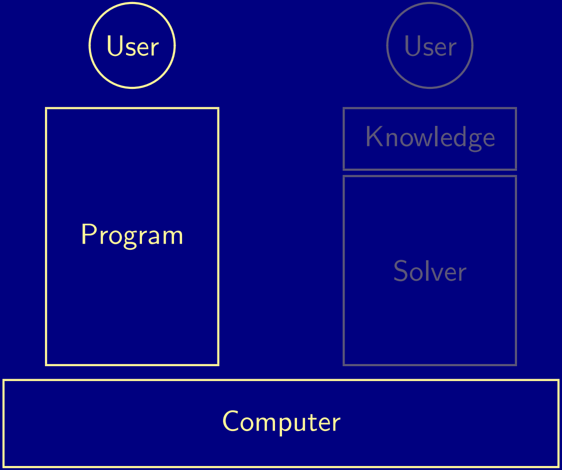
Potassco

Outline

Potassco

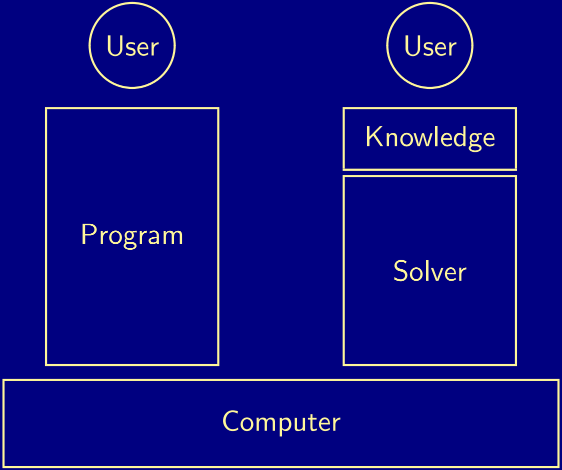
Outline

Traditional Software



Potassco

Knowledge-driven Software



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	

Industrial impact

Within SIEMENS, constraint technologies have been successfully used for solving configuration problems for more than 25 years. [...] approximately 80 percent of the maintenance costs and more than 60 percent of the development costs for the knowledge representation and reasoning tasks were saved.

In: A. Falkner et al. Twenty-Five Years of Successful Application of Constraint Technologies at Siemens. AI Magazine. 37(4):67-80, 2016.

Industrial impact

Within SIEMENS, constraint technologies have been successfully used for solving configuration problems for more than 25 years. [...] approximately 80 percent of the maintenance costs and more than 60 percent of the development costs for the knowledge representation and reasoning tasks were saved.

In: A. Falkner et al. Twenty-Five Years of Successful Application of Constraint Technologies at Siemens. AI Magazine. 37(4):67-80, 2016.

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
  - SBB: Train disposition
  - Siemens: Partner units configuration
  - Variantum: Product configuration

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 ?**
  - Debian, Ubuntu: Linux package configuration
  - Exeura: Call routing
  - **FCC: Radio frequency auction**
  - Gioia Tauro: Workforce management
  - NASA: Decision support for Space Shuttle
  - SBB: Train disposition
  - Siemens: Partner units configuration
  - Variantum: Product configuration

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative pro
- What is ASP good for?
  Solving knowledge-intense combinator
- What problems are this? — **And indu**
  - Debian, Ubuntu: Linux package con
  - Exeura: Call routing
  - **FCC: Radio frequency auction**
  - Gioia Tauro: Workforce management
  - NASA: Decision support for Space Shuttle
  - SBB: Train disposition
  - Siemens: Partner units configuration
  - Variantum: Product configuration

> 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

Potassco

Open and Closed world reasoning

- Open world reasoning
  - if a statement is true, it remains true
  - if a statement is false, it remains false
  - if a statement is unknown, it is either true or false
- Closed world reasoning
  - if a statement is true, it remains true
  - if a statement is false, it remains false
  - if a statement is unknown, it becomes false

Open and Closed world reasoning

- Open world reasoning
  - if a statement is true, it remains true
  - if a statement is false, it remains false
  - if a statement is unknown, it is either true or false
- Closed world reasoning
  - if a statement is true, it remains true
  - if a statement is false, it remains false
  - if a statement is unknown, it becomes false

Open and Closed world reasoning

- Open world reasoning
  - if a statement is true, it remains true
  - if a statement is false, it remains false
  - if a statement is unknown, it is either true or false

  is monotonic
- Closed world reasoning
  - if a statement is true, it remains true
  - if a statement is false, it remains false
  - if a statement is unknown, it becomes false

  is non-monotonic

Open and Closed world reasoning

- Open world reasoning
  - if a statement is true, it remains true
  - if a statement is false, it remains false
  - if a statement is unknown, it is either true or false

  is monotonic
- Closed world reasoning
  - if a statement is true, it remains true
  - if a statement is false, it remains false
  - if a statement is unknown, it becomes false

  is non-monotonic

  offers defaults, reachability, succinctness

Open and Closed world reasoning

- Open world reasoning
  - if a statement is true, it remains true
  - if a statement is false, it remains false
  - if a statement is unknown, it is either true or false

  is monotonic
- Closed world reasoning
  - if a statement is true, it remains true
  - if a statement is false, it remains false
  - if a statement is unknown, it becomes false

  is non-monotonic

  offers defaults, reachability, succinctness
- ASP offers both open and closed world reasoning
  by using stable model semantics

Logic programs

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

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

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

Logic programs

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

$$\underbrace{\texttt{a}_0}_{\text{head}} \;\; \texttt{:-} \;\; \underbrace{\texttt{a}_1, \ldots, \texttt{a}_\texttt{m}, \;\; \texttt{not} \;\; \texttt{a}_{\texttt{m}+1}, \ldots, \;\; \texttt{not} \;\; \texttt{a}_\texttt{n}.}_{\text{body}}$$

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

Logic programs

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

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

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

Logic programs

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

$$\texttt{a}_0 \texttt{ :- } \texttt{a}_1,\ldots,\texttt{a}_\texttt{m}, \texttt{ not } \texttt{a}_{\texttt{m}+1},\ldots, \texttt{ not } \texttt{a}_\texttt{n}.$$

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

- Semantics given by stable models, informally,
  models of P justifying each true atom by a proof

Potassco

Logic programs

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

  $$\texttt{a}_0 \texttt{ :- } \texttt{a}_1,\dots,\texttt{a}_\texttt{m}, \texttt{ not } \texttt{a}_{\texttt{m}+1},\dots, \texttt{ not } \texttt{a}_\texttt{n}.$$

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

- Semantics given by stable models, informally,
  models of P justifying each true atom by a proof

  Minimal models in the logic HT (Heyting'30) / G3 (Gödel'32)

Potassco

Open and Closed world reasoning

by example

- Alphabet $\{a, b\}$
- The rule
  - a

  has the
  - models $\{a\}, \{a, b\}$
  - minimal models $\{a\}$
  - stable models $\{a\}$

Potassco

Open and Closed world reasoning

by example

- Alphabet $\{a, b\}$
- The fact
  - a

  has the
  - models $\{a\}, \{a, b\}$
  - minimal models $\{a\}$
  - stable models $\{a\}$

Open and Closed world reasoning
by example

- Alphabet $\{a, b\}$
- The rule
  - $\neg b \rightarrow a$

  has the
  - models $\{a\}, \{b\}, \{a, b\}$
  - minimal models $\{a\}, \{b\}$
  - stable models $\{a\}$

Potassco

The logic of Here-and-There (HT)

- Formula $\varphi \ ::= \ \bot \ \mid \ a \ \mid \ \varphi \wedge \varphi \ \mid \ \varphi \vee \varphi \ \mid \ \varphi \rightarrow \varphi$
- Interpretation A pair $\langle H, T \rangle$ of sets of atoms with $H \subseteq T$
  - H is called "here" and
  - T is called "there"
- Note $\langle H, T \rangle$ is a simplified Kripke structure
- Intuition
  - H represents provably true atoms
  - T represents possibly true atoms
  - atoms not in T are false
- Idea
  - $\langle H, T \rangle \models \varphi \quad \sim \quad \varphi$ is provably true
  - $\langle T, T \rangle \models \varphi \quad \sim \quad \varphi$ is possibly true (ie, classically true)

Potassco

The logic of Here-and-There (HT)

- Formula $\varphi \ ::= \ \bot \ | \ a \ | \ \varphi \wedge \varphi \ | \ \varphi \vee \varphi \ | \ \varphi \rightarrow \varphi$
- Interpretation A pair $\langle H, T\rangle$ of sets of atoms with $H \subseteq T$
  - H is called "here" and
  - T is called "there"
- Note $\langle H, T\rangle$ is a simplified Kripke structure
- Intuition
  - H represents provably true atoms
  - T represents possibly true atoms
  - atoms not in T are false
- Idea
  - $\langle H, T\rangle \models \varphi \quad \sim \quad \varphi$ is provably true
  - $\langle T, T\rangle \models \varphi \quad \sim \quad \varphi$ is possibly true (ie, classically true)

The logic of Here-and-There (HT)

- Formula $\varphi \ ::= \ \bot \ | \ a \ | \ \varphi \wedge \varphi \ | \ \varphi \vee \varphi \ | \ \varphi \rightarrow \varphi$
- Interpretation A pair $\langle H, T \rangle$ of sets of atoms with $H \subseteq T$
  - H is called "here" and
  - T is called "there"
- Note $\langle H, T \rangle$ is a simplified Kripke structure
- Intuition
  - H represents provably true atoms
  - T represents possibly true atoms
  - atoms not in T are false
- Idea
  - $\langle H, T \rangle \models \varphi \quad \sim \quad \varphi$ is provably true
  - $\langle T, T \rangle \models \varphi \quad \sim \quad \varphi$ is possibly true (ie, classically true)

The logic of Here-and-There (HT)

- Formula $\varphi \ ::= \ \bot \ \mid \ a \ \mid \ \varphi \wedge \varphi \ \mid \ \varphi \vee \varphi \ \mid \ \varphi \rightarrow \varphi$
- Interpretation A pair $\langle H, T \rangle$ of sets of atoms with $H \subseteq T$
  - H is called "here" and
  - T is called "there"
- Note $\langle H, T \rangle$ is a simplified Kripke structure
- Intuition
  - H represents provably true atoms
  - T represents possibly true atoms
  - atoms not in T are false
- Idea
  - $\langle H, T \rangle \models \varphi \quad \sim \quad \varphi$ is provably true
  - $\langle T, T \rangle \models \varphi \quad \sim \quad \varphi$ is possibly true (ie, classically true)

The logic of Here-and-There (HT)

- Formula $\varphi \ ::= \ \bot \ | \ a \ | \ \varphi \wedge \varphi \ | \ \varphi \vee \varphi \ | \ \varphi \rightarrow \varphi$
- Interpretation A pair $\langle H, T \rangle$ of sets of atoms with $H \subseteq T$
  - H is called "here" and
  - T is called "there"
- Note $\langle H, T \rangle$ is a simplified Kripke structure
- Intuition
  - H represents provably true atoms
  - T represents possibly true atoms
  - atoms not in T are false
- Idea
  - $\langle H, T \rangle \models \varphi \quad \sim \quad \varphi$ is provably true
  - $\langle T, T \rangle \models \varphi \quad \sim \quad \varphi$ is possibly true (ie, classically true)

Potassco

The logic of Here-and-There (HT)

- Formula $\varphi \ ::= \ \bot \ | \ a \ | \ \varphi \wedge \varphi \ | \ \varphi \vee \varphi \ | \ \varphi \rightarrow \varphi$
- Interpretation A pair $\langle H, T \rangle$ of sets of atoms with $H \subseteq T$
  - H is called "here" and
  - T is called "there"
- Note $\langle H, T \rangle$ is a simplified Kripke structure
- Intuition
  - H represents provably true atoms
  - T represents possibly true atoms
  - atoms not in T are false
- Idea
  - $\langle H, T \rangle \models \varphi \quad \sim \quad \varphi$ is provably true
  - $\langle T, T \rangle \models \varphi \quad \sim \quad \varphi$ is possibly true (ie, classically true)

Satisfaction

- $\langle H, T \rangle \models a$ if $a \in H$ for any atom a
- $\langle H, T \rangle \models \varphi \wedge \psi$ if $\langle H, T \rangle \models \varphi$ and $\langle H, T \rangle \models \psi$
- $\langle H, T \rangle \models \varphi \vee \psi$ if $\langle H, T \rangle \models \varphi$ or $\langle H, T \rangle \models \psi$
- $\langle H, T \rangle \models \varphi \rightarrow \psi$ if $\langle X, T \rangle \models \varphi$ implies $\langle X, T \rangle \models \psi$ for both $X = H, T$
- Note $\langle H, T \rangle \models \neg\varphi$ if $\langle T, T \rangle \not\models \varphi$ since $\neg\varphi = \varphi \rightarrow \bot$
- An interpretation $\langle H, T \rangle$ is a model of φ, if $\langle H, T \rangle \models \varphi$

Potassco

Satisfaction

- $\langle H, T \rangle \models a$ if $a \in H$ for any atom a
- $\langle H, T \rangle \models \varphi \land \psi$ if $\langle H, T \rangle \models \varphi$ and $\langle H, T \rangle \models \psi$
- $\langle H, T \rangle \models \varphi \lor \psi$ if $\langle H, T \rangle \models \varphi$ or $\langle H, T \rangle \models \psi$
- $\langle H, T \rangle \models \varphi \rightarrow \psi$ if $\langle X, T \rangle \models \varphi$ implies $\langle X, T \rangle \models \psi$ for both $X = H, T$

- Note $\langle H, T \rangle \models \neg\varphi$ if $\langle T, T \rangle \not\models \varphi$ since $\neg\varphi = \varphi \rightarrow \bot$

- An interpretation $\langle H, T \rangle$ is a model of φ, if $\langle H, T \rangle \models \varphi$

Satisfaction

- $\langle H, T\rangle \models a$ if $a \in H$ for any atom a
- $\langle H, T\rangle \models \varphi \wedge \psi$ if $\langle H, T\rangle \models \varphi$ and $\langle H, T\rangle \models \psi$
- $\langle H, T\rangle \models \varphi \vee \psi$ if $\langle H, T\rangle \models \varphi$ or $\langle H, T\rangle \models \psi$
- $\langle H, T\rangle \models \varphi \rightarrow \psi$ if $\langle X, T\rangle \models \varphi$ implies $\langle X, T\rangle \models \psi$ for both $X = H, T$
- Note $\langle H, T\rangle \models \neg\varphi$ if $\langle T, T\rangle \not\models \varphi$ since $\neg\varphi = \varphi \rightarrow \perp$
- An interpretation $\langle H, T\rangle$ is a model of φ, if $\langle H, T\rangle \models \varphi$

Satisfaction

- $\langle H, T \rangle \models a$ if $a \in H$ for any atom a
- $\langle H, T \rangle \models \varphi \wedge \psi$ if $\langle H, T \rangle \models \varphi$ and $\langle H, T \rangle \models \psi$
- $\langle H, T \rangle \models \varphi \vee \psi$ if $\langle H, T \rangle \models \varphi$ or $\langle H, T \rangle \models \psi$
- $\langle H, T \rangle \models \varphi \rightarrow \psi$ if $\langle X, T \rangle \models \varphi$ implies $\langle X, T \rangle \models \psi$ for both $X = H, T$

- Note $\langle H, T \rangle \models \neg\varphi$ if $\langle T, T \rangle \not\models \varphi$ since $\neg\varphi = \varphi \rightarrow \bot$

- An interpretation $\langle H, T \rangle$ is a model of φ, if $\langle H, T \rangle \models \varphi$

Tautologies

H	T	a	$\neg a$	$a \vee \neg a$	$\neg\neg a$	$\neg\neg a \vee \neg a$	$a \leftarrow \neg\neg a$
$\{a\}$	$\{a\}$	$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{T}$
$\emptyset$	$\{a\}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{F}$
$\emptyset$	$\emptyset$	$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{T}$	$\boldsymbol{F}$	$\boldsymbol{T}$	$\boldsymbol{T}$

Tautologies

H	T	a	$\neg a$	$a \vee \neg a$	$\neg\neg a$	$\neg\neg a \vee \neg a$	$a \leftarrow \neg\neg a$
$\{a\}$	$\{a\}$	**T**	**F**	**T**	**T**	**T**	**T**
$\emptyset$	$\{a\}$	**F**	**F**	**F**	**T**	**T**	**F**
$\emptyset$	$\emptyset$	**F**	**T**	**T**	**F**	**T**	**T**

Potassco

Equilibrium models

(Pearce'96)

- A total interpretation $\langle T, T \rangle$ is an equilibrium model of a formula φ, if
  1. $\langle T, T \rangle \models \varphi$
  2. $\langle H, T \rangle \not\models \varphi$ for all $H \subset T$
- T is called a stable model of φ
- Note
  - $\langle T, T \rangle$ acts as a classical model
  - $\langle H, T \rangle \models P$ iff $H \models P^T$ (P^T is the reduct of P by T)

Potassco

Equilibrium models

(Pearce'96)

- A total interpretation $\langle T, T\rangle$ is an equilibrium model of a formula φ, if
  1. $\langle T, T\rangle \models \varphi$
  2. $\langle H, T\rangle \not\models \varphi$ for all $H \subset T$
- T is called a stable model of φ
- Note
  - $\langle T, T\rangle$ acts as a classical model
  - $\langle H, T\rangle \models P$ iff $H \models P^T$ (P^T is the reduct of P by T)

Potassco

Equilibrium models

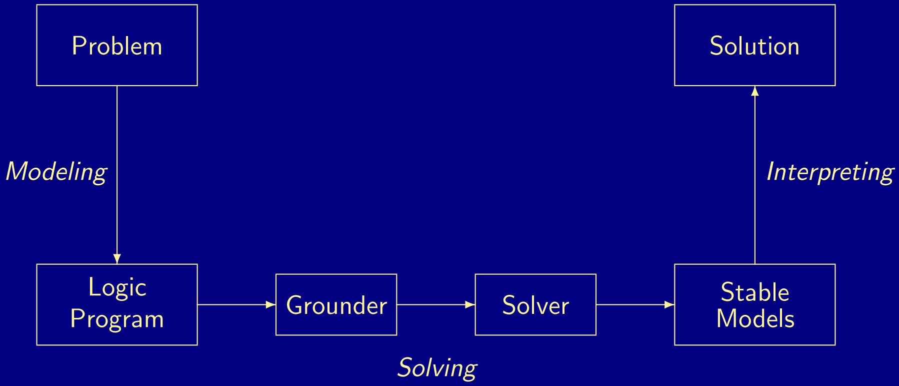
(Pearce'96)

- A total interpretation $\langle T, T \rangle$ is an equilibrium model of a formula φ, if
  1. $\langle T, T \rangle \models \varphi$
  2. $\langle H, T \rangle \not\models \varphi$ for all $H \subset T$
- T is called a stable model of φ
- Note
  - $\langle T, T \rangle$ acts as a classical model
  - $\langle H, T \rangle \models P$ iff $H \models P^T$ (P^T is the reduct of P by T)

Outline

Potassco

Modeling, grounding, and 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]`
  `#minimize { C : q(X), p(X,C) }`

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]
```

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
```

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
```

Outline

Potassco

Routing and scheduling

Applications

- Routing
  - multi-agent path finding
  - phylogenetic inference
  - wire routing
  - etc
- Routing and Scheduling
  - train scheduling
  - embedded system design
  - warehouse robotics
  - etc
- Techniques
  - index variables by time steps
  - view time as an order on variables

Routing and scheduling

Applications

- Routing
  - multi-agent path finding
  - phylogenetic inference
  - wire routing
  - etc
- Routing and Scheduling
  - train scheduling
  - embedded system design
  - warehouse robotics
  - etc
- Techniques
  - index variables by time steps
  - view time as an order on variables

Routing and scheduling

Applications

- Routing
  - multi-agent path finding
  - phylogenetic inference
  - wire routing
  - etc
- Routing and Scheduling
  - train scheduling
  - embedded system design
  - warehouse robotics
  - etc
- Techniques
  - index variables by time steps
  - view time as an order on variables

Potassco

Routing and scheduling

Applications

- Routing
  - multi-agent path finding
  - phylogenetic inference
  - wire routing
  - etc
- Routing and Scheduling
  - train scheduling
  - embedded system design
  - warehouse robotics
  - etc
- Techniques
  - index variables by time steps
  - view time as an order on variables

Routing and scheduling

Applications

- Routing
  - multi-agent path finding
  - phylogenetic inference
  - wire routing
  - etc
- Routing and Scheduling
  - train scheduling
  - embedded system design
  - warehouse robotics
  - etc

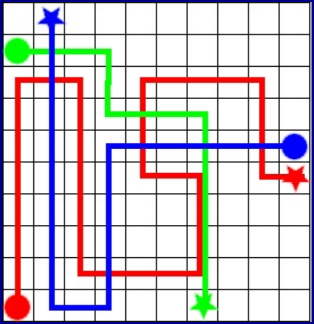
- Techniques
  - index variables by time steps
  - view time as an order on variables

Multi-agent path finding

- Problem Find (optimal) collision-free paths for a group of agents from their location to an (assigned) target
- Example

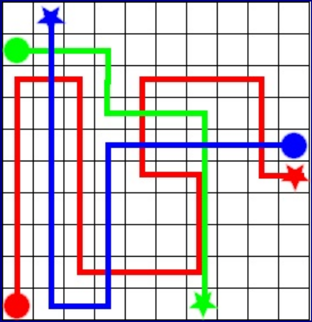
Multi-agent path finding

- Problem Find (optimal) collision-free paths for a group of agents from their location to an (assigned) target
- Example



Multi-agent path finding

- Problem Find (optimal) collision-free paths for a group of agents from their location to an (assigned) target
- Example



THREE ENGINEERS, HUNDREDS OF ROBOTS, ONE WAREHOUSE

Kiva Systems wants to revolutionize distribution centers by setting swarms of robots loose on the inventory BY ERICO GUIZZO

NO HANDS: Machines do the heavy lifting at a Staples Denver facility.

Timestep-based routing

No scheduling yet

```
% guess moves
{ move(A,U,V,T): edge(U,V) } <= 1 :- agent(A), T=1..n.

% infer agent positions
at(A,U,0) :- start(A,U).
at(A,V,T) :- move(A,_,V,T), T=1..n.
at(A,U,T) :- at(A,U,T-1), not move(A,U,_,T), T=1..n.

% ensure path-like strolls
:- move(A,U,_,T), not at(A,U,T-1).
:- goal(A,U), not at(A,U,n).

% handle vertex/swap/follow conflicts
:- { at(A,U,T) } > 1, vertex(U), T=0..n.
:- move(_,U,V,T), move(_,V,U,T).
:- at(A,U,T), at(B,U,T+1), A!=B, m=fc.

% ensure unique agent positions (redundant/for performance)
:- { at(A,U,T) } != 1, agent(A), T=1..n.
```

Potassco

Timestep-based to -free routing

in view of scheduling

- Idea $\mathtt{move(A,U,V,T)} \rightsquigarrow \mathtt{move(A,U,V)}$
- Pros
  - no timesteps
  - no explicit bound
- Cons
  - no cyclic (parts of) trajectories

Timestep-free routing, part I

No scheduling yet

```
% generate moves with in and out degrees of one
{ move(A,U,V): edge(U,V) } <= 1 :- agent(A), vertex(V).
{ move(A,U,V): edge(U,V) } <= 1 :- agent(A), vertex(U).

:-  move(A,U,_), not start(A,U), not move(A,_,U).
:-  move(A,_,U), not  goal(A,U), not move(A,U,_).

% fix in and out degrees of start and goal vertices
:- start(A,U), move(A,_,U).
:-  goal(A,U), move(A,U,_).

:- start(A,U), not  goal(A,U), not move(A,U,_).
:-  goal(A,U), not start(A,U), not move(A,_,U).
```

Potassco

Timestep-free routing, part I

No scheduling yet

```
% generate moves with in and out degrees of one
{ move(A,U,V): edge(U,V) } <= 1 :- agent(A), vertex(V).
{ move(A,U,V): edge(U,V) } <= 1 :- agent(A), vertex(U).

:-  move(A,U,_), not start(A,U), not move(A,_,U).
:-  move(A,_,U), not  goal(A,U), not move(A,U,_).

% fix in and out degrees of start and goal vertices
:- start(A,U), move(A,_,U).
:-  goal(A,U), move(A,U,_).

:- start(A,U), not  goal(A,U), not move(A,U,_).
:-  goal(A,U), not start(A,U), not move(A,_,U).
```

Potassco

Timestep-free routing, part II

No scheduling yet

```
% generate order considering conflict positions
resolve(A,B,U) :- start(A,U), move(B,_,U), A!=B.
resolve(A,B,U) :-  goal(B,U), move(A,_,U), A!=B.

{ resolve(A,B,U);
  resolve(B,A,U) } >= 1 :- move(A,_,U), move(B,_,U), A<B.

% discard invalid orders
:- resolve(A,B,U), resolve(B,A,U).
```

Potassco

Timestep-free routing, part III

No scheduling yet

- Acyclicity constraints

```
% check order
#edge ((A,U),(A,V)) : move(A,U,V).
#edge ((A,V),(B,U)) : resolve(A,B,U), move(A,U,V).
```

- Difference constraints

```
% check order
&diff{(A,U)+1}<=(A,V) :- move(A,U,V).
&diff{(A,V)+1}<=(B,U) :- resolve(A,B,U), move(A,U,V).
```

- Difference constraints

```
% check order
&diff{(A,U)+D}<=(A,V) :- move(A,U,V), edge(U,V,D).
&diff{(A,V)+D}<=(B,U) :- resolve(A,B,U), move(A,U,V), edge(U,V,D).
```

Potassco

Timestep-free routing, part III

No scheduling yet

- Acyclicity constraints (*clingo*)

```
% check order
#edge ((A,U),(A,V)) : move(A,U,V).
#edge ((A,V),(B,U)) : resolve(A,B,U), move(A,U,V).
```

- Difference constraints (*clingo*[DL])

```
% check order
&diff{(A,U)+1}<=(A,V) :- move(A,U,V).
&diff{(A,V)+1}<=(B,U) :- resolve(A,B,U), move(A,U,V).
```

- Difference constraints

```
% check order
&diff{(A,U)+D}<=(A,V) :- move(A,U,V), edge(U,V,D).
&diff{(A,V)+D}<=(B,U) :- resolve(A,B,U), move(A,U,V), edge(U,V,D).
```

Timestep-free routing and scheduling, III

- Acyclicity constraints (*clingo*, routing only)

```
% check order
#edge ((A,U),(A,V)) : move(A,U,V).
#edge ((A,V),(B,U)) : resolve(A,B,U), move(A,U,V).
```

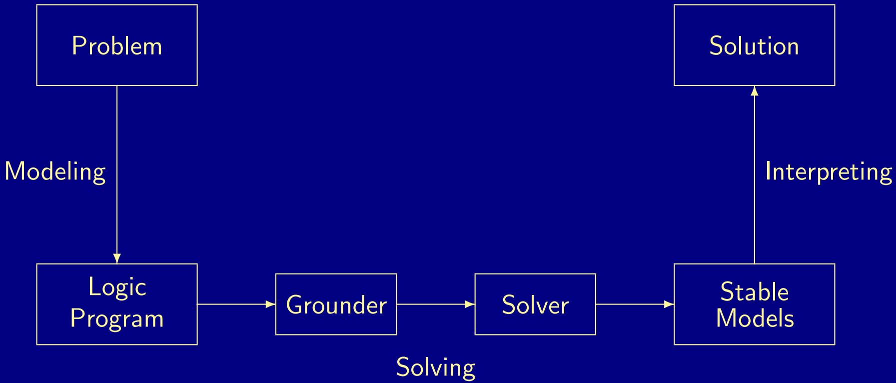
- Difference constraints (*clingo*[DL], routing only)

```
% check order
&diff{(A,U)+1}<=(A,V) :- move(A,U,V).
&diff{(A,V)+1}<=(B,U) :- resolve(A,B,U), move(A,U,V).
```

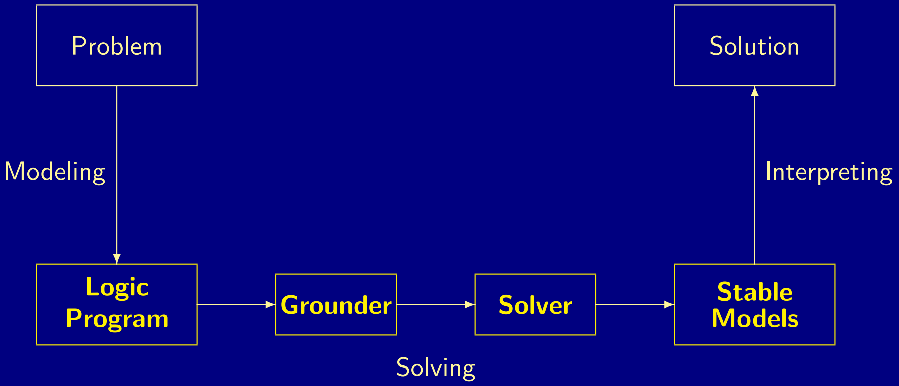
- Difference constraints (*clingo*[DL], routing and scheduling)

```
% check order
&diff{(A,U)+D}<=(A,V) :- move(A,U,V), edge(U,V,D).
&diff{(A,V)+D}<=(B,U) :- resolve(A,B,U), move(A,U,V), edge(U,V,D).
```

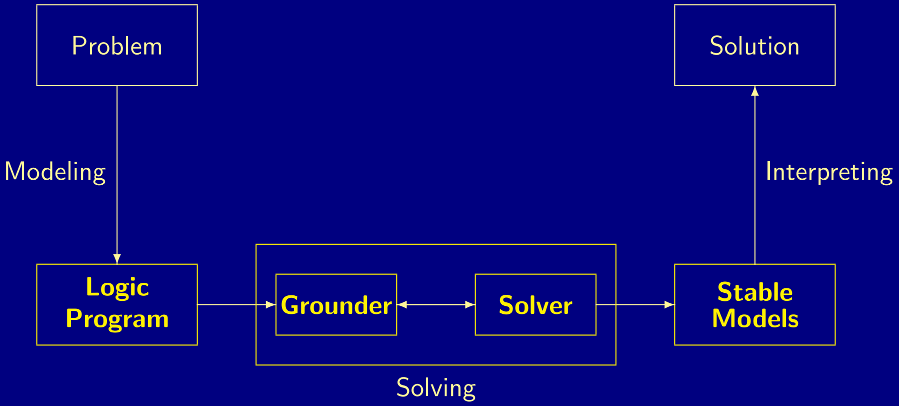
ASP solving process



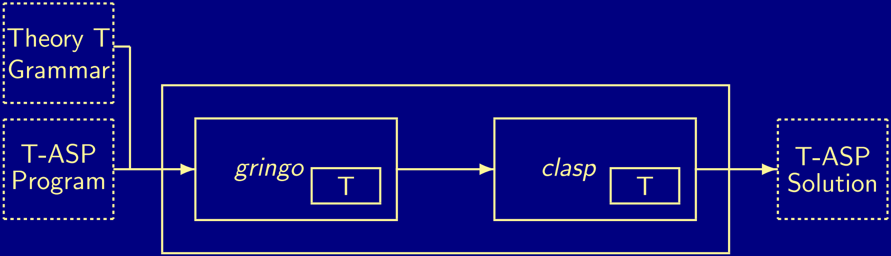
ASP solving process modulo theories



ASP solving process modulo theories



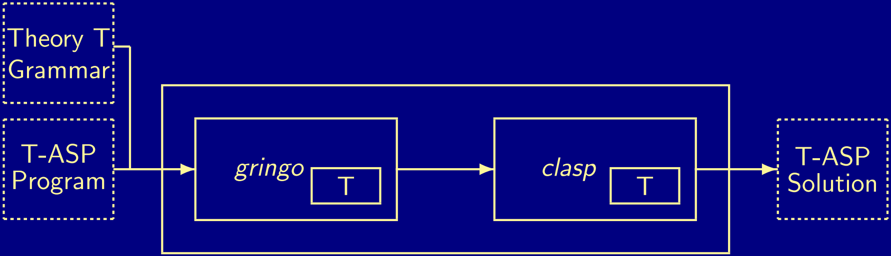
clingo's approach



- Challenge Logic programs with elusive theory atoms
- Example The atom "&sum{x;-y}<=4" stands for difference constraint $x - y \leq 4$

Potassco

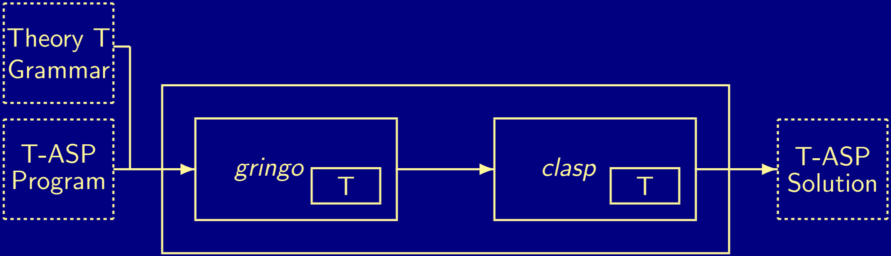
clingo's approach



- Challenge Logic programs with elusive theory atoms
- Example The atom "`&sum{x;-y}<=4`" stands for difference constraint $x - y \leq 4$

Potassco

clingo's approach



- Challenge Logic programs with elusive theory atoms
- Example The atom "`&sum{x;-y}<=4`" stands for difference constraint $x - y \leq 4$

Open and Closed world reasoning

on numeric domains

- Open world reasoning
  - if a variable occurs in true constraints, it is assigned appropriate values
  - if a variable occurs in no constraint, it takes all possible values

- Closed world reasoning
  - if a variable occurs in true constraints, it is assigned appropriate values
  - if a variable occurs in no constraint, it is undefined

Potassco

Open and Closed world reasoning
on numeric domains

- Open world reasoning
  - if a variable occurs in true constraints, it is assigned appropriate values
  - if a variable occurs in no constraint, it takes all possible values

- Closed world reasoning
  - if a variable occurs in true constraints, it is assigned appropriate values
  - if a variable occurs in no constraint, it is undefined

Potassco

Open and Closed world reasoning

on numeric domains

- Open world reasoning
  - if a variable occurs in true constraints, it is assigned appropriate values
  - if a variable occurs in no constraint, it takes all possible values

  is monotonic
- Closed world reasoning
  - if a variable occurs in true constraints, it is assigned appropriate values
  - if a variable occurs in no constraint, it is undefined

  is non-monotonic

Potassco

Open and Closed world reasoning

on numeric domains

- Open world reasoning
  - if a variable occurs in true constraints, it is assigned appropriate values
  - if a variable occurs in no constraint, it takes all possible values

  is monotonic
- Closed world reasoning
  - if a variable occurs in true constraints, it is assigned appropriate values
  - if a variable occurs in no constraint, it is undefined

  is non-monotonic

  offers defaults, succinctness

HT_c Syntax

- Signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{A} \rangle$
  - $\mathcal{X}$ variables
  - $\mathcal{D}$ domain
  - $\mathcal{A}$ atoms

- Note The syntax of atoms is left open
- Example Atom "$x - y \leq d$" with $x, y \in \mathcal{X}$ and $d \in \mathcal{D}$

- HT_c-formula φ over $\mathcal{A}$

$$\varphi \ ::= \ \bot \ | \ a \ | \ \varphi \wedge \varphi \ | \ \varphi \vee \varphi \ | \ \varphi \rightarrow \varphi \qquad \text{where } a \in \mathcal{A}$$

HT_c Syntax

- Signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{A} \rangle$
  - $\mathcal{X}$ variables
  - $\mathcal{D}$ domain
  - $\mathcal{A}$ atoms
- Note The syntax of atoms is left open
- Example Atom "$x - y \leq d$" with $x, y \in \mathcal{X}$ and $d \in \mathcal{D}$
- HT_c-formula φ over $\mathcal{A}$

$$\varphi \quad ::= \quad \bot \mid a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \qquad \text{where } a \in \mathcal{A}$$

HT_c Syntax

- Signature $\langle \mathcal{X}, \mathcal{D}, \mathcal{A} \rangle$
  - $\mathcal{X}$ variables
  - $\mathcal{D}$ domain
  - $\mathcal{A}$ atoms
- Note The syntax of atoms is left open
- Example Atom "$x - y \leq d$" with $x, y \in \mathcal{X}$ and $d \in \mathcal{D}$
- HT_c-formula φ over $\mathcal{A}$

$$\varphi \; ::= \; \bot \mid a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \qquad \text{where } a \in \mathcal{A}$$

HT_c Semantics

- Valuation $v : \mathcal{X} \to \mathcal{D} \cup \{\boldsymbol{u}\}$
  - $\boldsymbol{u} \notin \mathcal{X} \cup \mathcal{D}$ stands for undefined
- Set-based representation $v \subseteq \mathcal{X} \times \mathcal{D}$
  - $(x, c) \in v$ and $(x, d) \in v$ implies $c = d$
  - $(x, d) \notin v$ if $v(x) = \boldsymbol{u}$

  $\mathcal{V}$ is the set of all valuations over $\mathcal{X}$ and $\mathcal{D}$

- Atom denotation $[\![\cdot]\!] : \mathcal{A} \to 2^{\mathcal{V}}$
- Example

$$[\![\text{“}x - y \leq d\text{”}]\!] = \{v \in \mathcal{V} \mid v(x), v(y), d \in \mathbb{Z},\ v(x) - v(y) \leq d\}$$

Potassco

HT_c Semantics

- Valuation $v : \mathcal{X} \rightarrow \mathcal{D} \cup \{\boldsymbol{u}\}$
  - $\boldsymbol{u} \notin \mathcal{X} \cup \mathcal{D}$ stands for undefined

  Set-based representation $v \subseteq \mathcal{X} \times \mathcal{D}$
  - $(x, c) \in v$ and $(x, d) \in v$ implies $c = d$
  - $(x, d) \notin v$ if $v(x) = \boldsymbol{u}$

  $\mathcal{V}$ is the set of all valuations over $\mathcal{X}$ and $\mathcal{D}$

- Atom denotation $[\![\cdot]\!] : \mathcal{A} \rightarrow 2^{\mathcal{V}}$
- Example

$$[\![\text{“}x - y \leq d\text{”}]\!] = \{v \in \mathcal{V} \mid v(x), v(y), d \in \mathbb{Z},\ v(x) - v(y) \leq d\}$$

HT_c Semantics

- Valuation $v : \mathcal{X} \rightarrow \mathcal{D} \cup \{\boldsymbol{u}\}$
  - $\boldsymbol{u} \notin \mathcal{X} \cup \mathcal{D}$ stands for undefined

  Set-based representation $v \subseteq \mathcal{X} \times \mathcal{D}$
  - $(x, c) \in v$ and $(x, d) \in v$ implies $c = d$
  - $(x, d) \notin v$ if $v(x) = \boldsymbol{u}$

  $\mathcal{V}$ is the set of all valuations over $\mathcal{X}$ and $\mathcal{D}$

- Atom denotation $[\![\cdot]\!] : \mathcal{A} \rightarrow 2^{\mathcal{V}}$
- Example

$$[\![\text{“}x - y \leq d\text{”}]\!] = \{v \in \mathcal{V} \mid v(x), v(y), d \in \mathbb{Z},\ v(x) - v(y) \leq d\}$$

HT_c Semantics

- Valuation $v : \mathcal{X} \rightarrow \mathcal{D} \cup \{\boldsymbol{u}\}$
  - $\boldsymbol{u} \notin \mathcal{X} \cup \mathcal{D}$ stands for undefined

  Set-based representation $v \subseteq \mathcal{X} \times \mathcal{D}$
  - $(x, c) \in v$ and $(x, d) \in v$ implies $c = d$
  - $(x, d) \notin v$ if $v(x) = \boldsymbol{u}$

  $\mathcal{V}$ is the set of all valuations over $\mathcal{X}$ and $\mathcal{D}$
- Atom denotation $[\![\cdot]\!] : \mathcal{A} \rightarrow 2^{\mathcal{V}}$
- Example

$$[\![\text{“}x - y \leq d\text{”}]\!] = \{v \in \mathcal{V} \mid v(x), v(y), d \in \mathbb{Z},\ v(x) - v(y) \leq d\}$$

Potassco

HT_c-satisfaction

- HT_c-interpretation over $\mathcal{X}$,$\mathcal{D}$ is a pair $\langle h, t\rangle$ of valuations over $\mathcal{X}$,$\mathcal{D}$ such that $h \subseteq t$

- An HT_c-interpretation $\langle h, t\rangle$ satisfies a formula φ, written $\langle h, t\rangle \models \varphi$, if the following conditions hold
  1. $\langle h, t\rangle \not\models \bot$
  2. $\langle h, t\rangle \models a$ if both $h \in [\![\, a \,]\!]$ and $t \in [\![\, a \,]\!]$ for $a \in \mathcal{A}$
  3. $\langle h, t\rangle \models \varphi \wedge \psi$ if $\langle h, t\rangle \models \varphi$ and $\langle h, t\rangle \models \psi$
  4. $\langle h, t\rangle \models \varphi \vee \psi$ if $\langle h, t\rangle \models \varphi$ or $\langle h, t\rangle \models \psi$
  5. $\langle h, t\rangle \models \varphi \rightarrow \psi$ if $\langle h', t\rangle \not\models \varphi$ or $\langle h', t\rangle \models \psi$ for both $h' = h$ and $h' = t$.

HT_c-satisfaction

- HT_c-interpretation over $\mathcal{X}$,$\mathcal{D}$ is a pair $\langle h, t\rangle$ of valuations over $\mathcal{X}$,$\mathcal{D}$ such that $h \subseteq t$

- An HT_c-interpretation $\langle h, t\rangle$ satisfies a formula φ, written $\langle h, t\rangle \models \varphi$, if the following conditions hold
  1. $\langle h, t\rangle \not\models \bot$
  2. $\langle h, t\rangle \models a$ if both $h \in [\![\, a \,]\!]$ and $t \in [\![\, a \,]\!]$ for $a \in \mathcal{A}$
  3. $\langle h, t\rangle \models \varphi \wedge \psi$ if $\langle h, t\rangle \models \varphi$ and $\langle h, t\rangle \models \psi$
  4. $\langle h, t\rangle \models \varphi \vee \psi$ if $\langle h, t\rangle \models \varphi$ or $\langle h, t\rangle \models \psi$
  5. $\langle h, t\rangle \models \varphi \rightarrow \psi$ if $\langle h', t\rangle \not\models \varphi$ or $\langle h', t\rangle \models \psi$ for both $h' = h$ and $h' = t$.

Potassco

HT_c-equilibrium model

- A total interpretation $\langle t, t\rangle$ is an equilibrium model of a formula φ, if
  1. $\langle t, t\rangle \models \varphi$
  2. $\langle h, t\rangle \not\models \varphi$ for all $h \subset t$
- t is called an HT_c-stable model of φ

HT_c-equilibrium model

- A total interpretation $\langle t, t \rangle$ is an equilibrium model of a formula φ, if
  1. $\langle t, t \rangle \models \varphi$
  2. $\langle h, t \rangle \not\models \varphi$ for all $h \subset t$
- t is called an HT_c-stable model of φ

Potassco

HT_c benefits

- Semantic framework for ASP modulo theory systems (AMT) combining closed and open world reasoning
  - conservative extension of HT
  - flexibility due to open syntax and denotational semantics
  - study of AMT systems
  - study of language fragments
  - soundness of program transformations
  - warrant substitution of equivalent expressions
  - etc.

Outline

More features of interest

- Meta programming
- Qualitative and quantitative optimization
- Heuristic programming
- Application interface programming
  - Multi-shot solving
  - Theory solving
- Linear Temporal, Dynamic and Metric reasoning
- Visualization

- Playful? https://potassco.org

Potassco

More features of interest

- Meta programming
- Qualitative and quantitative optimization
- Heuristic programming
- Application interface programming
  - Multi-shot solving
  - Theory solving
- Linear Temporal, Dynamic and Metric reasoning
- Visualization

- Playful? `https://potassco.org`

Potassco

Outline

1 Motivation

2 Nutshell

3 Foundation

4 Usage

5 At work

6 Omissions

7 Recap

Take home message

Take home message

Modeling + Grounding + Solving

Take home message

Modeling + Grounding + Solving

ASP = DB+LP+KR+SAT

Take home message

Modeling + Grounding + Solving

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

Take home message

Modeling + Grounding + Solving

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

https://potassco.org

Potassco

Take home message

Modeling + Grounding + Solving

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

`https://potassco.org`

And it's fun !

Potassco