Towards embedded Answer Set Solving

Torsten Schaub

University of Potsdam

Outline

1 Introduction

2 Foundations

3 Modeling

4 Modeling and Controlling

5 Case-study: Ricochet Robots

6 Case-sketch: Preferences and optimization

7 Potassco

8 Summary

Potassco

Outline

1 Introduction
2 Foundations
3 Modeling
4 Modeling and Controlling
5 Case-study: Ricochet Robots
6 Case-sketch: Preferences and optimization
7 Potassco
8 Summary

Potassco

Answer Set Programming (ASP)

- ASP is an approach to declarative problem solving
  - describe the problem, not how to solve it
- ASP allows for solving hard search and optimization problems
  - Systems Biology
  - Product Configuration
  - Linux Package Configuration
  - Robotics
  - Music Composition
  - ...
- All search-problems in NP (and NP^{NP}) are expressible

Potassco

Answer Set Programming (ASP)

- ASP is an approach to declarative problem solving
  - describe the problem, not how to solve it
- ASP allows for solving hard search and optimization problems
  - Systems Biology
  - Product Configuration
  - Linux Package Configuration
  - Robotics
  - Music Composition
  - ...
- All search-problems in NP (and NP^{NP}) are expressible

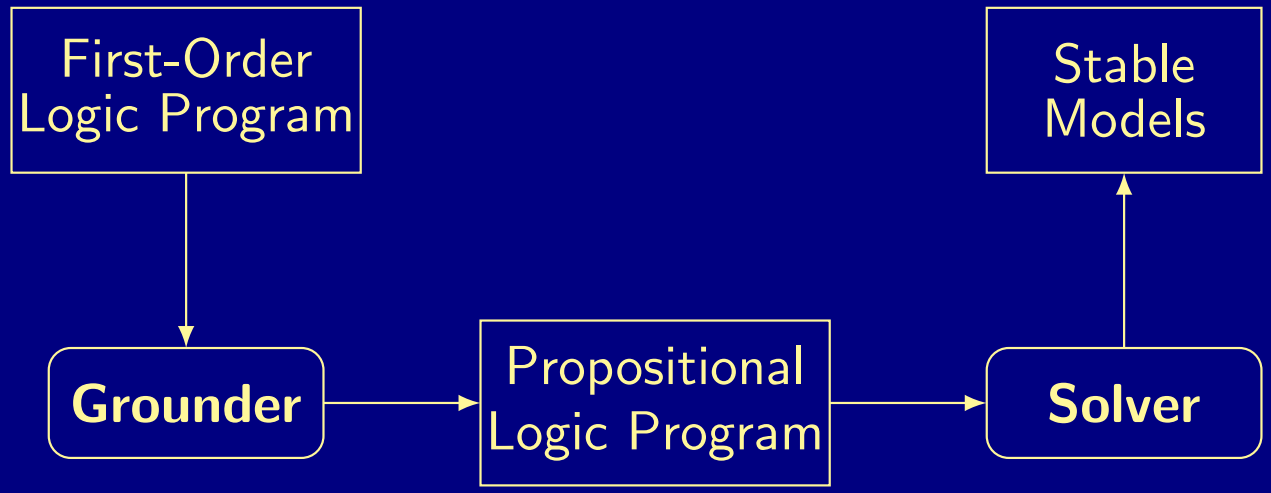
Potassco

Answer Set Programming (ASP)

- ASP is an approach to declarative problem solving
  - describe the problem, not how to solve it
- ASP allows for solving hard search and optimization problems
  - Systems Biology
  - Product Configuration
  - Linux Package Configuration
  - Robotics
  - Music Composition
  - ...
- All search-problems in NP (and NP^{NP}) are expressible

Potassco

The ASP Solving Process

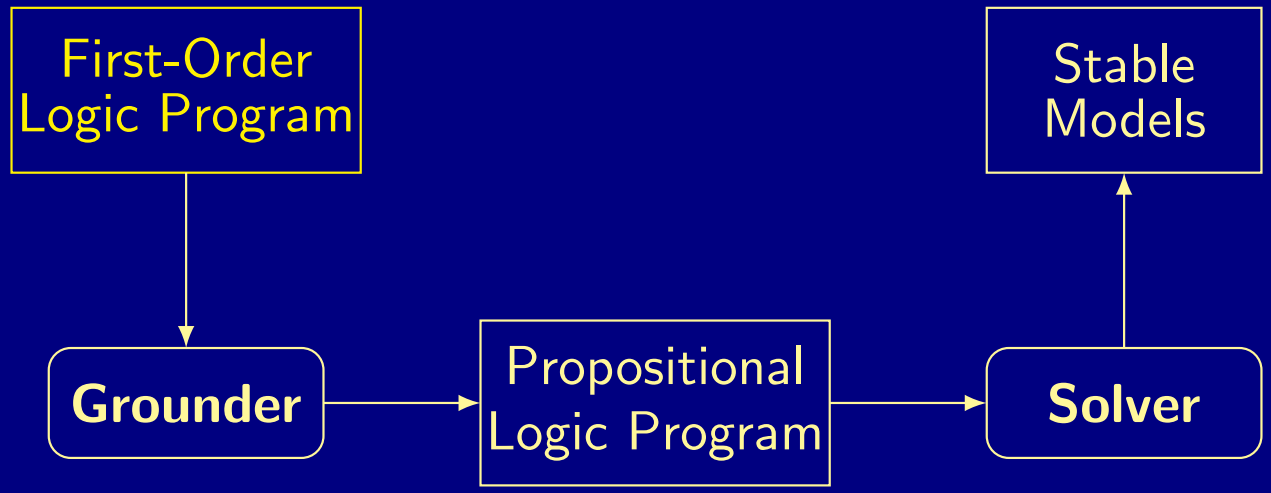


Expressive modeling language

Powerful grounding and solving tools

Potassco

The ASP Solving Process

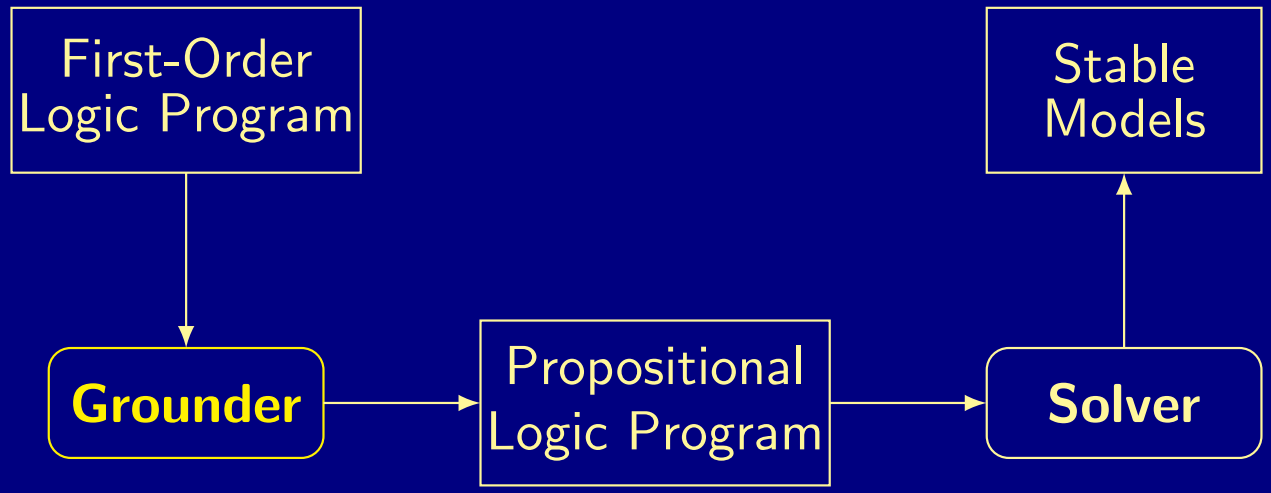


Expressive modeling language

Powerful grounding and solving tools

Potassco

The ASP Solving Process

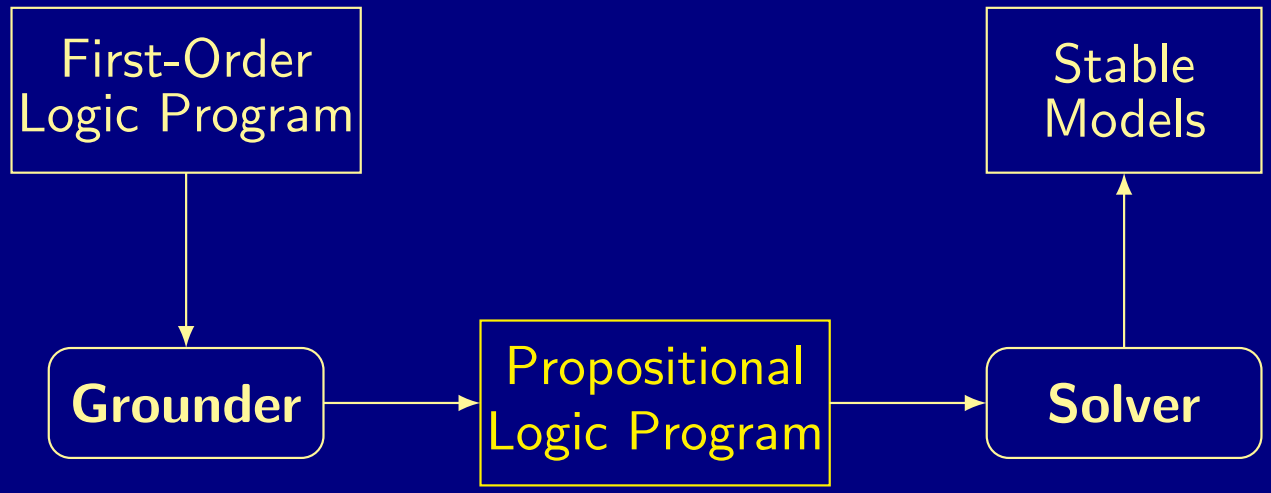


Expressive modeling language

Powerful grounding and solving tools

Potassco

The ASP Solving Process

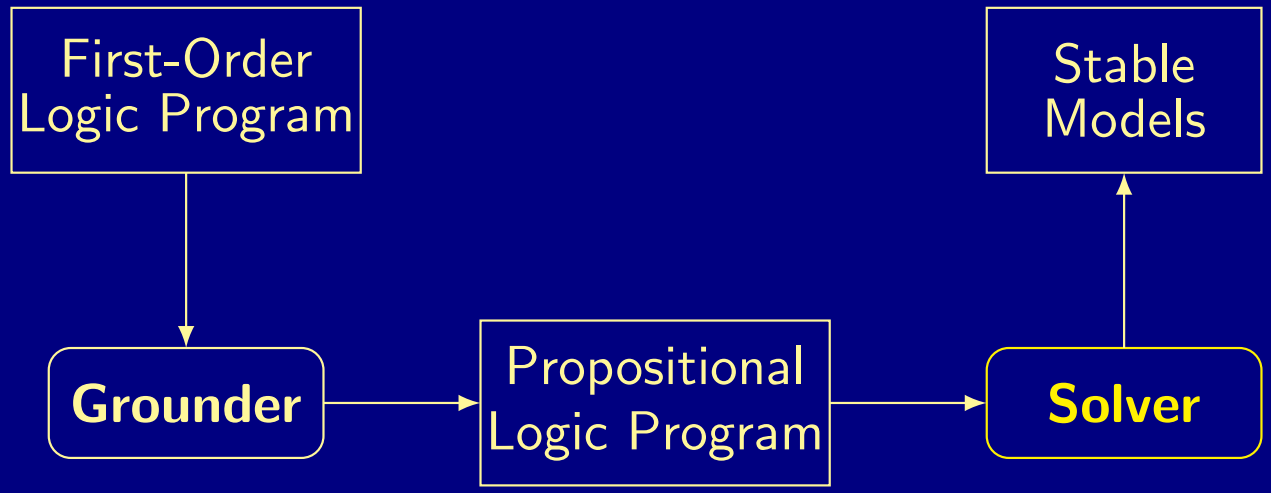


Expressive modeling language

Powerful grounding and solving tools

Potassco

The ASP Solving Process

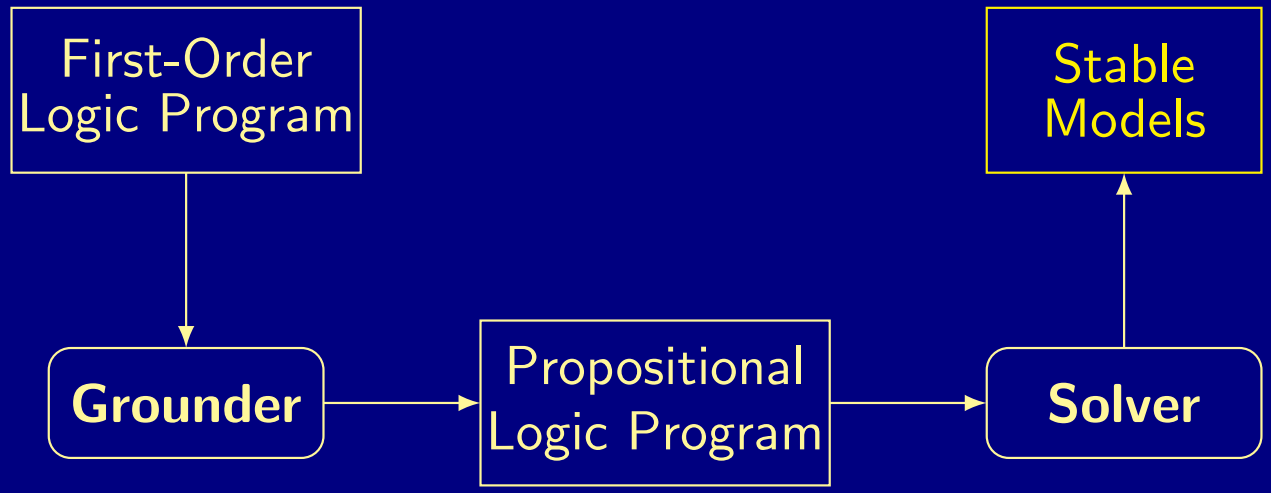


Expressive modeling language

Powerful grounding and solving tools

Potassco

The ASP Solving Process

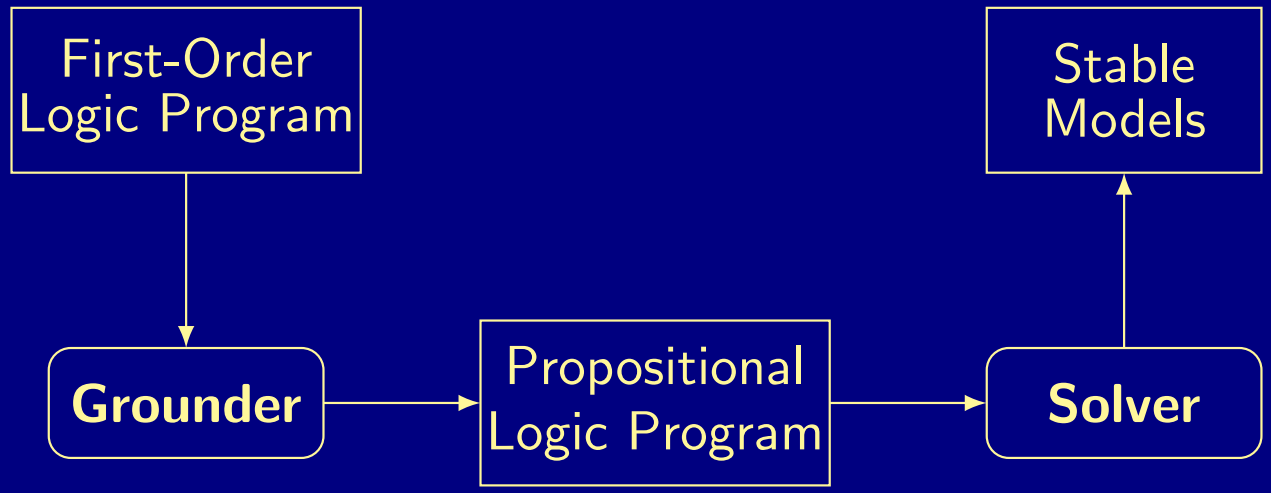


Expressive modeling language

Powerful grounding and solving tools

Potassco

The ASP Solving Process



- Expressive modeling language
- Powerful grounding and solving tools

Potassco

Outline

1 Introduction

2 Foundations

3 Modeling

4 Modeling and Controlling

5 Case-study: Ricochet Robots

6 Case-sketch: Preferences and optimization

7 Potassco

8 Summary

Potassco

Propositional Normal Logic Programs

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

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

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

- Semantics given by stable models, informally, sets X of atoms such that
  - X is a (classical) model of Π and
  - each atom in X is justified by some rule in Π

Potassco

Logic Programs

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

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

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

- Semantics given by stable models, informally, sets X of atoms such that
  - X is a (classical) model of Π and
  - each atom in X is justified by some rule in Π

Potassco

Logic Programs

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

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

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

- Semantics given by stable models, informally, sets X of atoms such that
  - X is a (classical) model of Π and
  - each atom in X is justified by some rule in Π

Potassco

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \quad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow (a \wedge \neg c) \vee y \quad y \leftarrow x \wedge b\} \cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $CF(\Pi)$:

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- Unsupported atoms
- Unfounded atoms

Potassco

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \quad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$RF(\Pi) = \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow (a \wedge \neg c) \vee y \quad y \leftarrow x \wedge b\}$$
$$\cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $RF(\Pi)$: (only true atoms shown)

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- Unsupported atoms
- Unfounded atoms

Potassco

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \quad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$RF(\Pi) = \{a \leftarrow \neg b \quad b \leftarrow \neg a \quad x \leftarrow (a \wedge \neg c) \vee y \quad y \leftarrow x \wedge b\}$$

$$\cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $RF(\Pi)$:

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- Unsupported atoms
- Unfounded atoms

Potassco

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \quad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\} \cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $RF(\Pi)$:

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- Unsupported atoms
- Unfounded atoms

Potassco

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \quad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\} \cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $CF(\Pi)$:

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- ~~Unsupported atoms~~
- Unfounded atoms

Potassco

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \quad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\} \cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $CF(\Pi)$:

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- ~~Unsupported atoms~~
- Unfounded atoms

Potassco

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \quad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow \neg b \quad b \leftrightarrow \neg a \quad x \leftrightarrow (a \wedge \neg c) \vee y \quad y \leftrightarrow x \wedge b\} \cup \{c \leftrightarrow \bot\}$$

$$LF(\Pi) = \{(x \vee y) \rightarrow a \wedge \neg c\}$$

Classical models of $CF(\Pi) \cup LF(\Pi)$:

$\{b\}$, $\{b, c\}$, $\{b, x, y\}$, $\{b, c, x, y\}$, $\{a, c\}$, $\{a, b, c\}$, $\{a, x\}$, $\{a, c, x\}$, $\{a, x, y\}$, $\{a, c, x, y\}$, $\{a, b, x, y\}$, $\{a, b, c, x, y\}$

- ~~Unsupported atoms~~
- ~~Unfounded atoms~~

Potassco

Logic Programs as Propositional Formulas

$$\Pi = \{a \leftarrow \sim b \quad b \leftarrow \sim a \quad x \leftarrow a, \sim c \quad x \leftarrow y \quad y \leftarrow x, b\}$$

$$CF(\Pi) = \{a \leftrightarrow (\bigvee_{(a \leftarrow B) \in \Pi} BF(B)) \mid a \in atom(\Pi)\}$$

$$BF(B) = \bigwedge_{b \in B \cap atom(\Pi)} b \wedge \bigwedge_{\sim c \in B} \neg c$$

$$LF(\Pi) = \{(\bigvee_{a \in L} a) \rightarrow (\bigvee_{a \in L, (a \leftarrow B) \in \Pi, B \cap L = \emptyset} BF(B)) \mid L \in loop(\Pi)\}$$

Classical models of $CF(\Pi) \cup LF(\Pi)$:

Theorem (Lin and Zhao)

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

- Size of $CF(\Pi)$ is linear in the size of Π
- Size of $LF(\Pi)$ may be exponential in the size of Π

Potassco

Outline

1 Introduction
2 Foundations
3 Modeling
4 Modeling and Controlling
5 Case-study: Ricochet Robots
6 Case-sketch: Preferences and optimization
7 Potassco
8 Summary

Potassco

Some language constructs

- Variables
  - `p(X) :- q(X)` over constants $\{\texttt{a}, \texttt{b}, \texttt{c}\}$ stands for
    `p(a) :- q(a), p(b) :- q(b), p(c) :- q(c)`
- Conditional Literals
  - `p :- q(X) : r(X)` given `r(a), r(b), r(c)` stands for
    `p :- q(a), q(b), q(c)`
- 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(Y) } 7`

Potassco

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)

Peanutshell

Logic program = Data + Generator + Tester (+ Optimizer)

Potassco

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)

Peanutshell

Logic program = Data + Generator + Tester (+ Optimizer)

Potassco

Satisfiability testing

$(a \leftrightarrow b) \wedge c$

Satisfiability testing

$(a \leftrightarrow b) \wedge c$

```
{ a ; b ; c }.

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

Potassco

Maximum satisfiability testing

"$(a \not\leftrightarrow b)$" $+ (a \leftrightarrow b) \wedge c$

```
{ a ; b ; c }.

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

 :~ a, b. [42@1]
 :~ not a, not b. [69@2]
```

Potassco

n-queens
Basic encoding

```
{ queen(1..n,1..n) }.

 :- { queen(I,J) } != n.
 :- queen(I,J), queen(I,JJ), J != JJ.
 :- queen(I,J), queen(II,J), I != II.
 :- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I-J = II-JJ.
 :- queen(I,J), queen(II,JJ), (I,J) != (II,JJ), I+J = II+JJ.
```

Potassco

n-queens

Advanced encoding

```
{ queen(I,1..n) } = 1 :- I = 1..n.
{ queen(1..n,J) } = 1 :- J = 1..n.

 :- { queen(D-J,J) } >= 2, D =   2..2*n.
 :- { queen(D+J,J) } >= 2, D = 1-n..n-1.
```

Potassco

n-queens

(Experimental) constraint encoding

```
1 $<= $queen(1..n) $<= n.

#disjoint { X : $queen(X) $+ 0 : X=1..n }.
#disjoint { X : $queen(X) $+ X : X=1..n }.
#disjoint { X : $queen(X) $- X : X=1..n }.
```

Potassco

Traveling salesperson
Basic encoding (no instance)

```
1 { cycle(X,Y) : edge(X,Y) } 1 :- node(X).
1 { cycle(X,Y) : edge(X,Y) } 1 :- node(Y).

reached(X) :- X = #min { Y : node(Y) }.
reached(Y) :- cycle(X,Y), reached(X).

:- node(Y), not reached(Y).

#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.
```

Potassco

Company Controls

```
controls(X,Y) :-
        #sum+ { S: owns(X,Y,S);
                S,Z: controls(X,Z), owns(Z,Y,S) } > 50,
        company(X), company(Y), X != Y.


company(c_1).    owns(c_1,c_2,60).
                 owns(c_1,c_3,20).
company(c_2).    owns(c_2,c_3,35).
company(c_3).    owns(c_3,c_4,51).
company(c_4).
```

Outline

1 Introduction

2 Foundations

3 Modeling

4 Modeling and Controlling

5 Case-study: Ricochet Robots

6 Case-sketch: Preferences and optimization

7 Potassco

8 Summary

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*

  Multi-shot solving: *ground* | *solve*

  ➥ **continuously changing logic programs**

Agents, Assisted Living, Robotics, Planning, Query-answering, etc

clingo 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*

  Multi-shot solving: *ground* | *solve*

  ➥ **continuously changing logic programs**

Agents, Assisted Living, Robotics, Planning, Query-answering, etc

clingo 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: *ground* | *solve*

  ➥ **continuously changing logic programs**

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: *ground* | *solve*

  ➥ **continuously changing logic programs**

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: *ground** | *solve**

  ➥ **continuously changing logic programs**

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: $\mathit{ground} \mid \mathit{solve}$
- Multi-shot solving: $(\mathit{ground}^* \mid \mathit{solve}^*)^*$

  ➥ **continuously changing logic programs**

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**)*

  ➥ **continuously changing logic programs**

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**| *theory*)*

  ➥ **continuously changing logic programs**

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**| *theory* | . . .)*

  ➥ **continuously changing logic programs**

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**| *theory* | . . .)*

  ➥ **continuously changing logic programs**

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**| *theory* | . . .)*

  ➥ **continuously changing logic programs**

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**| *theory* | . . .)*

  ➥ **continuously changing logic programs**

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

From One- to Multi-shot ASP solving

- Claim ASP is an under-the-hood technology

  That is, in practice, it mainly serves as a solving engine within an encompassing software environment

- Single-shot solving: *ground* | *solve*
- Multi-shot solving: (*input* | *ground** | *solve**| *theory* | . . .)*

  ➥ **continuously changing logic programs**

- Application areas

  Agents, Assisted Living, Robotics, Planning, Query-answering, etc

- Implementation *clingo* 4 — providing operative solving processes dealing with continously changing logic programs

Potassco

Clingo = ASP + Control

- ASP

```
#program <name> [ (<parameters>) ]
          #program play(t).
#external <atom> [ :  <body> ]
          #external mark(X,Y,P,t) :  field(X,Y), player(P).
```

- Control

Lua (www.lua.org)

```
          prg:solve(), prg:ground(parts), ...
```

Python (www.python.org)

```
          prg.solve(), prg.ground(parts), ...
```

- Integration

in ASP: embedded scripting language (#script)
in Lua/Python: library import (import gringo)

Potassco

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    `prg.solve(), prg.ground(parts), ...`
- Integration
  - in ASP: embedded scripting language (`#script`)
  - in Lua/Python: library import (`import gringo`)

Potassco

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    `prg.solve(), prg.ground(parts), ...`
- Integration
  - in ASP: embedded scripting language (`#script`)
  - in Lua/Python: library import (`import gringo`)

Potassco

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    - Example `prg.solve(), prg.ground(parts), ...`
- Integration

  in ASP: embedded scripting language (`#script`)
  in Lua/Python: library import (`import gringo`)

Potassco

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    - Example `prg.solve(), prg.ground(parts), ...`
- Integration

  in ASP: embedded scripting language (`#script`)
  in Lua/Python: library import (`import gringo`)

Potassco

Clingo = ASP + Control

- ASP
  - `#program <name> [ (<parameters>) ]`
    - Example `#program play(t).`
  - `#external <atom> [ : <body> ]`
    - Example `#external mark(X,Y,P,t) : field(X,Y), player(P).`
- Control
  - Lua (`www.lua.org`)
    - Example `prg:solve(), prg:ground(parts), ...`
  - Python (`www.python.org`)
    - Example `prg.solve(), prg.ground(parts), ...`
- Integration
  - in ASP: embedded scripting language (`#script`)
  - in Lua/Python: library import (`import gringo`)

Potassco

Vanilla *Clingo*

- Emulating *Clingo* in *Clingo* 4

```
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```

Potassco

Vanilla *Clingo*

- Emulating *Clingo* in *Clingo* 4

```
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```

Potassco

Vanilla *Clingo*

- Emulating *Clingo* in *Clingo* 4

```
#script (python)
def main(prg):
    parts = []
    parts.append(("base", []))
    prg.ground(parts)
    prg.solve()
#end.
```

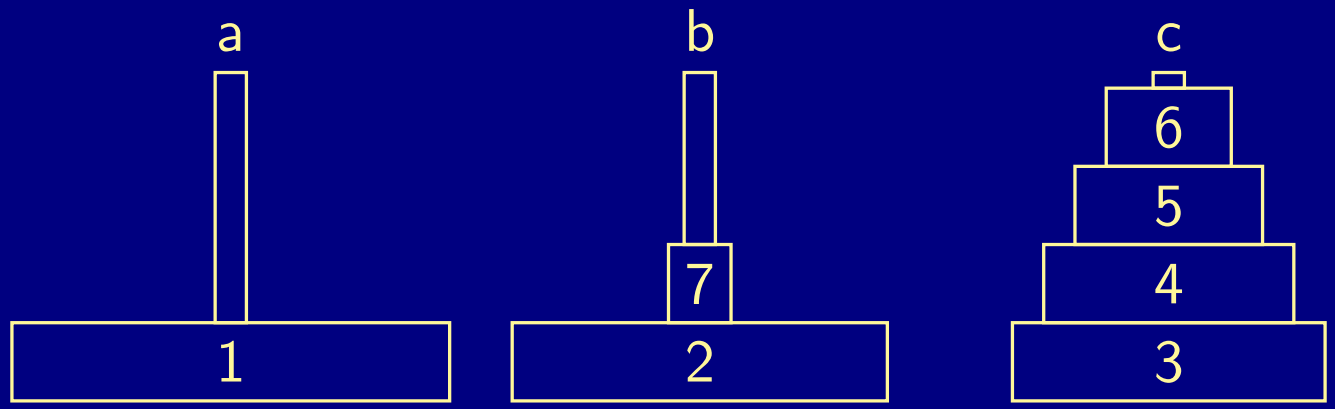
Potassco

Towers of Hanoi Instance

- Emulating *iClingo* (an incremental ASP solver) in *Clingo* 4
  - Incremental grounding
  - Incremental solving

Potassco

Towers of Hanoi Instance




```
peg(a;b;c).
init_on(1,a).
init_on((2;7),b).
init_on((3;4;5;6),c).
```

```
disk(1..7).

goal_on((3;4),a).
goal_on((1;2;5;6;7),c).
```

Potassco

Towers of Hanoi Encoding (base)

```
#program base.

on(D,P,0) :- init_on(D,P).
```

Towers of Hanoi Encoding (cumulative)

```
#program cumulative(t).

1 { move(D,P,t) : disk(D), peg(P) } 1.

moved(D,t) :- move(D,_,t).
blocked(D,P,t) :- on(D+1,P,t-1), disk(D).
blocked(D,P,t) :- blocked(D+1,P,t), disk(D).
:- move(D,P,t), blocked(D-1,P,t).
:- moved(D,t), on(D,P,t-1), blocked(D,P,t).

on(D,P,t) :- on(D,P,t-1), not moved(D,t).
on(D,P,t) :- move(D,P,t).
:- not 1 { on(D,P,t) : peg(P) } 1, disk(D).
```

Potassco

Towers of Hanoi Encoding (volatile)

```
#program volatile(t).
#external query(t).

:- goal_on(D,P), not on(D,P,t), query(t).
```

Exercising control

An `#external atom` can be controlled from Python (or Lua) via

- `assign_external(self,atom,value)`
  where `value` is either `True`, `False`, or `None`
- `release_external(self,atom)`
  sets `atom` permanently to `False`

Potassco

Towers of Hanoi Encoding (volatile)

```
#program volatile(t).
#external query(t).

:- goal_on(D,P), not on(D,P,t), query(t).
```

Exercising control

An `#external atom` can be controlled from Python (or Lua) via

- `assign_external(self,atom,value)`
  where `value` is either `True`, `False`, or `None`
- `release_external(self,atom)`
  sets `atom` permanently to `False`

Potassco

Incremental Solving (embedded)

```
#script (python)

from gringo import SolveResult, Fun

def main(prg):
    ret, parts, step = SolveResult.UNSAT, [], 1
    parts.append(("base", []))
    while ret == SolveResult.UNSAT:
        parts.append(("cumulative", [step]))
        parts.append(("volatile", [step]))
        prg.ground(parts)
        prg.release_external(Fun("query", [step-1]))
        prg.assign_external(Fun("query", [step]), True)
        ret, parts, step = prg.solve(), [], step+1

#end.
```

Potassco

Incremental Solving (library)

```
from sys import stdout
from gringo import SolveResult, Fun, Control

prg = Control()
prg.load("toh.lp")

ret, parts, step = SolveResult.UNSAT, [], 1
parts.append(("base", []))
while ret == SolveResult.UNSAT:
    parts.append(("cumulative", [step]))
    parts.append(("volatile", [step]))
    prg.ground(parts)
    prg.release_external(Fun("query", [step-1]))
    prg.assign_external(Fun("query", [step]), True)
    f = lambda m: stdout.write(str(m))
    ret, parts, step = prg.solve(on_model=f), [], step+1
```

Potassco

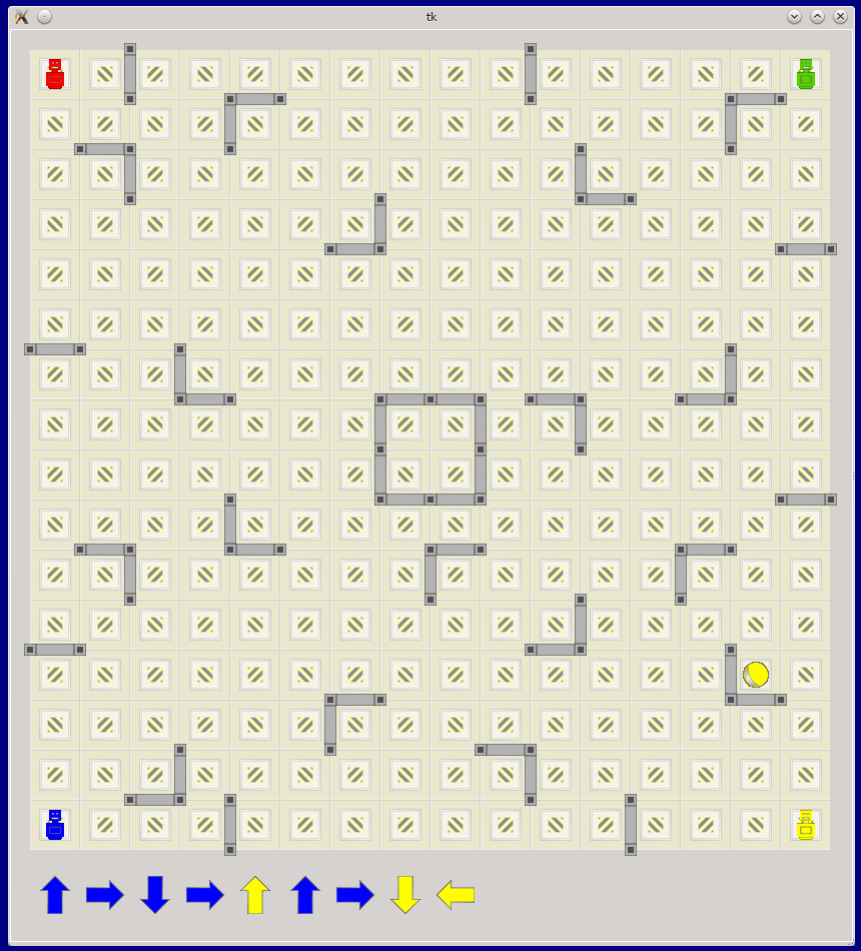
Outline

1 Introduction
2 Foundations
3 Modeling
4 Modeling and Controlling
5 Case-study: Ricochet Robots
6 Case-sketch: Preferences and optimization
7 Potassco
8 Summary

Potassco

Alex Rudolph's Ricochet Robots

Solving goal(13) from cornered robots



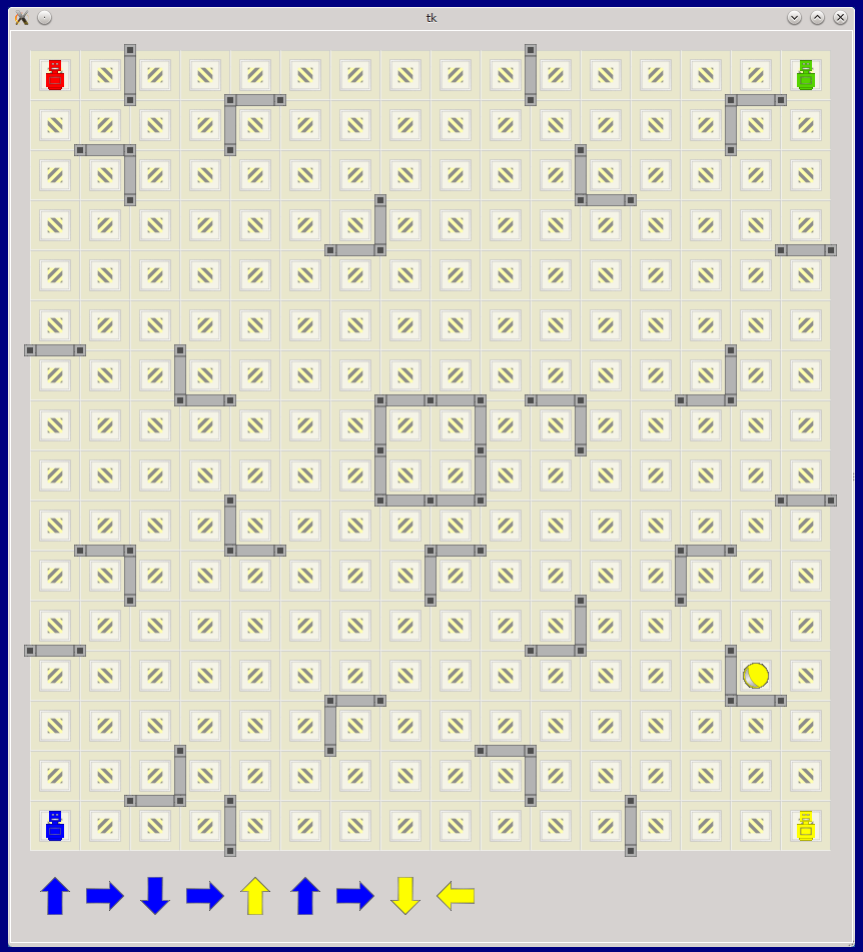
- Four robots roaming
  - horizontally
  - vertically

  up to blocking objects, ricocheting (optionally)
- Goal Robot on target (sharing same color)

Potassco

Alex Rudolph's Ricochet Robots

Solving `goal(13)` from cornered robots



- Four robots roaming
  - horizontally
  - vertically

  up to blocking objects,

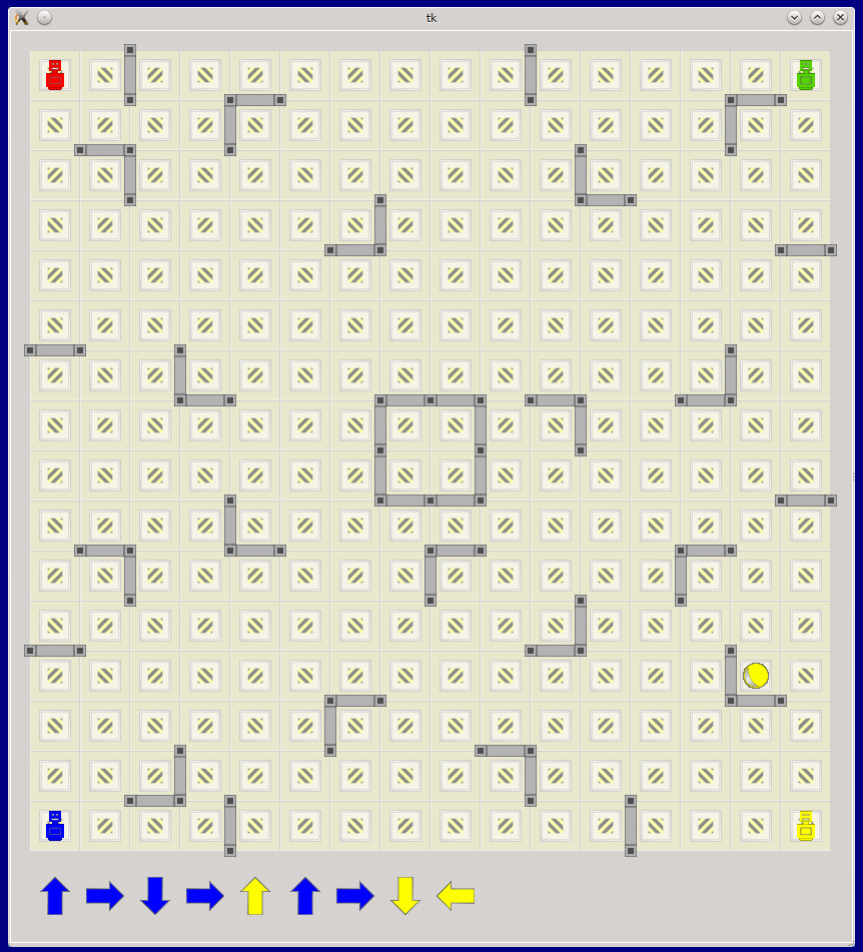
  ricocheting (optionally)

- Goal Robot on target (sharing same color)

Potassco

Alex Rudolph's Ricochet Robots

Solving `goal(13)` from cornered robots



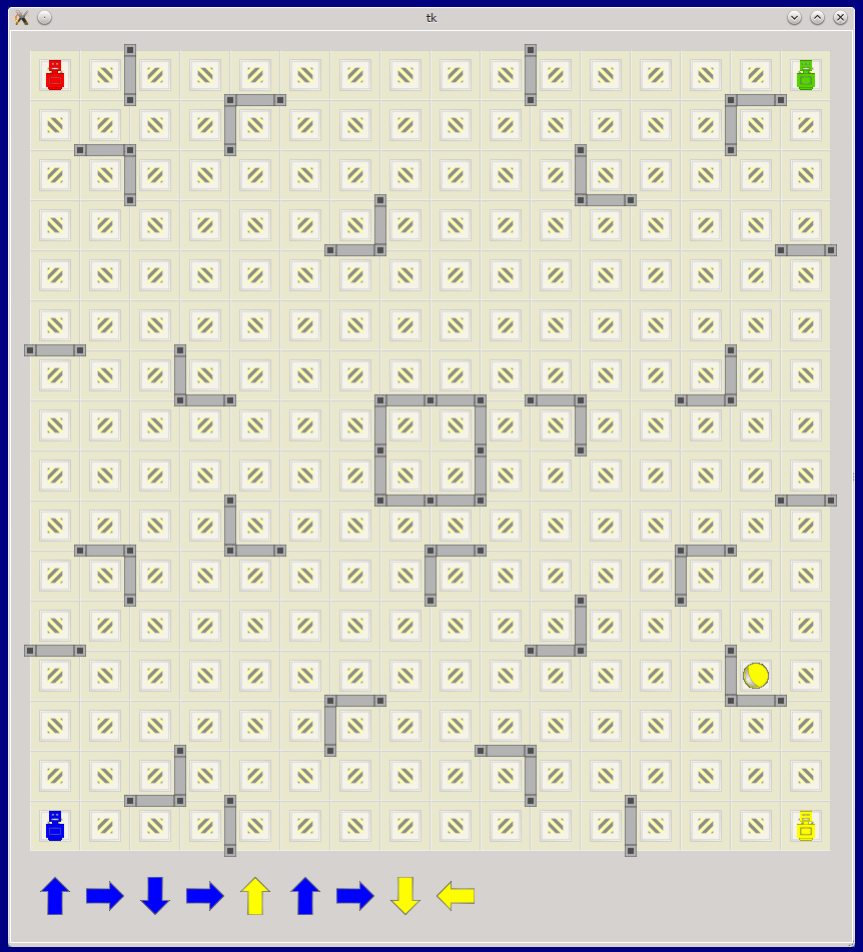
- Four robots roaming
  - horizontally
  - vertically

  up to blocking objects, ricocheting (optionally)
- Goal Robot on target (sharing same color)

Potassco

Alex Rudolph's Ricochet Robots

Solving `goal(13)` from cornered robots

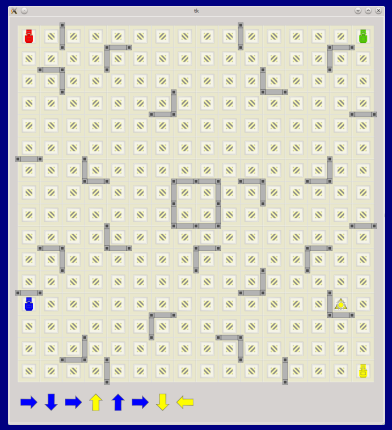


- Four robots roaming
  - horizontally
  - vertically

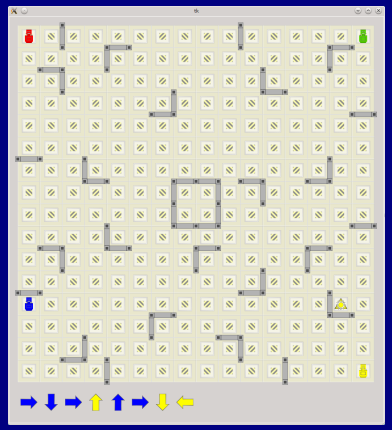
  up to blocking objects, ricocheting (optionally)
- Goal Robot on target (sharing same color)

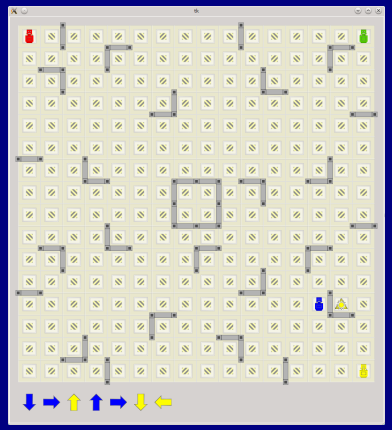
Potassco

Solving `goal(13)` from cornered robots (ctd)

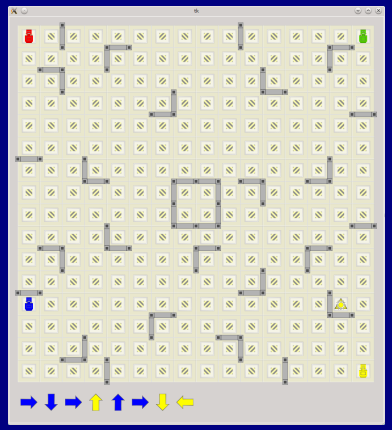
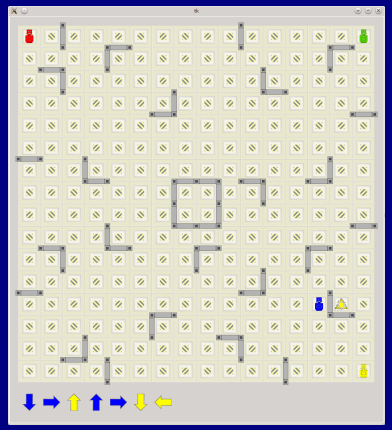
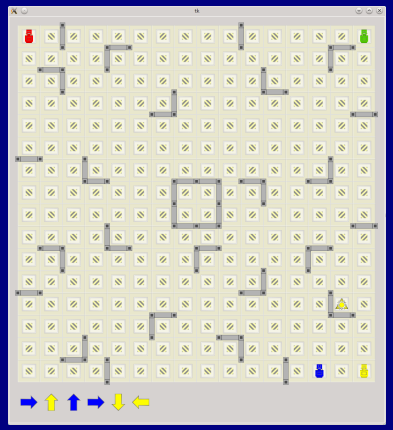


Solving `goal(13)` from cornered robots (ctd)

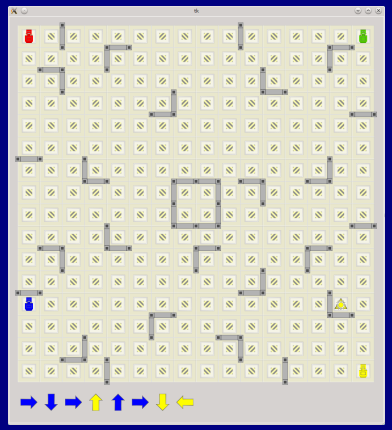
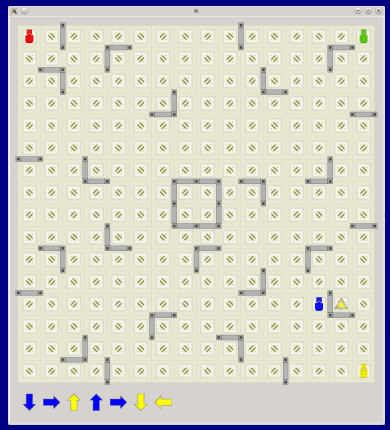
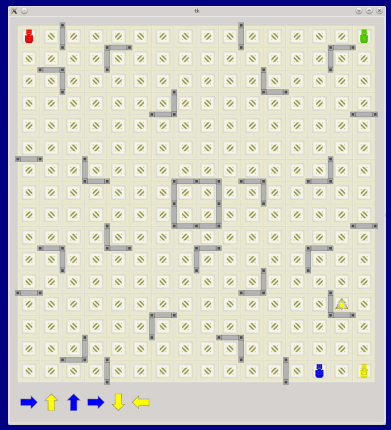
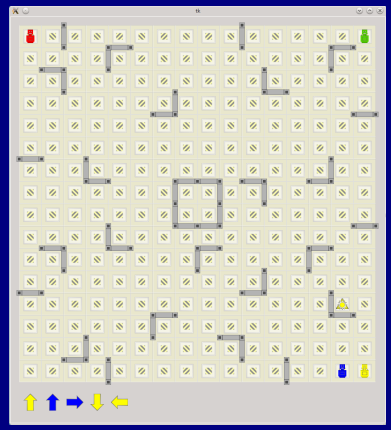




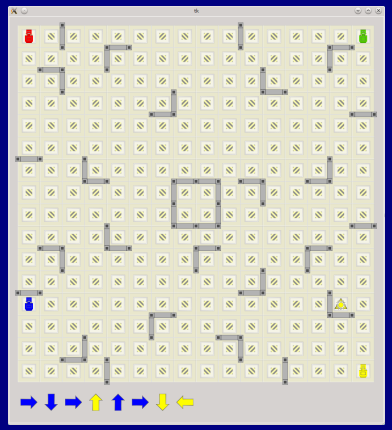
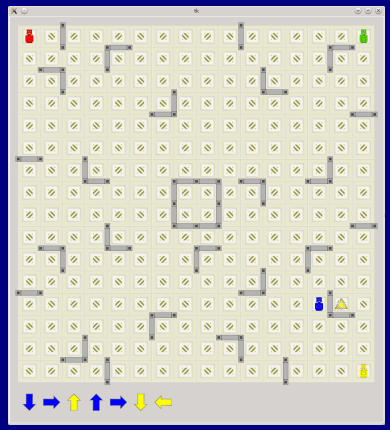
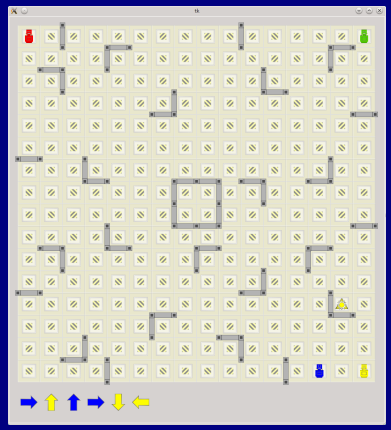
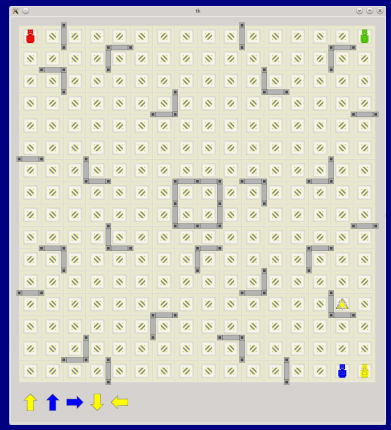
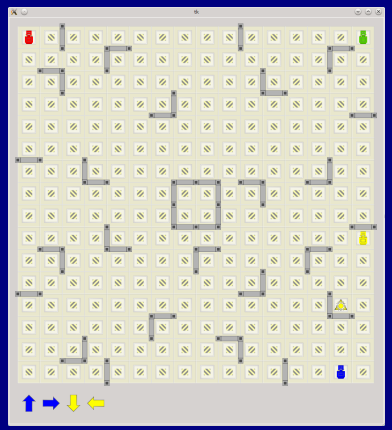
Solving `goal(13)` from cornered robots (ctd)

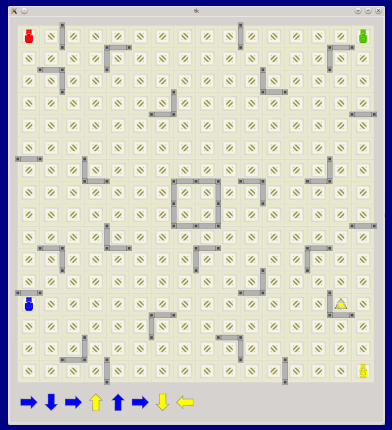
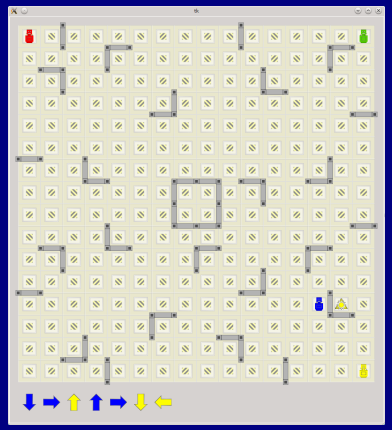
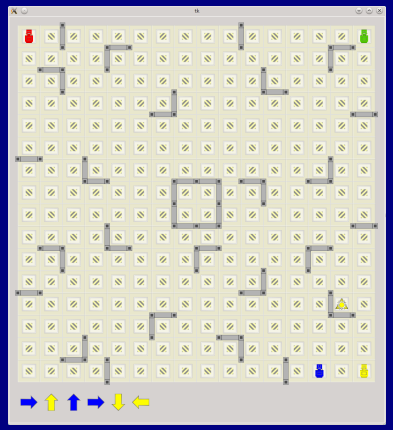
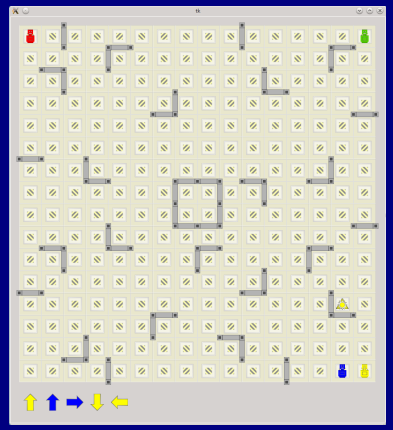
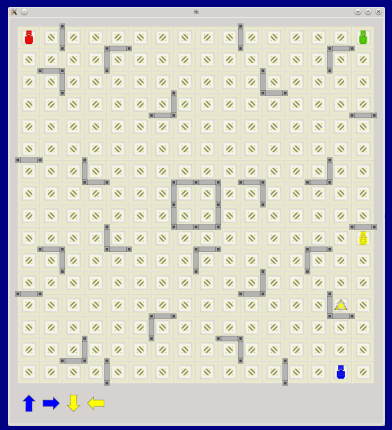
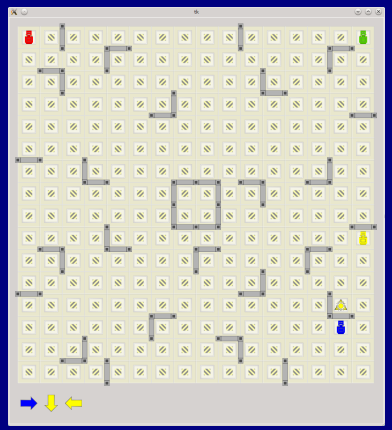
Solving `goal(13)` from cornered robots (ctd)

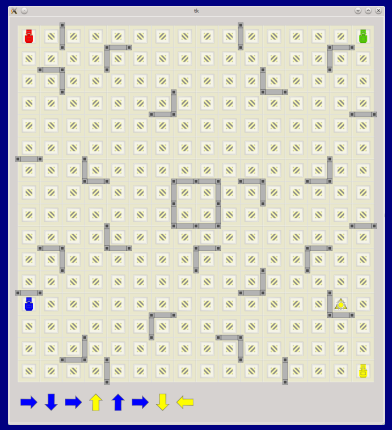
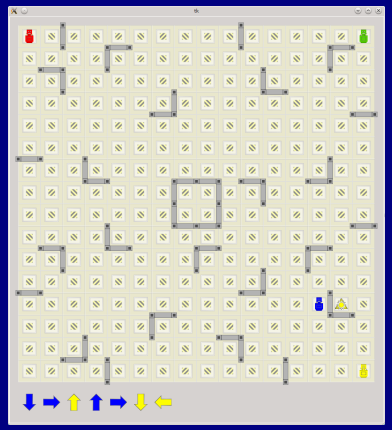
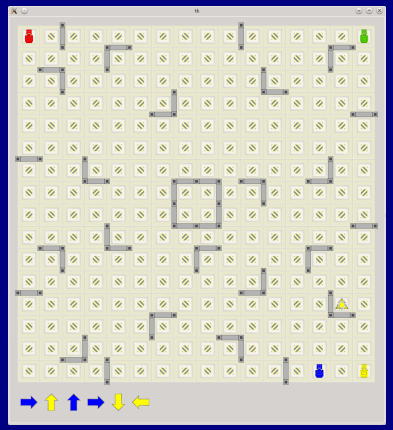
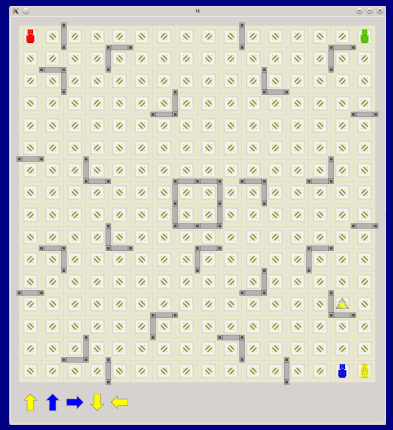
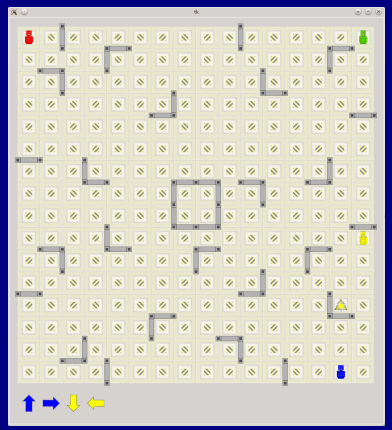
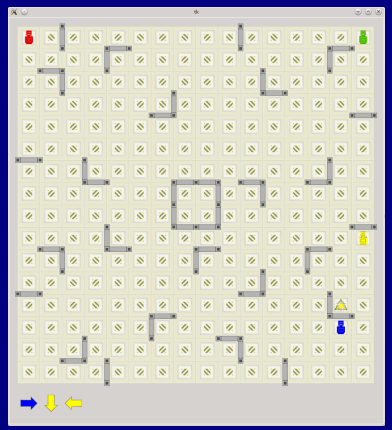
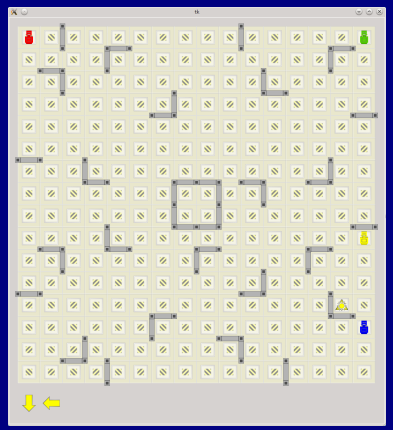
Solving `goal(13)` from cornered robots (ctd)

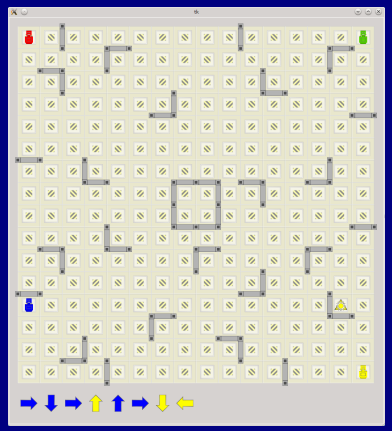
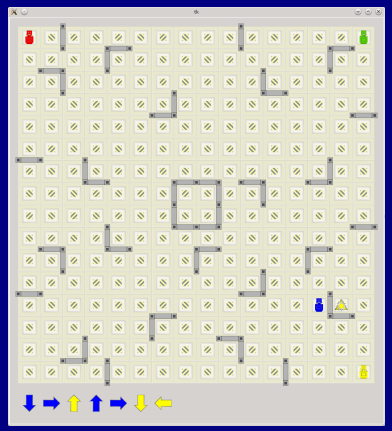
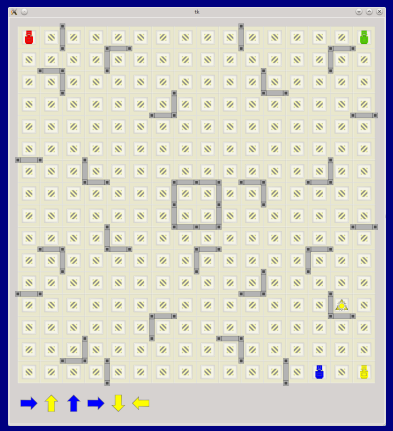
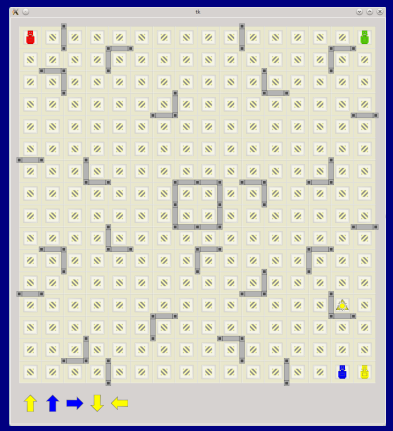
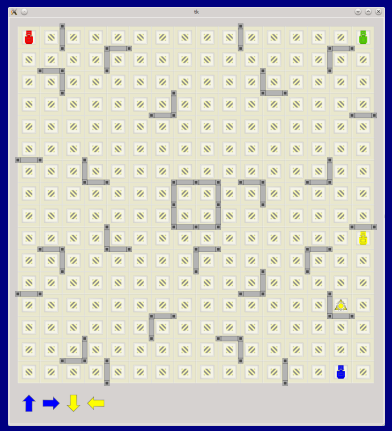
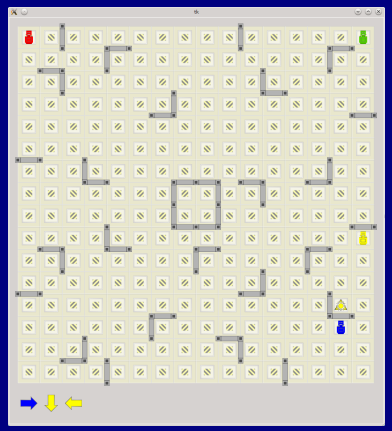
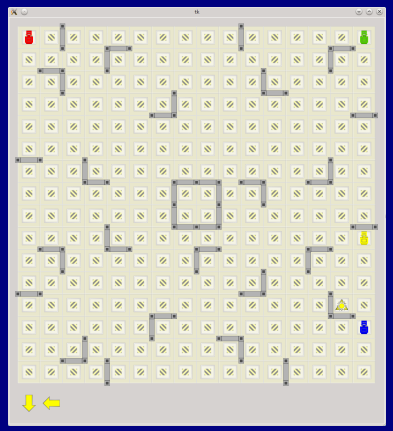
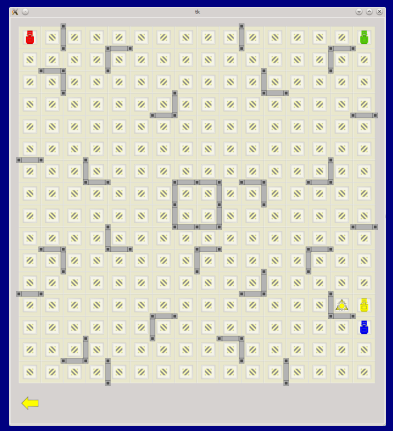
Solving `goal(13)` from cornered robots (ctd)

Solving `goal(13)` from cornered robots (ctd)

Solving `goal(13)` from cornered robots (ctd)

Potassco

board.lp

```
dim(1..16).

barrier( 2, 1, 1, 0).  barrier(13,11, 1, 0).  barrier( 9, 7, 0, 1).
barrier(10, 1, 1, 0).  barrier(11,12, 1, 0).  barrier(11, 7, 0, 1).
barrier( 4, 2, 1, 0).  barrier(14,13, 1, 0).  barrier(14, 7, 0, 1).
barrier(14, 2, 1, 0).  barrier( 6,14, 1, 0).  barrier(16, 9, 0, 1).
barrier( 2, 3, 1, 0).  barrier( 3,15, 1, 0).  barrier( 2,10, 0, 1).
barrier(11, 3, 1, 0).  barrier(10,15, 1, 0).  barrier( 5,10, 0, 1).
barrier( 7, 4, 1, 0).  barrier( 4,16, 1, 0).  barrier( 8,10, 0,-1).
barrier( 3, 7, 1, 0).  barrier(12,16, 1, 0).  barrier( 9,10, 0,-1).
barrier(14, 7, 1, 0).  barrier( 5, 1, 0, 1).  barrier( 9,10, 0, 1).
barrier( 7, 8, 1, 0).  barrier(15, 1, 0, 1).  barrier(14,10, 0, 1).
barrier(10, 8,-1, 0).  barrier( 2, 2, 0, 1).  barrier( 1,12, 0, 1).
barrier(11, 8, 1, 0).  barrier(12, 3, 0, 1).  barrier(11,12, 0, 1).
barrier( 7, 9, 1, 0).  barrier( 7, 4, 0, 1).  barrier( 7,13, 0, 1).
barrier(10, 9,-1, 0).  barrier(16, 4, 0, 1).  barrier(15,13, 0, 1).
barrier( 4,10, 1, 0).  barrier( 1, 6, 0, 1).  barrier(10,14, 0, 1).
barrier( 2,11, 1, 0).  barrier( 4, 7, 0, 1).  barrier( 3,15, 0, 1).
barrier( 8,11, 1, 0).  barrier( 8, 7, 0, 1).
```

Potassco

targets.lp

```
#external goal(1..16).

target(red,    5, 2) :- goal(1).
target(red,   15, 2) :- goal(2).
target(green,  2, 3) :- goal(3).
target(blue,  12, 3) :- goal(4).
target(yellow, 7, 4) :- goal(5).
target(blue,   4, 7) :- goal(6).
target(green, 14, 7) :- goal(7).
target(yellow,11, 8) :- goal(8).
target(yellow, 5,10) :- goal(9).
target(green,  2,11) :- goal(10).
target(red,   14,11) :- goal(11).
target(green, 11,12) :- goal(12).
target(yellow,15,13) :- goal(13).
target(blue,   7,14) :- goal(14).
target(red,    3,15) :- goal(15).
target(blue,  10,15) :- goal(16).

robot(red;green;blue;yellow).
#external pos((red;green;blue;yellow),1..16,1..16).
```

Potassco

ricochet.lp

```
time(1..horizon).
dir(-1,0;1,0;0,-1;0,1).

stop( DX, DY,X,   Y   ) :- barrier(X,Y,DX,DY).
stop(-DX,-DY,X+DX,Y+DY) :- stop(DX,DY,X,Y).

pos(R,X,Y,0) :- pos(R,X,Y).

1 { move(R,DX,DY,T) : robot(R), dir(DX,DY) } 1 :- time(T).
move(R,T) :- move(R,_,_,T).

halt(DX,DY,X-DX,Y-DY,T) :- pos(_,X,Y,T), dir(DX,DY), dim(X-DX), dim(Y-DY),
                           not stop(-DX,-DY,X,Y), T < horizon.

goto(R,DX,DY,X,Y,T) :- pos(R,X,Y,T), dir(DX,DY), T < horizon.
goto(R,DX,DY,X+DX,Y+DY,T) :- goto(R,DX,DY,X,Y,T), dim(X+DX), dim(Y+DY),
                         not stop(DX,DY,X,Y), not halt(DX,DY,X,Y,T).

pos(R,X,Y,T) :- move(R,DX,DY,T), goto(R,DX,DY,X,Y,T-1),
                not goto(R,DX,DY,X+DX,Y+DY,T-1).
pos(R,X,Y,T) :- pos(R,X,Y,T-1), time(T), not move(R,T).

:- target(R,X,Y), not pos(R,X,Y,horizon).

#show move/4.
```

Potassco

Solving `goal(13)` from cornered robots

```
$ clingo board.lp targets.lp ricochet.lp -c horizon=9 \
       <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16).   goal(13).")

clingo version 4.5.0
Reading from board.lp ...
Solving...
Answer: 1
move(red,0,1,1)     move(red,1,0,2) move(red,0,1,3)    move(red,-1,0,4) move(red,0,1,5) \
move(yellow,0,-1,6) move(red,1,0,7) move(yellow,0,1,8) move(yellow,-1,0,9)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 1.895s (Solving: 1.45s 1st Model: 1.45s Unsat: 0.00s)
CPU Time     : 1.880s


$ clingo board.lp targets.lp ricochet.lp -c horizon=8 \
       <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16).   goal(13).")

clingo version 4.5.0
Reading from board.lp ...
Solving...
UNSATISFIABLE

Models       : 0
Calls        : 1
Time         : 2.817s (Solving: 2.41s 1st Model: 0.00s Unsat: 2.41s)
CPU Time     : 2.800s
```

Potassco

Solving `goal(13)` from cornered robots

```
$ clingo board.lp targets.lp ricochet.lp -c horizon=9 \
       <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16).   goal(13).")

clingo version 4.5.0
Reading from board.lp ...
Solving...
Answer: 1
move(red,0,1,1)     move(red,1,0,2) move(red,0,1,3)    move(red,-1,0,4) move(red,0,1,5) \
move(yellow,0,-1,6) move(red,1,0,7) move(yellow,0,1,8) move(yellow,-1,0,9)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 1.895s (Solving: 1.45s 1st Model: 1.45s Unsat: 0.00s)
CPU Time     : 1.880s


$ clingo board.lp targets.lp ricochet.lp -c horizon=8 \
       <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16).   goal(13).")

clingo version 4.5.0
Reading from board.lp ...
Solving...
UNSATISFIABLE

Models       : 0
Calls        : 1
Time         : 2.817s (Solving: 2.41s 1st Model: 0.00s Unsat: 2.41s)
CPU Time     : 2.800s
```

Potassco

Solving `goal(13)` from cornered robots

```
$ clingo board.lp targets.lp ricochet.lp -c horizon=9 \
        <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16).   goal(13).")

clingo version 4.5.0
Reading from board.lp ...
Solving...
Answer: 1
move(red,0,1,1)     move(red,1,0,2) move(red,0,1,3)    move(red,-1,0,4) move(red,0,1,5) \
move(yellow,0,-1,6) move(red,1,0,7) move(yellow,0,1,8) move(yellow,-1,0,9)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 1.895s (Solving: 1.45s 1st Model: 1.45s Unsat: 0.00s)
CPU Time     : 1.880s


$ clingo board.lp targets.lp ricochet.lp -c horizon=8 \
        <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16).   goal(13).")

clingo version 4.5.0
Reading from board.lp ...
Solving...
UNSATISFIABLE

Models       : 0
Calls        : 1
Time         : 2.817s (Solving: 2.41s 1st Model: 0.00s Unsat: 2.41s)
CPU Time     : 2.800s
```

Potassco

Solving `goal(13)` from cornered robots

```
$ clingo board.lp targets.lp ricochet.lp -c horizon=9 \
        <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16).   goal(13).")

clingo version 4.5.0
Reading from board.lp ...
Solving...
Answer: 1
move(red,0,1,1)     move(red,1,0,2) move(red,0,1,3)    move(red,-1,0,4) move(red,0,1,5) \
move(yellow,0,-1,6) move(red,1,0,7) move(yellow,0,1,8) move(yellow,-1,0,9)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 1.895s (Solving: 1.45s 1st Model: 1.45s Unsat: 0.00s)
CPU Time     : 1.880s


$ clingo board.lp targets.lp ricochet.lp -c horizon=8 \
        <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16).   goal(13).")

clingo version 4.5.0
Reading from board.lp ...
Solving...
UNSATISFIABLE

Models       : 0
Calls        : 1
Time         : 2.817s (Solving: 2.41s 1st Model: 0.00s Unsat: 2.41s)
CPU Time     : 2.800s
```

Potassco

optimization.lp

```
goon(T) :- target(R,X,Y), T = 0..horizon, not pos(R,X,Y,T).

:- move(R,DX,DY,T-1), time(T), not goon(T-1), not move(R,DX,DY,T).

#minimize{ 1,T : goon(T) }.
```

Potassco

Solving `goal(13)` from cornered robots

```
$ clingo board.lp targets.lp ricochet.lp optimization.lp -c horizon=20 --quiet=1,0 \
        <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16).   goal(13).")
clingo version 4.5.0
Reading from board.lp ...
Solving...
Optimization: 20
Optimization: 19
Optimization: 18
Optimization: 17
Optimization: 16
Optimization: 15
Optimization: 14
Optimization: 13
Optimization: 12
Optimization: 11
Optimization: 10
Optimization: 9
Answer: 12
move(blue,0,-1,1)   move(blue,1,0,2)    move(yellow,0,-1,3) move(blue,0,1,4)    move(yellow,-1,0,5) \
move(blue,1,0,6)    move(blue,0,-1,7)   move(yellow,1,0,8)  move(yellow,0,1,9)  move(yellow,0,1,10) \
move(yellow,0,1,11) move(yellow,0,1,12) move(yellow,0,1,13) move(yellow,0,1,14) move(yellow,0,1,15) \
move(yellow,0,1,16) move(yellow,0,1,17) move(yellow,0,1,18) move(yellow,0,1,19) move(yellow,0,1,20)
OPTIMUM FOUND

Models       : 12
  Optimum    : yes
Optimization : 9
Calls        : 1
Time         : 16.145s (Solving: 15.01s 1st Model: 3.35s Unsat: 2.02s)
CPU Time     : 16.080s
```

Potassco

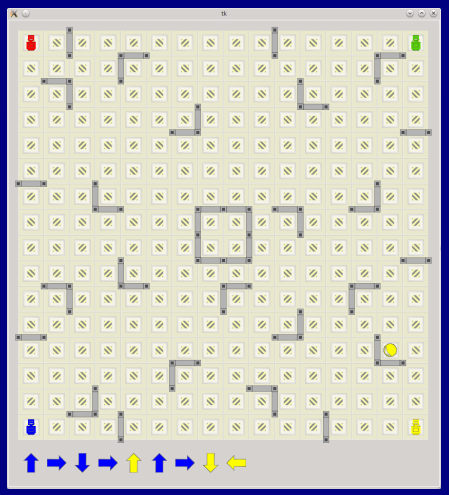
Solving `goal(13)` from cornered robots

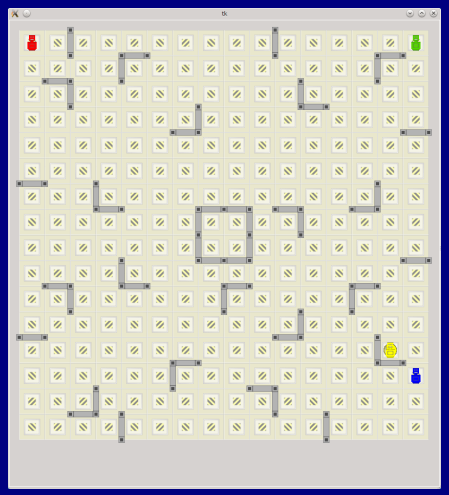
```
$ clingo board.lp targets.lp ricochet.lp optimization.lp -c horizon=20 --quiet=1,0 \
        <(echo "pos(red,1,1). pos(green,16,1). pos(blue,1,16). pos(yellow,16,16).   goal(13).")
clingo version 4.5.0
Reading from board.lp ...
Solving...
Optimization: 20
Optimization: 19
Optimization: 18
Optimization: 17
Optimization: 16
Optimization: 15
Optimization: 14
Optimization: 13
Optimization: 12
Optimization: 11
Optimization: 10
Optimization: 9
Answer: 12
move(blue,0,-1,1)   move(blue,1,0,2)    move(yellow,0,-1,3) move(blue,0,1,4)    move(yellow,-1,0,5) \
move(blue,1,0,6)    move(blue,0,-1,7)   move(yellow,1,0,8)  move(yellow,0,1,9)  move(yellow,0,1,10) \
move(yellow,0,1,11) move(yellow,0,1,12) move(yellow,0,1,13) move(yellow,0,1,14) move(yellow,0,1,15) \
move(yellow,0,1,16) move(yellow,0,1,17) move(yellow,0,1,18) move(yellow,0,1,19) move(yellow,0,1,20)
OPTIMUM FOUND

Models       : 12
  Optimum    : yes
Optimization : 9
Calls        : 1
Time         : 16.145s (Solving: 15.01s 1st Model: 3.35s Unsat: 2.02s)
CPU Time     : 16.080s
```

Potassco

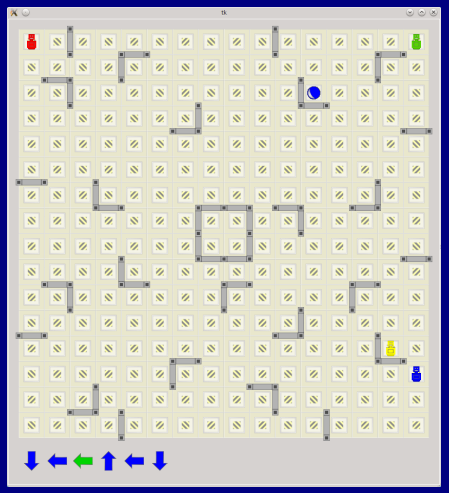
Playing in rounds

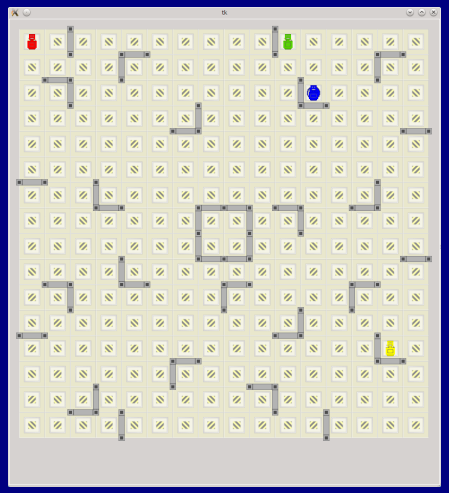
Round 1: `goal(13)`





Round 2: `goal(4)`





Potassco

Control loop

1. Create an operational *clingo* object
2. Load and ground the logic programs encoding Ricochet Robot (relative to some fixed `horizon`) within the control object
3. While there is a goal, do the following
   1. Enforce the initial robot positions
   2. Enforce the current goal
   3. Solve the logic program contained in the control object

Potassco

Ricochet Robot Player

ricochet.py

```
from gringo import Control, Model, Fun

class Player:
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
        self.ctl.ground([("base", [])])

    def solve(self, goal):
        for x in self.undo_external:
            self.ctl.assign_external(x, False)
        self.undo_external = []
        for x in self.last_positions + [goal]:
            self.ctl.assign_external(x, True)
            self.undo_external.append(x)
        self.last_solution = None
        self.ctl.solve(on_model=self.on_model)
        return self.last_solution

    def on_model(self, model):
        self.last_solution = model.atoms()
        self.last_positions = []
        for atom in model.atoms(Model.ATOMS):
            if (atom.name() == "pos" and len(atom.args()) == 4 and atom.args()[3] == self.horizon):
                self.last_positions.append(Fun("pos", atom.args()[:-1]))

horizon   = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"),     1,  1]), Fun("pos", [Fun("blue"),    1, 16]),
             Fun("pos", [Fun("green"),  16,  1]), Fun("pos", [Fun("yellow"), 16, 16])]
sequence  = [Fun("goal", [13]), Fun("goal", [4]), Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
```

Potassco

Variables of interest

- `last_positions` holds the starting positions of the robots for each turn
- `last_solution` holds the last solution of a search call
  (Note that callbacks cannot return values directly)
- `undo_external` holds a list containing the current goal and starting positions to be cleared upon the next step
- `horizon` holds the maximum number of moves to find a solution
- `ctl` holds the actual object providing an interface to the grounder and solver; it holds all state information necessary for multi-shot solving

Potassco

Variables of interest

- `last_positions` holds the starting positions of the robots for each turn
- `last_solution` holds the last solution of a search call
  (Note that callbacks cannot return values directly)
- `undo_external` holds a list containing the current goal and starting positions to be cleared upon the next step
- `horizon` holds the maximum number of moves to find a solution
- `ctl` holds the actual object providing an interface to the grounder and solver; it holds all state information necessary for multi-shot solving

Potassco

Variables of interest

- `last_positions` holds the starting positions of the robots for each turn
- `last_solution` holds the last solution of a search call
  (Note that callbacks cannot return values directly)
- `undo_external` holds a list containing the current goal and starting positions to be cleared upon the next step
- `horizon` holds the maximum number of moves to find a solution
- `ctl` holds the actual object providing an interface to the grounder and solver; it holds all state information necessary for multi-shot solving

Potassco

Variables of interest

- `last_positions` holds the starting positions of the robots for each turn
- `last_solution` holds the last solution of a search call
  (Note that callbacks cannot return values directly)
- `undo_external` holds a list containing the current goal and starting positions to be cleared upon the next step
- `horizon` holds the maximum number of moves to find a solution
- `ctl` holds the actual object providing an interface to the grounder and solver; it holds all state information necessary for multi-shot solving

Potassco

Variables of interest

- `last_positions` holds the starting positions of the robots for each turn
- `last_solution` holds the last solution of a search call
  (Note that callbacks cannot return values directly)
- `undo_external` holds a list containing the current goal and starting positions to be cleared upon the next step
- `horizon` holds the maximum number of moves to find a solution
- `ctl` holds the actual object providing an interface to the grounder and solver; it holds all state information necessary for multi-shot solving

Potassco

Variables of interest

- `last_positions` holds the starting positions of the robots for each turn
- `last_solution` holds the last solution of a search call
  (Note that callbacks cannot return values directly)
- `undo_external` holds a list containing the current goal and starting positions to be cleared upon the next step
- `horizon` holds the maximum number of moves to find a solution
- `ctl` holds the actual object providing an interface to the grounder and solver; it holds all state information necessary for multi-shot solving

Potassco

Ricochet Robot Player
Setup and control loop

```
from gringo import Control, Model, Fun

class Player:
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
        self.ctl.ground([("base", [])])

    def solve(self, goal):
        for x in self.undo_external:
            self.ctl.assign_external(x, False)
        self.undo_external = []
        for x in self.last_positions + [goal]:
            self.ctl.assign_external(x, True)
            self.undo_external.append(x)
        self.last_solution = None
        self.ctl.solve(on_model=self.on_model)
        return self.last_solution

    def on_model(self, model):
        self.last_solution = model.atoms()
        self.last_positions = []
        for atom in model.atoms(Model.ATOMS):
            if (atom.name() == "pos" and len(atom.args()) == 4 and atom.args()[3] == self.horizon):
                self.last_positions.append(Fun("pos", atom.args()[:-1]))

horizon   = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"),     1,  1]), Fun("pos", [Fun("blue"),    1, 16]),
             Fun("pos", [Fun("green"),  16,  1]), Fun("pos", [Fun("yellow"), 16, 16])]
sequence  = [Fun("goal", [13]), Fun("goal", [4]), Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
```

Potassco

Setup and control loop

```
horizon   = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"),     1,  1]),
             Fun("pos", [Fun("blue"),    1, 16]),
             Fun("pos", [Fun("green"),  16,  1]),
             Fun("pos", [Fun("yellow"), 16, 16])]
sequence  = [Fun("goal", [13]),
             Fun("goal",  [4]),
             Fun("goal",  [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
```

1. Initializing variables
2. Creating a player object (wrapping a *clingo* object)
3. Playing in rounds

Potassco

Setup and control loop

```
>> horizon   = 15
>> encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
>> positions = [Fun("pos", [Fun("red"),     1,  1]),
>>              Fun("pos", [Fun("blue"),    1, 16]),
>>              Fun("pos", [Fun("green"),  16,  1]),
>>              Fun("pos", [Fun("yellow"), 16, 16])]
>> sequence  = [Fun("goal", [13]),
>>              Fun("goal",  [4]),
>>              Fun("goal",  [7])]
```

```
player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
```

1. Initializing variables
2. Creating a player object (wrapping a *clingo* object)
3. Playing in rounds

Potassco

Setup and control loop

```
horizon   = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"),     1,  1]),
             Fun("pos", [Fun("blue"),    1, 16]),
             Fun("pos", [Fun("green"),  16,  1]),
             Fun("pos", [Fun("yellow"), 16, 16])]
sequence  = [Fun("goal", [13]),
             Fun("goal",  [4]),
             Fun("goal",  [7])]

>> player = Player(horizon, positions, encodings)
   for goal in sequence:
       print player.solve(goal)
```

1. Initializing variables
2. Creating a player object (wrapping a *clingo* object)
3. Playing in rounds

Potassco

Setup and control loop

```
   horizon   = 15
   encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
   positions = [Fun("pos", [Fun("red"),     1,  1]),
                Fun("pos", [Fun("blue"),    1, 16]),
                Fun("pos", [Fun("green"),  16,  1]),
                Fun("pos", [Fun("yellow"), 16, 16])]
   sequence  = [Fun("goal", [13]),
                Fun("goal",  [4]),
                Fun("goal",  [7])]

   player = Player(horizon, positions, encodings)
>> for goal in sequence:
>>     print player.solve(goal)
```

1. Initializing variables
2. Creating a player object (wrapping a *clingo* object)
3. Playing in rounds

Potassco

Setup and control loop

```
horizon   = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"),     1,  1]),
             Fun("pos", [Fun("blue"),    1, 16]),
             Fun("pos", [Fun("green"),  16,  1]),
             Fun("pos", [Fun("yellow"), 16, 16])]
sequence  = [Fun("goal", [13]),
             Fun("goal",  [4]),
             Fun("goal",  [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
```

1. Initializing variables
2. Creating a player object (wrapping a *clingo* object)
3. Playing in rounds

Potassco

Ricochet Robot Player

__init__

```
from gringo import Control, Model, Fun

class Player:
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
        self.ctl.ground([("base", [])])

    def solve(self, goal):
        for x in self.undo_external:
            self.ctl.assign_external(x, False)
        self.undo_external = []
        for x in self.last_positions + [goal]:
            self.ctl.assign_external(x, True)
            self.undo_external.append(x)
        self.last_solution = None
        self.ctl.solve(on_model=self.on_model)
        return self.last_solution

    def on_model(self, model):
        self.last_solution = model.atoms()
        self.last_positions = []
        for atom in model.atoms(Model.ATOMS):
            if (atom.name() == "pos" and len(atom.args()) == 4 and atom.args()[3] == self.horizon):
                self.last_positions.append(Fun("pos", atom.args()[:-1]))

horizon   = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"),     1,  1]), Fun("pos", [Fun("blue"),    1, 16]),
             Fun("pos", [Fun("green"),  16,  1]), Fun("pos", [Fun("yellow"), 16, 16])]
sequence  = [Fun("goal", [13]), Fun("goal", [4]), Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
```

Potassco

__init__

```
def __init__(self, horizon, positions, files):
    self.last_positions = positions
    self.last_solution = None
    self.undo_external = []
    self.horizon = horizon
    self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
    for x in files:
        self.ctl.load(x)
    self.ctl.ground([("base", [])])
```

1. Initializing variables
2. Creating *clingo* object
3. Loading encoding and instance
4. Grounding encoding and instance

Potassco

__init__

```
    def __init__(self, horizon, positions, files):
>>      self.last_positions = positions
>>      self.last_solution = None
>>      self.undo_external = []
>>      self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
        self.ctl.ground([("base", [])])
```

1. Initializing variables
2. Creating *clingo* object
3. Loading encoding and instance
4. Grounding encoding and instance

Potassco

__init__

```
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
>>      self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
        self.ctl.ground([("base", [])])
```

1. Initializing variables
2. Creating *clingo* object
3. Loading encoding and instance
4. Grounding encoding and instance

Potassco

__init__

```
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
>>      for x in files:
>>          self.ctl.load(x)
        self.ctl.ground([("base", [])])
```

1. Initializing variables
2. Creating *clingo* object
3. Loading encoding and instance
4. Grounding encoding and instance

Potassco

__init__

```
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
>>      self.ctl.ground([("base", [])])
```

1. Initializing variables
2. Creating *clingo* object
3. Loading encoding and instance
4. Grounding encoding and instance

Potassco

__init__

```
def __init__(self, horizon, positions, files):
    self.last_positions = positions
    self.last_solution = None
    self.undo_external = []
    self.horizon = horizon
    self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
    for x in files:
        self.ctl.load(x)
    self.ctl.ground([("base", [])])
```

1. Initializing variables
2. Creating *clingo* object
3. Loading encoding and instance
4. Grounding encoding and instance

Potassco

Ricochet Robot Player
solve

```
from gringo import Control, Model, Fun

class Player:
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
        self.ctl.ground([("base", [])])

    def solve(self, goal):
        for x in self.undo_external:
            self.ctl.assign_external(x, False)
        self.undo_external = []
        for x in self.last_positions + [goal]:
            self.ctl.assign_external(x, True)
            self.undo_external.append(x)
        self.last_solution = None
        self.ctl.solve(on_model=self.on_model)
        return self.last_solution

    def on_model(self, model):
        self.last_solution = model.atoms()
        self.last_positions = []
        for atom in model.atoms(Model.ATOMS):
            if (atom.name() == "pos" and len(atom.args()) == 4 and atom.args()[3] == self.horizon):
                self.last_positions.append(Fun("pos", atom.args()[:-1]))

horizon   = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"),     1,  1]), Fun("pos", [Fun("blue"),    1, 16]),
             Fun("pos", [Fun("green"),  16,  1]), Fun("pos", [Fun("yellow"), 16, 16])]
sequence  = [Fun("goal", [13]), Fun("goal", [4]), Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
```

Potassco

solve

```
def solve(self, goal):
      for x in self.undo_external:
          self.ctl.assign_external(x, False)
      self.undo_external = []
      for x in self.last_positions + [goal]:
          self.ctl.assign_external(x, True)
          self.undo_external.append(x)
      self.last_solution = None
      self.ctl.solve(on_model=self.on_model)
      return self.last_solution
```

1. Unsetting previous external atoms (viz. previous goal and positions)
2. Setting next external atoms (viz. next goal and positions)
3. Computing next stable model
   by passing user-defined `on_model` method

Potassco

solve

```
    def solve(self, goal):
>>          for x in self.undo_external:
>>              self.ctl.assign_external(x, False)
            self.undo_external = []
            for x in self.last_positions + [goal]:
                self.ctl.assign_external(x, True)
                self.undo_external.append(x)
            self.last_solution = None
            self.ctl.solve(on_model=self.on_model)
            return self.last_solution
```

1. Unsetting previous external atoms (viz. previous goal and positions)
2. Setting next external atoms (viz. next goal and positions)
3. Computing next stable model
   by passing user-defined `on_model` method

Potassco

solve

```
    def solve(self, goal):
            for x in self.undo_external:
                self.ctl.assign_external(x, False)
>>          self.undo_external = []
>>          for x in self.last_positions + [goal]:
>>              self.ctl.assign_external(x, True)
>>              self.undo_external.append(x)
            self.last_solution = None
            self.ctl.solve(on_model=self.on_model)
            return self.last_solution
```

1. Unsetting previous external atoms (viz. previous goal and positions)
2. Setting next external atoms (viz. next goal and positions)
3. Computing next stable model
   by passing user-defined `on_model` method

Potassco

solve

```
    def solve(self, goal):
            for x in self.undo_external:
                self.ctl.assign_external(x, False)
            self.undo_external = []
            for x in self.last_positions + [goal]:
                self.ctl.assign_external(x, True)
                self.undo_external.append(x)
>>          self.last_solution = None
>>          self.ctl.solve(on_model=self.on_model)
>>          return self.last_solution
```

1. Unsetting previous external atoms (viz. previous goal and positions)
2. Setting next external atoms (viz. next goal and positions)
3. Computing next stable model
   by passing user-defined `on_model` method

Potassco

solve

```
def solve(self, goal):
    for x in self.undo_external:
        self.ctl.assign_external(x, False)
    self.undo_external = []
    for x in self.last_positions + [goal]:
        self.ctl.assign_external(x, True)
        self.undo_external.append(x)
    self.last_solution = None
    self.ctl.solve(on_model=self.on_model)
    return self.last_solution
```

1. Unsetting previous external atoms (viz. previous goal and positions)
2. Setting next external atoms (viz. next goal and positions)
3. Computing next stable model
   by passing user-defined `on_model` method

Potassco

solve

```
def solve(self, goal):
    for x in self.undo_external:
        self.ctl.assign_external(x, False)
    self.undo_external = []
    for x in self.last_positions + [goal]:
        self.ctl.assign_external(x, True)
        self.undo_external.append(x)
    self.last_solution = None
    self.ctl.solve(on_model=self.on_model)
    return self.last_solution
```

1. Unsetting previous external atoms (viz. previous goal and positions)
2. Setting next external atoms (viz. next goal and positions)
3. Computing next stable model
   by passing user-defined `on_model` method

Potassco

Ricochet Robot Player
on_model

```
from gringo import Control, Model, Fun

class Player:
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
        self.ctl.ground([("base", [])])

    def solve(self, goal):
        for x in self.undo_external:
            self.ctl.assign_external(x, False)
        self.undo_external = []
        for x in self.last_positions + [goal]:
            self.ctl.assign_external(x, True)
            self.undo_external.append(x)
        self.last_solution = None
        self.ctl.solve(on_model=self.on_model)
        return self.last_solution

    def on_model(self, model):
        self.last_solution = model.atoms()
        self.last_positions = []
        for atom in model.atoms(Model.ATOMS):
            if (atom.name() == "pos" and len(atom.args()) == 4 and atom.args()[3] == self.horizon):
                self.last_positions.append(Fun("pos", atom.args()[:-1]))

horizon   = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"),     1,  1]), Fun("pos", [Fun("blue"),    1, 16]),
             Fun("pos", [Fun("green"),  16,  1]), Fun("pos", [Fun("yellow"), 16, 16])]
sequence  = [Fun("goal", [13]), Fun("goal", [4]), Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
```

Potassco

on_model

```
def on_model(self, model):
      self.last_solution = model.atoms()
      self.last_positions = []
      for atom in model.atoms(Model.ATOMS):
          if (atom.name() == "pos" and
                  len(atom.args()) == 4 and
                  atom.args()[3] == self.horizon):
              self.last_positions.append(Fun("pos", atom.args()[:-1]))
```

1. Storing stable model
2. Extracting atoms (viz. last robot positions)
   by adding `pos(R,X,Y)` for each `pos(R,X,Y,horizon)`

Potassco

on_model

```
    def on_model(self, model):
>>          self.last_solution = model.atoms()
            self.last_positions = []
            for atom in model.atoms(Model.ATOMS):
                if (atom.name() == "pos" and
                        len(atom.args()) == 4 and
                        atom.args()[3] == self.horizon):
                    self.last_positions.append(Fun("pos", atom.args()[:-1]))
```

1. Storing stable model
2. Extracting atoms (viz. last robot positions)
   by adding `pos(R,X,Y)` for each `pos(R,X,Y,horizon)`

Potassco

on_model

```
    def on_model(self, model):
          self.last_solution = model.atoms()
>>        self.last_positions = []
>>        for atom in model.atoms(Model.ATOMS):
>>            if (atom.name() == "pos" and
>>                    len(atom.args()) == 4 and
>>                    atom.args()[3] == self.horizon):
>>                self.last_positions.append(Fun("pos", atom.args()[:-1]))
```

1. Storing stable model
2. Extracting atoms (viz. last robot positions)
   by adding `pos(R,X,Y)` for each `pos(R,X,Y,horizon)`

Potassco

on_model

```
    def on_model(self, model):
          self.last_solution = model.atoms()
>>        self.last_positions = []
>>        for atom in model.atoms(Model.ATOMS):
>>            if (atom.name() == "pos" and
>>                    len(atom.args()) == 4 and
>>                    atom.args()[3] == self.horizon):
>>                self.last_positions.append(Fun("pos", atom.args()[:-1]))
```

1. Storing stable model
2. Extracting atoms (viz. last robot positions)
   by adding `pos(R,X,Y)` for each `pos(R,X,Y,horizon)`

Potassco

on_model

```
def on_model(self, model):
      self.last_solution = model.atoms()
      self.last_positions = []
      for atom in model.atoms(Model.ATOMS):
          if (atom.name() == "pos" and
                  len(atom.args()) == 4 and
                  atom.args()[3] == self.horizon):
              self.last_positions.append(Fun("pos", atom.args()[:-1]))
```

1. Storing stable model
2. Extracting atoms (viz. last robot positions)
   by adding `pos(R,X,Y)` for each `pos(R,X,Y,horizon)`

Potassco

on_model

```
def on_model(self, model):
      self.last_solution = model.atoms()
      self.last_positions = []
      for atom in model.atoms(Model.ATOMS):
          if (atom.name() == "pos" and
                  len(atom.args()) == 4 and
                  atom.args()[3] == self.horizon):
              self.last_positions.append(Fun("pos", atom.args()[:-1]))
```

1. Storing stable model
2. Extracting atoms (viz. last robot positions)
   by adding `pos(R,X,Y)` for each `pos(R,X,Y,horizon)`

Potassco

ricochet.py

```
from gringo import Control, Model, Fun

class Player:
    def __init__(self, horizon, positions, files):
        self.last_positions = positions
        self.last_solution = None
        self.undo_external = []
        self.horizon = horizon
        self.ctl = Control(['-c', 'horizon={0}'.format(self.horizon)])
        for x in files:
            self.ctl.load(x)
        self.ctl.ground([("base", [])])

    def solve(self, goal):
        for x in self.undo_external:
            self.ctl.assign_external(x, False)
        self.undo_external = []
        for x in self.last_positions + [goal]:
            self.ctl.assign_external(x, True)
            self.undo_external.append(x)
        self.last_solution = None
        self.ctl.solve(on_model=self.on_model)
        return self.last_solution

    def on_model(self, model):
        self.last_solution = model.atoms()
        self.last_positions = []
        for atom in model.atoms(Model.ATOMS):
            if (atom.name() == "pos" and len(atom.args()) == 4 and atom.args()[3] == self.horizon):
                self.last_positions.append(Fun("pos", atom.args()[:-1]))

horizon   = 15
encodings = ["board.lp", "targets.lp", "ricochet.lp", "optimization.lp"]
positions = [Fun("pos", [Fun("red"),     1,  1]), Fun("pos", [Fun("blue"),    1, 16]),
             Fun("pos", [Fun("green"),  16,  1]), Fun("pos", [Fun("yellow"), 16, 16])]
sequence  = [Fun("goal", [13]), Fun("goal", [4]), Fun("goal", [7])]

player = Player(horizon, positions, encodings)
for goal in sequence:
    print player.solve(goal)
```

Potassco

Let's play!

```
$ python ricochet.py
[move(red,0,1,1), move(yellow,-1,0,14), move(yellow,-1,0,12), move(yellow,-1,0,11),
 move(yellow,-1,0,9), move(red,1,0,7), move(red,1,0,2), move(yellow,-1,0,10),
 move(yellow,-1,0,13), move(yellow,-1,0,15), move(red,-1,0,4), move(yellow,0,-1,6),
 move(red,0,1,3), move(red,0,1,5), move(yellow,0,1,8)]
[move(blue,0,1,15), move(blue,0,1,11), move(blue,0,1,8), move(blue,0,1,3),
 move(blue,1,0,2), move(blue,0,1,9), move(blue,-1,0,7), move(blue,0,1,10),
 move(blue,0,1,13), move(blue,-1,0,4), move(blue,0,-1,1), move(blue,0,-1,6),
 move(green,-1,0,5), move(blue,0,1,12), move(blue,0,1,14)]
[move(green,1,0,15), move(green,1,0,8), move(green,1,0,5), move(green,1,0,4),
 move(green,1,0,3), move(green,1,0,10), move(green,1,0,7), move(green,1,0,12),
 move(green,1,0,9), move(green,1,0,2), move(green,1,0,11), move(green,1,0,13),
 move(green,1,0,6), move(green,1,0,14), move(green,0,1,1)]

$ ./visualize.py
```

http://potassco.sourceforge.net

☞ gringo/clingo distribution ./examples/clingo/robots/

Potassco

Let's play!

```
$ python ricochet.py
[move(red,0,1,1), move(yellow,-1,0,14), move(yellow,-1,0,12), move(yellow,-1,0,11),
 move(yellow,-1,0,9), move(red,1,0,7), move(red,1,0,2), move(yellow,-1,0,10),
 move(yellow,-1,0,13), move(yellow,-1,0,15), move(red,-1,0,4), move(yellow,0,-1,6),
 move(red,0,1,3), move(red,0,1,5), move(yellow,0,1,8)]
[move(blue,0,1,15), move(blue,0,1,11), move(blue,0,1,8), move(blue,0,1,3),
 move(blue,1,0,2), move(blue,0,1,9), move(blue,-1,0,7), move(blue,0,1,10),
 move(blue,0,1,13), move(blue,-1,0,4), move(blue,0,-1,1), move(blue,0,-1,6),
 move(green,-1,0,5), move(blue,0,1,12), move(blue,0,1,14)]
[move(green,1,0,15), move(green,1,0,8), move(green,1,0,5), move(green,1,0,4),
 move(green,1,0,3), move(green,1,0,10), move(green,1,0,7), move(green,1,0,12),
 move(green,1,0,9), move(green,1,0,2), move(green,1,0,11), move(green,1,0,13),
 move(green,1,0,6), move(green,1,0,14), move(green,0,1,1)]

$ ./visualize.py
```

http://potassco.sourceforge.net

☞ gringo/clingo distribution ./examples/clingo/robots/

Potassco

Let's play!

```
$ python ricochet.py
[move(red,0,1,1), move(yellow,-1,0,14), move(yellow,-1,0,12), move(yellow,-1,0,11),
 move(yellow,-1,0,9), move(red,1,0,7), move(red,1,0,2), move(yellow,-1,0,10),
 move(yellow,-1,0,13), move(yellow,-1,0,15), move(red,-1,0,4), move(yellow,0,-1,6),
 move(red,0,1,3), move(red,0,1,5), move(yellow,0,1,8)]
[move(blue,0,1,15), move(blue,0,1,11), move(blue,0,1,8), move(blue,0,1,3),
 move(blue,1,0,2), move(blue,0,1,9), move(blue,-1,0,7), move(blue,0,1,10),
 move(blue,0,1,13), move(blue,-1,0,4), move(blue,0,-1,1), move(blue,0,-1,6),
 move(green,-1,0,5), move(blue,0,1,12), move(blue,0,1,14)]
[move(green,1,0,15), move(green,1,0,8), move(green,1,0,5), move(green,1,0,4),
 move(green,1,0,3), move(green,1,0,10), move(green,1,0,7), move(green,1,0,12),
 move(green,1,0,9), move(green,1,0,2), move(green,1,0,11), move(green,1,0,13),
 move(green,1,0,6), move(green,1,0,14), move(green,0,1,1)]

$ ./visualize.py
```

http://potassco.sourceforge.net

☞ gringo/clingo distribution ./examples/clingo/robots/

Potassco

Let's play!

```
$ python ricochet.py
[move(red,0,1,1), move(yellow,-1,0,14), move(yellow,-1,0,12), move(yellow,-1,0,11),
 move(yellow,-1,0,9), move(red,1,0,7), move(red,1,0,2), move(yellow,-1,0,10),
 move(yellow,-1,0,13), move(yellow,-1,0,15), move(red,-1,0,4), move(yellow,0,-1,6),
 move(red,0,1,3), move(red,0,1,5), move(yellow,0,1,8)]
[move(blue,0,1,15), move(blue,0,1,11), move(blue,0,1,8), move(blue,0,1,3),
 move(blue,1,0,2), move(blue,0,1,9), move(blue,-1,0,7), move(blue,0,1,10),
 move(blue,0,1,13), move(blue,-1,0,4), move(blue,0,-1,1), move(blue,0,-1,6),
 move(green,-1,0,5), move(blue,0,1,12), move(blue,0,1,14)]
[move(green,1,0,15), move(green,1,0,8), move(green,1,0,5), move(green,1,0,4),
 move(green,1,0,3), move(green,1,0,10), move(green,1,0,7), move(green,1,0,12),
 move(green,1,0,9), move(green,1,0,2), move(green,1,0,11), move(green,1,0,13),
 move(green,1,0,6), move(green,1,0,14), move(green,0,1,1)]

$ ./visualize.py
```

http://potassco.sourceforge.net

☞ gringo/clingo distribution ./examples/clingo/robots/

Potassco

Outline

1 Introduction
2 Foundations
3 Modeling
4 Modeling and Controlling
5 Case-study: Ricochet Robots
6 Case-sketch: Preferences and optimization
7 Potassco
8 Summary

Potassco

Motivation

- Preferences are pervasive
- The identification of preferred, or optimal, solutions is often indispensable in real-world applications

  In many cases, this also involves the combination of various qualitative and quantitative preferences

- Only optimization statements representing objective functions using sum or count aggregates are established components of ASP systems
- Example $\#minimize\{40 : sauna, 70 : dive\}$

Potassco

Motivation

- Preferences are pervasive
- The identification of preferred, or optimal, solutions is often indispensable in real-world applications

  In many cases, this also involves the combination of various qualitative and quantitative preferences

- Only optimization statements representing objective functions using sum or count aggregates are established components of ASP systems
- Example $\#minimize\{40 : sauna, 70 : dive\}$

Potassco

Motivation

- Preferences are pervasive
- The identification of preferred, or optimal, solutions is often indispensable in real-world applications

  In many cases, this also involves the combination of various qualitative and quantitative preferences

- Only optimization statements representing objective functions using sum or count aggregates are established components of ASP systems
- Example $\#minimize\{40 : sauna, 70 : dive\}$

Potassco

Motivation

- Preferences are pervasive
- The identification of preferred, or optimal, solutions is often indispensable in real-world applications

  In many cases, this also involves the combination of various qualitative and quantitative preferences

- Only optimization statements representing objective functions using sum or count aggregates are established components of ASP systems
- Example $\#minimize\{40 : sauna, 70 : dive\}$

The asprin framework

- asprin is a framework for handling preferences among the stable models of logic programs
  - general because it captures numerous existing approaches to preference from the literature
  - flexible because it allows for an easy implementation of new or extended existing approaches
- asprin builds upon advanced control capacities for incremental and meta solving, allowing for

  without any modifications to the ASP solver

  significantly reducing redundancies

  via an implementation through ordinary ASP encodings

Potassco

The asprin framework

- asprin is a framework for handling preferences among the stable models of logic programs
  - general because it captures numerous existing approaches to preference from the literature
  - flexible because it allows for an easy implementation of new or extended existing approaches
- asprin builds upon advanced control capacities for incremental and meta solving, allowing for
  - search for specific preferred solutions without any modifications to the ASP solver
  - continuous integrated solving process significantly reducing redundancies
  - user-configurability via an implementation through ordinary ASP encodings

Potassco

The asprin framework

- asprin is a framework for handling preferences among the stable models of logic programs
  - general because it captures numerous existing approaches to preference from the literature
  - flexible because it allows for an easy implementation of new or extended existing approaches
- asprin builds upon advanced control capacities for incremental and meta solving, allowing for
  - search for specific preferred solutions without any modifications to the ASP solver
  - continuous integrated solving process significantly reducing redundancies
  - high customizability via an implementation through ordinary ASP encodings

Potassco

The asprin framework

- asprin is a framework for handling preferences among the stable models of logic programs
  - general because it captures numerous existing approaches to preference from the literature
  - flexible because it allows for an easy implementation of new or extended existing approaches
- asprin builds upon advanced control capacities for incremental and meta solving, allowing for
  - search for specific preferred solutions without any modifications to the ASP solver
  - continuous integrated solving process significantly reducing redundancies
  - high customizability via an implementation through ordinary ASP encodings

Potassco

Example

$$\#preference(costs, less(weight))\{40 : sauna, 70 : dive\}$$
$$\#preference(fun, superset)\{sauna, dive, hike, \sim bunji\}$$
$$\#preference(temps, aso)\{dive > sauna \,\|\, hot, sauna > dive \,\|\, \neg hot\}$$
$$\#preference(all, pareto)\{name(costs), name(fun), name(temps)\}$$

$$\#optimize(all)$$

Potassco

asprin's library

- Basic preference types
  - `subset` and `superset`
  - `less(cardinality)` and `more(cardinality)`
  - `less(weight)` and `more(weight)`
  - `aso` (Answer Set Optimization)
  - `poset` (Qualitative Preferences)
- Composite preference types
  - `neg`
  - `and`
  - `pareto`
  - `lexico`
- See *Potassco Guide* on how to define further types

  and `http://potassco.sourceforge.net/labs.html#asprin`

Potassco

asprin's library

- Basic preference types
  - `subset` and `superset`
  - `less(cardinality)` and `more(cardinality)`
  - `less(weight)` and `more(weight)`
  - `aso` (Answer Set Optimization)
  - `poset` (Qualitative Preferences)
- Composite preference types
  - `neg`
  - `and`
  - `pareto`
  - `lexico`
- See *Potassco Guide* on how to define further types and `http://potassco.sourceforge.net/labs.html#asprin`

Potassco

asprin's library

- Basic preference types
  - `subset` and `superset`
  - `less(cardinality)` and `more(cardinality)`
  - `less(weight)` and `more(weight)`
  - `aso` (Answer Set Optimization)
  - `poset` (Qualitative Preferences)
- Composite preference types
  - `neg`
  - `and`
  - `pareto`
  - `lexico`
- See *Potassco Guide* on how to define further types

  and `http://potassco.sourceforge.net/labs.html#asprin`

Potassco

asprin's library

- Basic preference types
  - `subset` and `superset`
  - `less(cardinality)` and `more(cardinality)`
  - `less(weight)` and `more(weight)`
  - `aso` (Answer Set Optimization)
  - `poset` (Qualitative Preferences)
- Composite preference types
  - `neg`
  - `and`
  - `pareto`
  - `lexico`
- See *Potassco Guide* on how to define further types

  and `http://potassco.sourceforge.net/labs.html#asprin`

Potassco

Outline

1 Introduction
2 Foundations
3 Modeling
4 Modeling and Controlling
5 Case-study: Ricochet Robots
6 Case-sketch: Preferences and optimization
7 Potassco
8 Summary

Potassco

potassco.sourceforge.net

Potassco, the Potsdam Answer Set Solving Collection,
bundles tools for ASP developed at the University of Potsdam:

- Grounder gringo, lingo
- Solver clasp, claspfolio, claspar, aspeed
- Grounder+Solver Clingo, Clingcon, ROSoClingo
- Further Tools aspartame, aspcud, asprin, chasp, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, etc

- Benchmarks, Repository asparagus.cs.uni-potsdam.de

Potassco

potassco.sourceforge.net

Potassco, the Potsdam Answer Set Solving Collection,
bundles tools for ASP developed at the University of Potsdam:

- Grounder gringo, lingo
- Solver clasp, claspfolio, claspar, aspeed
- Grounder+Solver Clingo, Clingcon, ROSoClingo
- Further Tools aspartame, aspcud, asprin, chasp, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, etc

- Benchmark repository asparagus.cs.uni-potsdam.de

Potassco

potassco.sourceforge.net

Potassco, the Potsdam Answer Set Solving Collection,
bundles tools for ASP developed at the University of Potsdam:

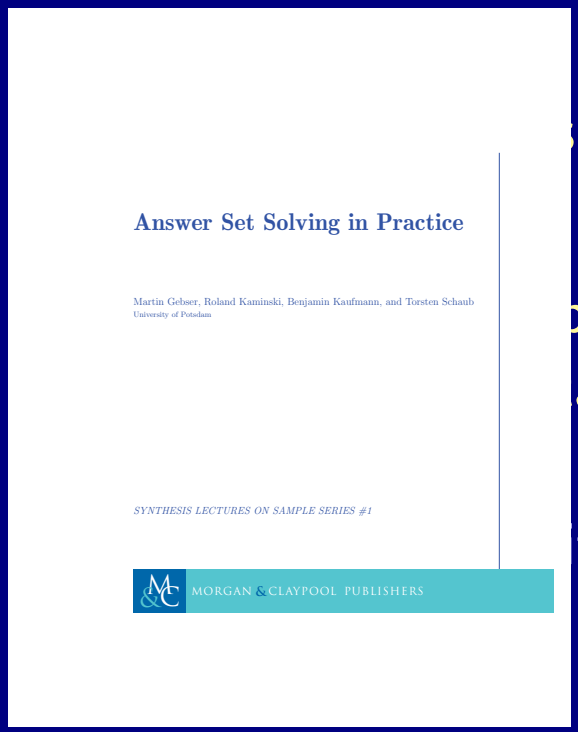
- Grounder gringo, lingo
- Solver clasp, claspfolio, claspar, aspeed
- Grounder+Solver Clingo, Clingcon, ROSoClingo
- Further Tools aspartame, aspcud, asprin, chasp, claspre, clavis, coala, fimo, insight, metasp, plasp, piclasp, etc

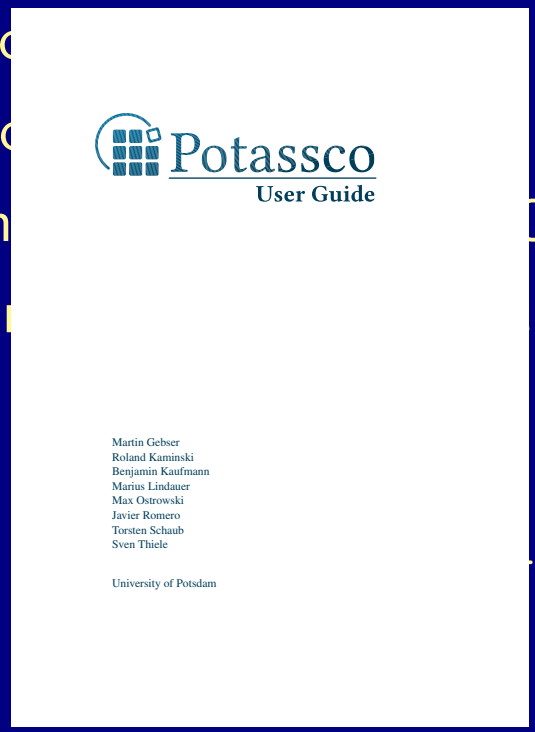
- Benchmark repository asparagus.cs.uni-potsdam.de

Potassco

potassco.sourceforge.net

Potassco, the Potsdam Answer Set Solving Collection,
bundles tools for ASP developed at the University of Potsdam:





Potassco

Outline

1 Introduction

2 Foundations

3 Modeling

4 Modeling and Controlling

5 Case-study: Ricochet Robots

6 Case-sketch: Preferences and optimization

7 Potassco

8 Summary

Potassco

Summary

- ASP is a viable tool for Knowledge Representation and Reasoning
- ASP offers efficient and versatile off-the-shelf solving technology
- ASP offers an expanding functionality and ease of use
  - rapid application development tool
- ASP has a growing range of applications

Potassco

Summary

- ASP is a viable tool for Knowledge Representation and Reasoning
- ASP offers efficient and versatile off-the-shelf solving technology
- ASP offers an expanding functionality and ease of use
  - rapid application development tool
- ASP has a growing range of applications

ASP = DB+LP+KR+SAT

Potassco

Summary

- ASP is a viable tool for Knowledge Representation and Reasoning
- ASP offers efficient and versatile off-the-shelf solving technology
- ASP offers an expanding functionality and ease of use
  - rapid application development tool
- ASP has a growing range of applications

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

Potassco

Summary

- ASP is a viable tool for Knowledge Representation and Reasoning
- ASP offers efficient and versatile off-the-shelf solving technology
- ASP offers an expanding functionality and ease of use
  - rapid application development tool
- ASP has a growing range of applications

`http://potassco.sourceforge.net`

Potassco