Experiencing Answer Set Programming at Work Today and Tomorrow

Torsten Schaub

University of Potsdam

Potassco

Outline

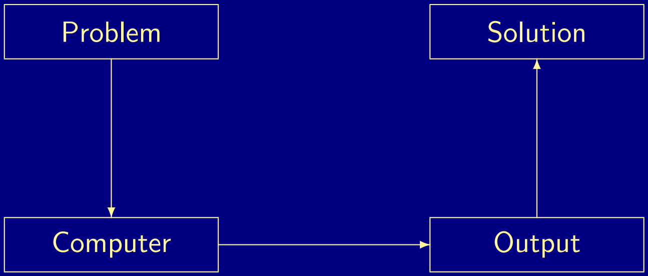
Potassco

Outline

Outline

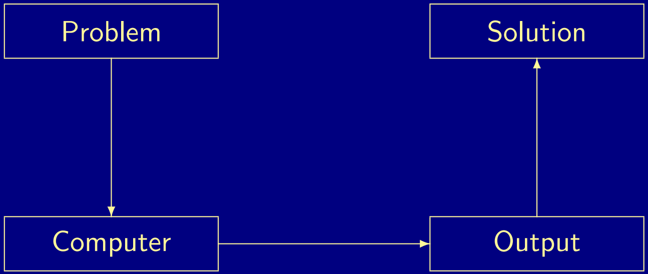
Informatics

"What is the problem?" versus *"How to solve the problem?"*



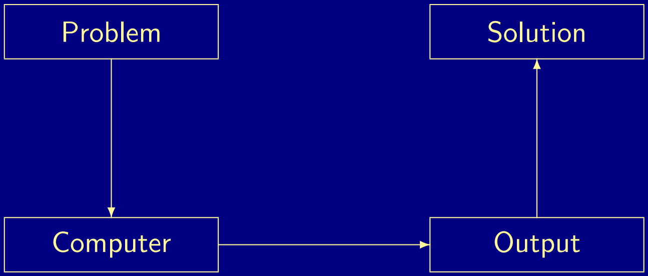
Informatics

"What is the problem?" versus *"How to solve the problem?"*



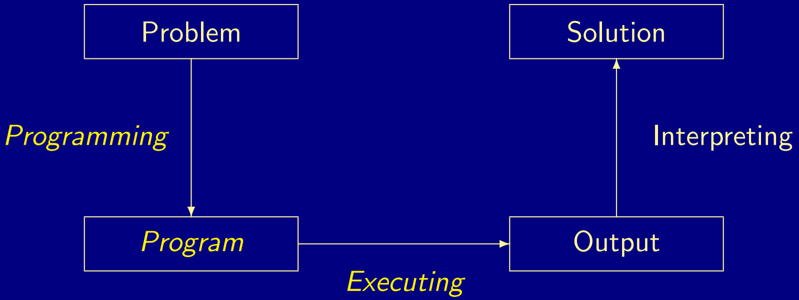
Traditional programming

"What is the problem?" versus *"How to solve the problem?"*



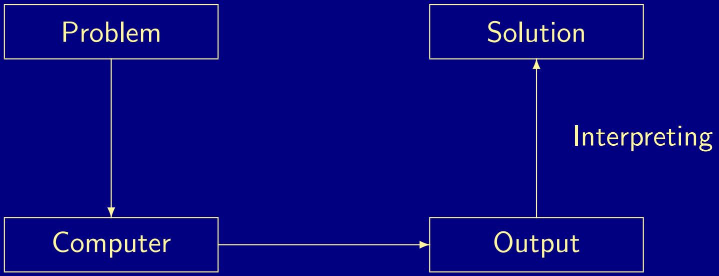
Traditional programming

"What is the problem?" versus *"How to solve the problem?"*



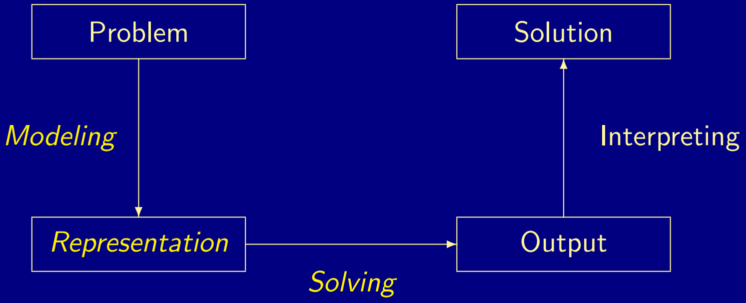
Declarative problem solving

"What is the problem?" versus *"How to solve the problem?"*



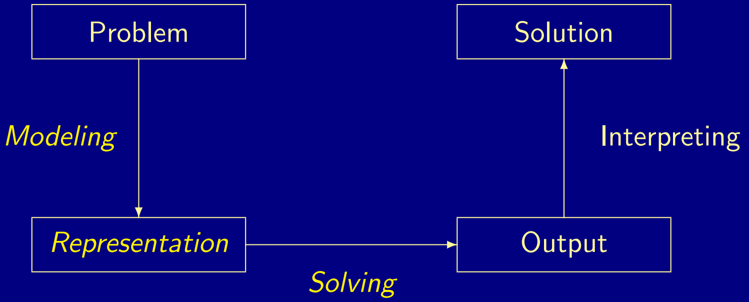
Declarative problem solving

"What is the problem?" versus *"How to solve the problem?"*



Declarative problem solving

"What is the problem?" versus *"How to solve the problem?"*



Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is versatile as reflected by the ASP solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT competitions
- ASP embraces many emerging application areas, and users

Potassco

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is versatile as reflected by the ASP solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT competitions
- ASP embraces many emerging application areas, and users

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is versatile as reflected by the ASP solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT competitions
- ASP embraces many emerging application areas, and users

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is versatile as reflected by the ASP solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT competitions
- ASP embraces many emerging application areas, and users

Potassco

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is versatile as reflected by the ASP solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT competitions
- ASP embraces many emerging application areas, and users

Potassco

Answer Set Programming

in a Nutshell

- ASP is an approach to declarative problem solving, combining
  - a rich yet simple modeling language
  - with high-performance solving capacities
- ASP has its roots in
  - (deductive) databases
  - logic programming (with negation)
  - (logic-based) knowledge representation and (nonmonotonic) reasoning
  - constraint solving (in particular, SATisfiability testing)
- ASP allows for solving all search problems in NP (and NP^{NP}) in a uniform way
- ASP is versatile as reflected by the ASP solver *clasp*, winning first places at ASP, CASC, MISC, PB, and SAT competitions
- ASP embraces many emerging application areas, and users

Potassco

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

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

Model Generation based approach (eg. SATisfiability testing)

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

Automated planning, Kautz and Selman (ECAI'92)

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

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

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

Model Generation based approach (eg. SATisfiability testing)

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

Automated planning, Kautz and Selman (ECAI'92)

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

Potassco

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

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

Model Generation based approach (eg. SATisfiability testing)

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

Automated planning, Kautz and Selman (ECAI'92)

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

KR's shift of paradigm

Theorem Proving based approach (eg. Prolog)

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

Model Generation based approach (eg. SATisfiability testing)

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

Automated planning, Kautz and Selman (ECAI'92)

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

Potassco

Model Generation based Problem Solving

Representation	Solution
constraint satisfaction problem	assignment
propositional horn theories	smallest model
propositional theories	models
propositional theories	minimal models
propositional theories	stable models
propositional programs	minimal models
propositional programs	supported models
propositional programs	stable models
first-order theories	models
first-order theories	minimal models
first-order theories	stable models
first-order theories	Herbrand models
auto-epistemic theories	expansions
default theories	extensions

Model Generation based Problem Solving

Representation	Solution
constraint satisfaction problem	assignment
propositional horn theories	smallest model
propositional theories	models
propositional theories	minimal models
propositional theories	stable models
propositional programs	minimal models
propositional programs	supported models
propositional programs	stable models
first-order theories	models
first-order theories	minimal models
first-order theories	stable models
first-order theories	Herbrand models
auto-epistemic theories	expansions
default theories	extensions
⋮	⋮

Model Generation based Problem Solving

Representation	Solution	
constraint satisfaction problem	assignment	
propositional horn theories	smallest model	
propositional theories	models	**SAT**
propositional theories	minimal models	
propositional theories	stable models	
propositional programs	minimal models	
propositional programs	supported models	
propositional programs	stable models	
first-order theories	models	
first-order theories	minimal models	
first-order theories	stable models	
first-order theories	Herbrand models	
auto-epistemic theories	expansions	
default theories	extensions	
⋮	⋮	

Model Generation based Problem Solving

Representation	Solution	
constraint satisfaction problem	assignment	
propositional horn theories	smallest model	
propositional theories	models	**SAT**
propositional theories	minimal models	
propositional theories	stable models	
propositional programs	minimal models	
propositional programs	supported models	
propositional programs	stable models	
first-order theories	models	
first-order theories	minimal models	
first-order theories	stable models	
first-order theories	Herbrand models	
auto-epistemic theories	expansions	**NMR**
default theories	extensions	**NMR**
⋮	⋮	

Potassco

Model Generation based Problem Solving

Representation	Solution	
constraint satisfaction problem	assignment	
propositional horn theories	smallest model	
propositional theories	models	**SAT**
propositional theories	minimal models	
propositional theories	stable models	
propositional programs	minimal models	
propositional programs	supported models	
propositional programs	stable models	**ASP**
first-order theories	models	
first-order theories	minimal models	
first-order theories	stable models	
first-order theories	Herbrand models	
auto-epistemic theories	expansions	**NMR**
default theories	extensions	**NMR**
⋮	⋮	

Potassco

Answer Set Programming *in general*

Representation	Solution	
constraint satisfaction problem	assignment	
propositional horn theories	smallest model	
propositional theories	models	
propositional theories	minimal models	
propositional theories	stable models	**ASP**
propositional programs	minimal models	
propositional programs	supported models	
propositional programs	stable models	**ASP**
first-order theories	models	
first-order theories	minimal models	
first-order theories	stable models	**ASP**
first-order theories	Herbrand models	
auto-epistemic theories	expansions	
default theories	extensions	
⋮	⋮	

Answer Set Programming *in general*

Representation	Solution	
constraint satisfaction problem	assignment	
propositional horn theories	smallest model	
propositional theories	models	
propositional theories	minimal models	
propositional theories	stable models	**ASP**
propositional programs	minimal models	
propositional programs	supported models	
propositional programs	stable models	**ASP**
first-order theories	models	
first-order theories	minimal models	
first-order theories	stable models	**ASP**
first-order theories	Herbrand models	
auto-epistemic theories	expansions	
default theories	extensions	
⋮	⋮	

Potassco

Answer Set Programming *in practice*

Representation	Solution
constraint satisfaction problem	assignment
propositional horn theories	smallest model
propositional theories	models
propositional theories	minimal models
propositional theories	stable models
propositional programs	minimal models
propositional programs	supported models
propositional programs	stable models
first-order theories	models
first-order theories	minimal models
first-order theories	stable models
first-order theories	Herbrand models
auto-epistemic theories	expansions
default theories	extensions
first-order programs	stable Herbrand models

ASP versus LP

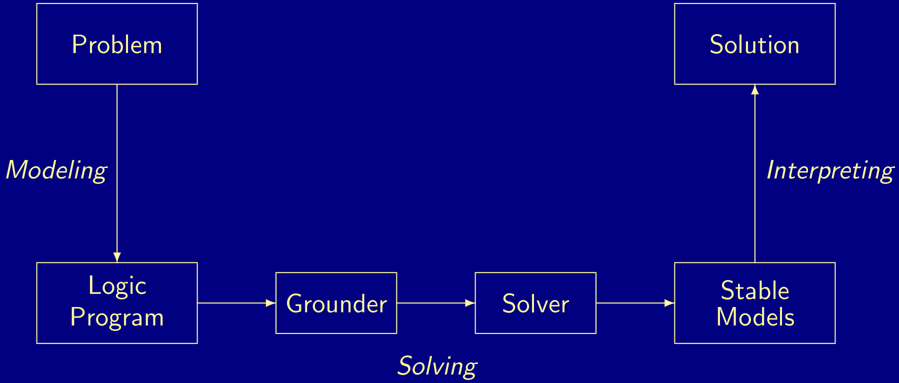
ASP	Prolog
Model generation	Query orientation
Bottom-up	Top-down
Modeling language	Programming language
Rule-based format	
Instantiation Flat terms	Unification Nested terms
(Turing +) $NP(^{NP})$	Turing

ASP versus SAT

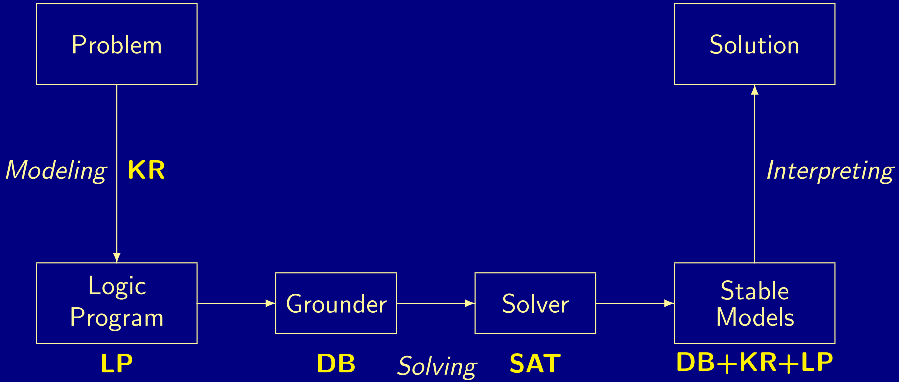
ASP	SAT
Model generation	
Bottom-up	
Constructive Logic	Classical Logic
Closed (and open) world reasoning	Open world reasoning
Modeling language	—
Complex reasoning modes	Satisfiability testing
Satisfiability	Satisfiability
Enumeration/Projection	—
Intersection/Union	—
Optimization	—
(Turing +) $NP(^{NP})$	NP

Potassco

ASP solving


Problem
Solution
Modeling
Interpreting
Logic Program
Grounder
Solver
Stable Models
Solving

Rooting ASP solving



Answer Set Programming

in a Hazelnutshell

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

  tailored to Knowledge Representation and Reasoning

Answer Set Programming

in a Hazelnutshell

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

  tailored to Knowledge Representation and Reasoning

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

Answer Set Programming

in a Hazelnutshell

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

  tailored to Knowledge Representation and Reasoning

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

Potassco

Declarativity versus Scalability

Declarativity

ASP does separate a problem's representation from the algorithms used for solving it

Scalability

1. ASP does not separate a problem's representation from its induced combinatorics
2. Boolean constraint technology is rather sensitive to search parameters

Followup to: M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Challenges in Answer Set Solving. In *Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, pages 74–90. Springer, 2011

Declarativity versus Scalability

Declarativity

ASP does separate a problem's representation from the algorithms used for solving it

Scalability

1. ASP does not separate a problem's representation from its induced combinatorics
2. Boolean constraint technology is rather sensitive to search parameters

Followup to: M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Challenges in Answer Set Solving. In *Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, pages 74–90. Springer, 2011

Declarativity versus Scalability

Declarativity

ASP does separate a problem's representation from the algorithms used for solving it

Scalability

1. ASP does not separate a problem's representation from its induced combinatorics
2. Boolean constraint technology is rather sensitive to search parameters

Followup to: M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Challenges in Answer Set Solving. In *Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, pages 74–90. Springer, 2011

Declarativity versus Scalability

Declarativity

ASP does separate a problem's representation from the algorithms used for solving it

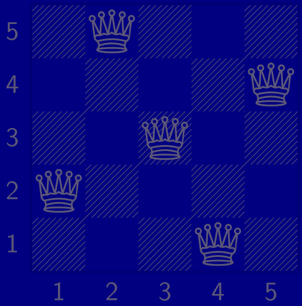
Scalability

1. ASP does not separate a problem's representation from its induced combinatorics
2. Boolean constraint technology is rather sensitive to search parameters

Followup to: M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Challenges in Answer Set Solving. In *Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, pages 74–90. Springer, 2011

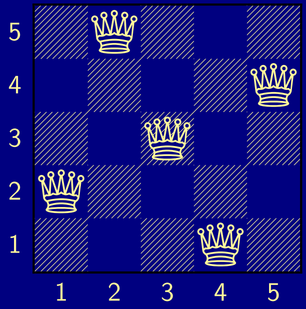
Outline

The n-queens problem



- Place n queens on an $n \times n$ chess board
- Queens must not attack one another

The n-queens problem



- Place n queens on an $n \times n$ chess board
- Queens must not attack one another

Basic encoding

queensB.lp

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

 :- not { 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

Advanced encoding

queensA.lp

```
{ 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.
```

Corrupted encoding

queensC.lp

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

 :- not { queen(I,J,K) } == n.
 :- queen(I,J,K), queen(I,JJ,K), J != JJ.
 :- queen(I,J,K), queen(II,J,K), I != II.
 :- queen(I,J,K), queen(II,JJ,K), (I,J)!=(II,JJ), I-J==II-JJ.
 :- queen(I,J,K), queen(II,JJ,K), (I,J)!=(II,JJ), I+J==II+JJ.

queen(I,J) :- queen(I,J,K).
```

Potassco

Grounding size

via `wc --lines`

n	queensB.lp	queensA.lp	queensC.lp
10	3053	310	30413
20	25493	830	509613
30	87333	1550	2619613
40	208573	2470	8342413
50	409213	3590	20460013
60	709253	4910	42554413
70	1128693	6430	79007613
80	1687533	8150	135001613
90	2405773	10070	217255513
100	3303413	12190	331350013

Challenge one

Fact

ASP Modeling (still) requires Craft, Experience, and Knowledge

Challenge

Theory and Tools for Non-Ground Pre-processing

Challenge one

Fact

ASP Modeling (still) requires Craft, Experience, and Knowledge

Challenge

Theory and Tools for Non-Ground Pre-processing

Challenge one

Fact

ASP Modeling (still) requires Craft, Experience, and Knowledge

Challenge

Theory and Tools for Non-Ground Pre-processing — *Just like SQL !*

Outline

Outline

Towards conflict-driven search

Boolean constraint solving algorithms pioneered for SAT led to:

- Traditional DPLL-style approach
  (DPLL stands for 'Davis-Putnam-Logemann-Loveland')
  - (Unit) propagation
  - (Chronological) backtracking
  - in ASP, eg *smodels*
- Modern CDCL-style approach
  (CDCL stands for 'Conflict-Driven Constraint Learning')
  - (Unit) propagation
  - Conflict analysis (via resolution)
  - Learning + Backjumping + Assertion
  - in ASP, eg *clasp*

DPLL-style solving

loop

propagate // deterministically assign literals

if no conflict **then**

if all variables assigned **then return** solution

else *decide* // non-deterministically assign some literal

else

if top-level conflict **then return** unsatisfiable

else

backtrack // unassign literals propagated after last decision

flip // assign complement of last decision literal

CDCL-style solving

loop

    propagate // deterministically assign literals

    if no conflict **then**

        if all variables assigned **then return** solution

        else *decide* // non-deterministically assign some literal

    else

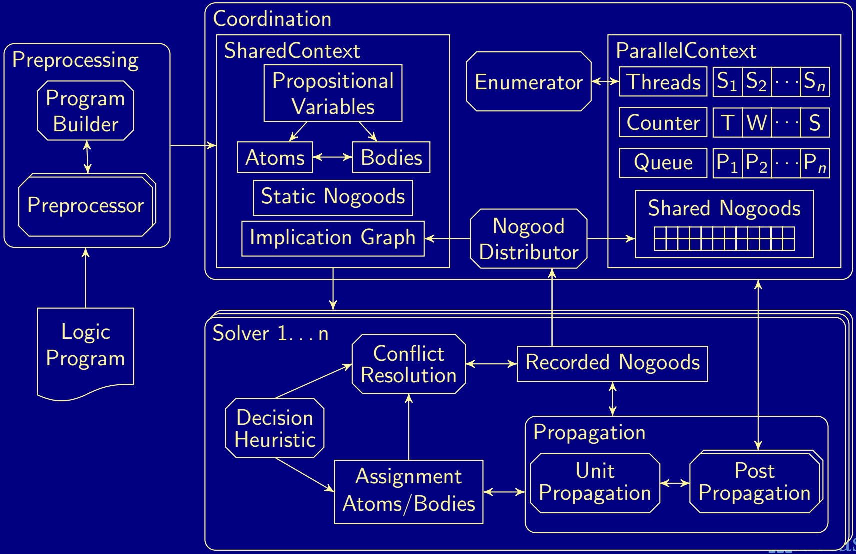
        if top-level conflict **then return** unsatisfiable

        else

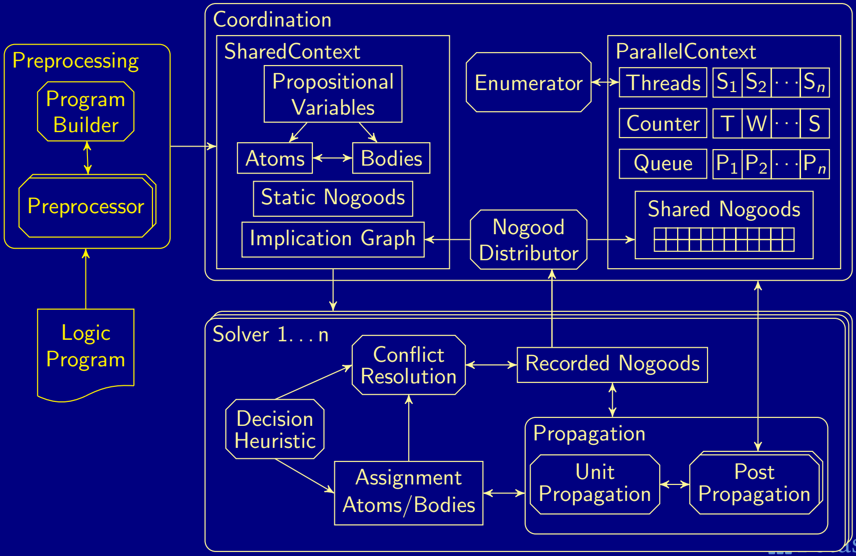
            analyze // analyze conflict and add conflict constraint

            backjump // unassign literals until conflict constraint is unit

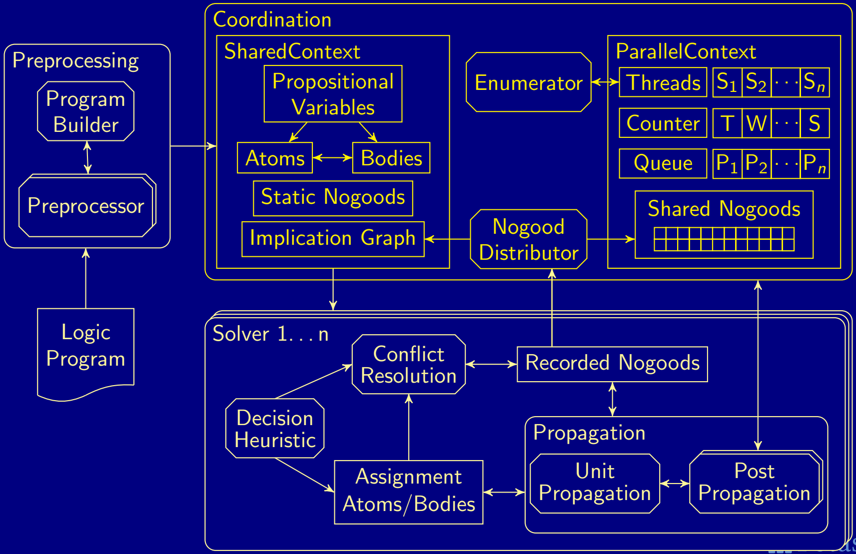
Multi-threaded architecture of *clasp*



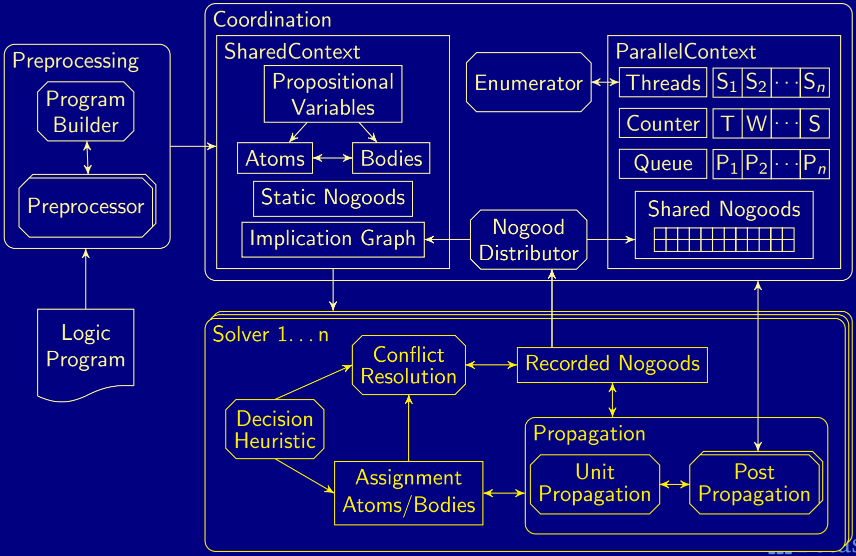
Multi-threaded architecture of *clasp*



Multi-threaded architecture of *clasp*



Multi-threaded architecture of *clasp*



Challenge two

Fact

Boolean constraint technology is rather sensitive to search parameters

Challenge

Robust ASP solving technology

Challenge two

Fact

Boolean constraint technology is rather sensitive to search parameters

Challenge

Robust ASP solving technology

Challenge two

Fact

Boolean constraint technology is rather sensitive to search parameters

Challenge

Robust ASP solving technology — *Taming the oracle !*

Potassco

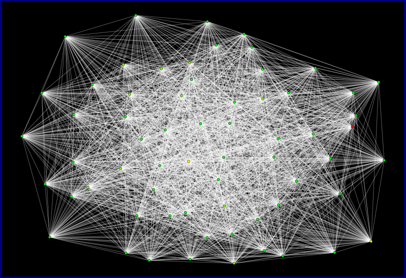
Inside *clasp*, or the encoding's impact

`queens{B,A}.lp, n=8`

Inside *clasp*, or the encoding's impact

queens{B,A}.lp, n=8

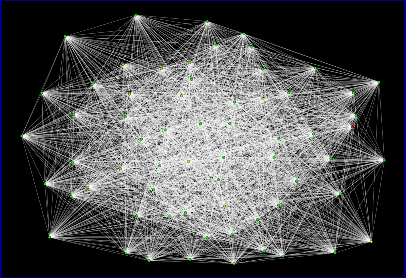
queensB.lp



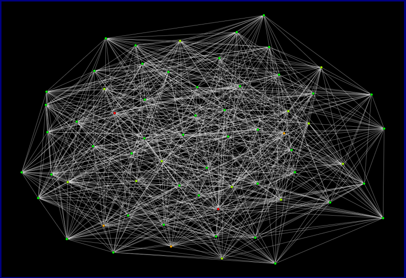
Inside *clasp*, or the encoding's impact

queens{B,A}.lp, n=8

queensB.lp



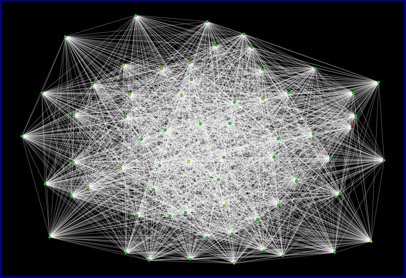
queensA.lp



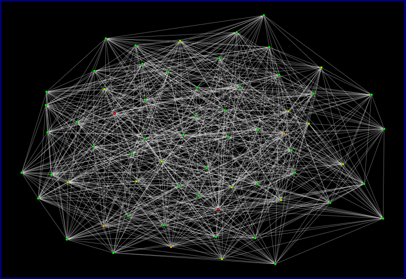
Inside *clasp*, or the encoding's impact

`queens{B,A}.lp, n=8`

`queensB.lp`



`queensA.lp`



Like the pictures. . . ?

➥ Check out Arne König's talk on Tuesday at 16:00+ during TechComm 3

Outline

Configurations

clasp version 2.1.3

```
--configuration=<arg>   : Configure default configuration [frumpy]
    <arg>: frumpy|jumpy|handy|crafty|trendy|chatty
      frumpy: Use conservative defaults
      jumpy : Use aggressive defaults
      handy : Use defaults geared towards large problems
      crafty: Use defaults geared towards crafted problems
      trendy: Use defaults geared towards industrial problems
      chatty: Use 4 competing threads initialized via the default portfolio
```

Comparing configurations
on `queensA.lp`

n	`--frumpy`	`--jumpy`	`--handy`	`--crafty`	`--trendy`	`--chatty`
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Comparing configurations
on queensA.lp

n	--frumpy	--jumpy	--handy	--crafty	--trendy	--chatty
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Comparing configurations
on `queensA.lp`

n	`--frumpy`	`--jumpy`	`--handy`	`--crafty`	`--trendy`	`--chatty`
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Potassco

Comparing configurations

on queensA.lp

n	--frumpy	--jumpy	--handy	--crafty	--trendy	--chatty
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Comparing configurations
on queensA.lp

n	--frumpy	--jumpy	--handy	--crafty	--trendy	--chatty
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

default

(times in seconds, cut-off at 60 seconds)

Comparing configurations
on queensA.lp

n	--frumpy	--jumpy	--handy	--crafty	--trendy	--chatty
50	0.063	0.023	3.416	0.030	1.805	0.061
100	20.364	0.099	7.891	0.136	7.321	0.121
150	60.000	0.212	14.522	0.271	19.883	0.347
200	60.000	0.415	15.026	0.667	32.476	0.753
500	60.000	3.199	60.000	7.471	60.000	6.104

(times in seconds, cut-off at 60 seconds)

Outline

clasp's default portfolio for parallel solving

via `clasp --print-portfolio`

```
[CRAFTY]: --heuristic=vsids --restarts=x,128,1.5 --deletion=3,75,10.0 --del-init-r=1000,9000 --del-grow=1.1,20.
[TRENDY]: --heuristic=vsids --restarts=d,100,0.7 --deletion=3,50 --del-init=500,19500 --del-grow=1.1,20.0,x,100
[FRUMPY]: --heuristic=berkmin --restarts=x,100,1.5 --deletion=1,75 --del-init-r=200,40000 --del-max=400000 --de
[JUMPY]:  --heuristic=vsids --restarts=l,100 --del-init-r=1000,20000 --del-algo=basic,2 --deletion=3,75 --del-g
[STRONG]: --heuristic=berkmin --restarts=x,100,1.5 --deletion=1,75 --del-init-r=200,40000 --del-max=400000 --de
[HANDY]:  --heuristic=vsids --restarts=d,100,0.7 --deletion=2,50,20.0 --del-max=200000 --del-algo=sort,2 --del-
[S2]: --heuristic=vsids --reverse-arcs=1 --otfs=1 --local-restarts --save-progress=0 --contraction=250 --counte
[S4]: --heuristic=vsids --restarts=l,256 --counter-restart=3 --strengthen=recursive --update-lbd --del-glue=2 -
[SLOW]:  --heuristic=berkmin --berk-max=512 --restarts=f,16000 --lookahead=atom,50
[VMTF]:  --heuristic=vmtf --str=no --contr=0 --restarts=x,100,1.3 --del-init-r=800,9200
[SIMPLE]:  --heuristic=vsids  --strengthen=recursive --restarts=x,100,1.5,15 --contraction=0
[LUBY-SP]: --heuristic=vsids --restarts=l,128 --save-p --otfs=1 --init-w=2 --contr=0 --opt-heu=3
[LOCAL-R]: --berk-max=512 --restarts=x,100,1.5,6 --local-restarts --init-w=2 --contr=0
```

- *clasp*'s portfolio is fully customizable
- configurations are assigned in a round-robin fashion to threads during parallel solving
- `--chatty` uses four threads with `CRAFTY`, `TRENDY`, `FRUMPY`, and `JUMPY`

clasp's default portfolio for parallel solving

via `clasp --print-portfolio`

```
[CRAFTY]: --heuristic=vsids --restarts=x,128,1.5 --deletion=3,75,10.0 --del-init-r=1000,9000 --del-grow=1.1,20.
[TRENDY]: --heuristic=vsids --restarts=d,100,0.7 --deletion=3,50 --del-init=500,19500 --del-grow=1.1,20.0,x,100
[FRUMPY]: --heuristic=berkmin --restarts=x,100,1.5 --deletion=1,75 --del-init-r=200,40000 --del-max=400000 --de
[JUMPY]:  --heuristic=vsids --restarts=l,100 --del-init-r=1000,20000 --del-algo=basic,2 --deletion=3,75 --del-g
[STRONG]: --heuristic=berkmin --restarts=x,100,1.5 --deletion=1,75 --del-init-r=200,40000 --del-max=400000 --de
[HANDY]:  --heuristic=vsids --restarts=d,100,0.7 --deletion=2,50,20.0 --del-max=200000 --del-algo=sort,2 --del-
[S2]: --heuristic=vsids --reverse-arcs=1 --otfs=1 --local-restarts --save-progress=0 --contraction=250 --counte
[S4]: --heuristic=vsids --restarts=l,256 --counter-restart=3 --strengthen=recursive --update-lbd --del-glue=2 -
[SLOW]:  --heuristic=berkmin --berk-max=512 --restarts=f,16000 --lookahead=atom,50
[VMTF]:  --heuristic=vmtf --str=no --contr=0 --restarts=x,100,1.3 --del-init-r=800,9200
[SIMPLE]:  --heuristic=vsids  --strengthen=recursive --restarts=x,100,1.5,15 --contraction=0
[LUBY-SP]: --heuristic=vsids --restarts=l,128 --save-p --otfs=1 --init-w=2 --contr=0 --opt-heu=3
[LOCAL-R]: --berk-max=512 --restarts=x,100,1.5,6 --local-restarts --init-w=2 --contr=0
```

- *clasp*'s portfolio is fully customizable
- configurations are assigned in a round-robin fashion to threads during parallel solving
- `--chatty` uses four threads with `CRAFTY`, `TRENDY`, `FRUMPY`, and `JUMPY`

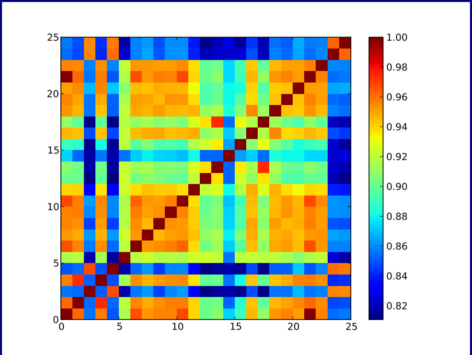
clasp's default portfolio for parallel solving
via `clasp --print-portfolio`

```
[CRAFTY]: --heuristic=vsids --restarts=x,128,1.5 --deletion=3,75,10.0 --del-init-r=1000,9000 --del-grow=1.1,20.
[TRENDY]: --heuristic=vsids --restarts=d,100,0.7 --deletion=3,50 --del-init=500,19500 --del-grow=1.1,20.0,x,100
[FRUMPY]: --heuristic=berkmin --restarts=x,100,1.5 --deletion=1,75 --del-init-r=200,40000 --del-max=400000 --de
[JUMPY]:  --heuristic=vsids --restarts=l,100 --del-init-r=1000,20000 --del-algo=basic,2 --deletion=3,75 --del-g
[STRONG]: --heuristic=berkmin --restarts=x,100,1.5 --deletion=1,75 --del-init-r=200,40000 --del-max=400000 --de
[HANDY]:  --heuristic=vsids --restarts=d,100,0.7 --deletion=2,50,20.0 --del-max=200000 --del-algo=sort,2 --del-
[S2]: --heuristic=vsids --reverse-arcs=1 --otfs=1 --local-restarts --save-progress=0 --contraction=250 --counte
[S4]: --heuristic=vsids --restarts=l,256 --counter-restart=3 --strengthen=recursive --update-lbd --del-glue=2 -
[SLOW]:  --heuristic=berkmin --berk-max=512 --restarts=f,16000 --lookahead=atom,50
[VMTF]:  --heuristic=vmtf --str=no --contr=0 --restarts=x,100,1.3 --del-init-r=800,9200
[SIMPLE]:  --heuristic=vsids  --strengthen=recursive --restarts=x,100,1.5,15 --contraction=0
[LUBY-SP]: --heuristic=vsids --restarts=l,128 --save-p --otfs=1 --init-w=2 --contr=0 --opt-heu=3
[LOCAL-R]: --berk-max=512 --restarts=x,100,1.5,6 --local-restarts --init-w=2 --contr=0
```

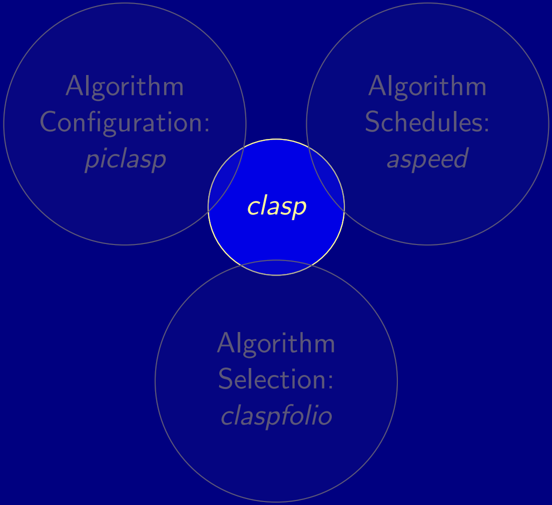
- *clasp*'s portfolio is fully customizable
- configurations are assigned in a round-robin fashion to threads during parallel solving
- `--chatty` uses four threads with `CRAFTY`, `TRENDY`, `FRUMPY`, and `JUMPY`

Outline

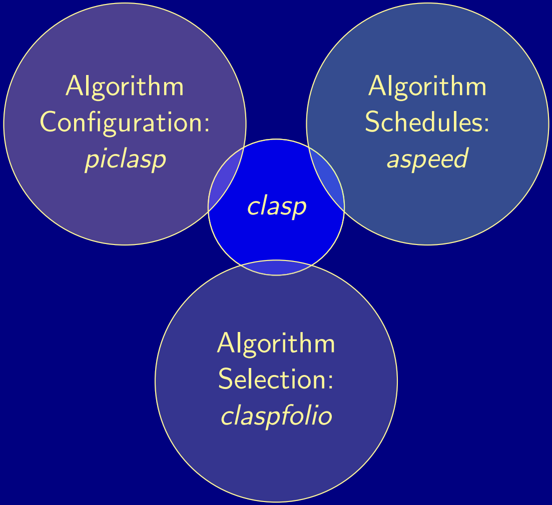
Correlation of *clasp* configurations



Algorithm engineering



Algorithm engineering



piclasp

Task

Identify an individual configuration for solving a specific problem class (having a homogeneous instance set)

Approach

Use an algorithm configurator (eg *SMAC* or *ParamILS*) for finding a well performing configuration

piclasp

Task

Identify an individual configuration for solving a specific problem class (having a homogeneous instance set)

Approach

Use an algorithm configurator (eg *SMAC* or *ParamILS*) for finding a well performing configuration

piclasp's search space

```
Clasp - Search Options:

  --heuristic=<arg>       : Configure decision heuristic
      <arg>: Berkmin|Vmtf|Vsids|Unit|None
        Berkmin: Apply BerkMin-like heuristic
        Vmtf   : Apply Siege-like heuristic
        Vsids  : Apply Chaff-like heuristic
        Unit   : Apply Smodels-like heuristic (Default if --no-lookback)
        None   : Select the first free variable
  --[no-]init-moms        : Initialize heuristic with MOMS-score
  --score-other=<n>       : Score 0=no|1=loop|2=all other learnt nogoods
  --sign-def=<n>          : Default sign: 0=type|1=no|2=yes|3=rnd
  --[no-]sign-fix         : Disable sign heuristics and use default signs only
  --berk-max=<n>          : Consider at most <n> nogoods in Berkmin heuristic
  --[no-]berk-huang       : Enable/Disable Huang-scoring in Berkmin
  --[no-]berk-once        : Score sets (instead of multisets) in Berkmin
  --vmtf-mtf=<n>          : In Vmtf move <n> conflict-literals to the front
  --vsids-decay=<n>       : In Vsids use 1.0/0.<n> as decay factor
  --[no-]nant             : In Unit count only atoms in NAnt(P)
  --opt-heuristic[=0..3]: Use opt. in 1=sign|2=model|3=both heuristics
  --save-progress[=<n>]   : Use RSat-like progress saving on backjumps > <n>
  --rand-freq=<p>         : Make random decisions with probability <p>
  --init-watches=0..2   : Configure watched literal initialization [1]
      Watch 0=first|1=random|2=least watched literals in nogoods
  --seed=<n>              : Set random number generator's seed to <n>

  --lookahead[=<arg>|no]  : Configure failed-literal detection (fld)
      <arg>: <type>[,<n 1..umax>] / Implicit: atom
        <type>: Run fld via atom|body|hybrid lookahead
        <n>   : Disable fld after <n> applications ([-1]=no limit)
```

aspeed

Task

Synthesize a timeout- and time-minimal schedule of configurations for solving a heterogeneous set of problem instances

Approach

Use ASP (and runtime data) for finding such a schedule

aspeed

Task

Synthesize a timeout- and time-minimal schedule of configurations for solving a heterogeneous set of problem instances

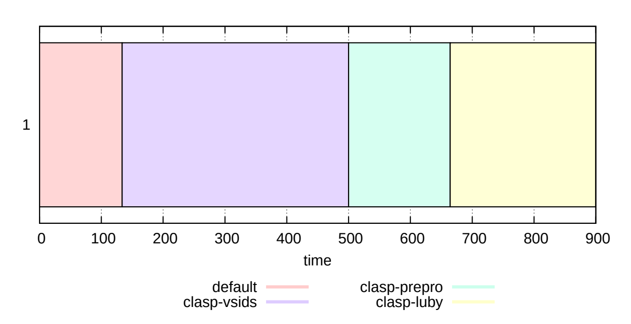
Approach

Use ASP (and runtime data) for finding such a schedule

aspeed's basic encoding

```
solver(S)  :- time(_,S,_).
time(S,T)  :- time(_,S,T).
unit(1..N) :- units(N).

{ slice(U,S,T) : time(S,T) : T <= K : unit(U) } 1 :- solver(S), kappa(K).

 :- not [ slice(U,S,T) = T ] K, kappa(K), unit(U).

slice(S,T) :- slice(_,S,T).
solved(I,S) :- slice(S,T), time(I,S,T).
solved(I,S) :- solved(J,S), order(I,J,S).
solved(I)   :- solved(I,_).

#maximize { solved(I) @ 2 }.
#minimize [ slice(S,T) = T*T @ 1 ].
```

A resulting schedule



claspfolio

Task

Select an individual configuration for solving a specific problem instance (from a heterogeneous instance set)

Approach

Use instance features to select a promising configuration from a portfolio via trained classifiers

claspfolio

Task

Select an individual configuration for solving a specific problem instance (from a heterogeneous instance set)

Approach

Use instance features to select a promising configuration from a portfolio via trained classifiers

claspre features

- Plain instance features
  - Number of atoms
  - Number of rule types
  - ...
- Features after preprocessing
  - Tightness
  - Equivalences between atoms and bodies
  - Number of constraint types
  - ...
- Search features after restarting
  - Number of choices
  - Number of types of learnt nogoods
  - Number of deleted nogoods
  - Average backjump length
  - ...

All in all $32 + 25 \cdot 2$ features are calculated

claspre features

- Plain instance features
  - Number of atoms
  - Number of rule types
  - ...
- Features after preprocessing
  - Tightness
  - Equivalences between atoms and bodies
  - Number of constraint types
  - ...
- Search features after restarting
  - Number of choices
  - Number of types of learnt nogoods
  - Number of deleted nogoods
  - Average backjump length
  - ...

All in all $32 + 25 \cdot 2$ features are calculated

claspre features

- Plain instance features
  - Number of atoms
  - Number of rule types
  - ...
- Features after preprocessing
  - Tightness
  - Equivalences between atoms and bodies
  - Number of constraint types
  - ...
- Search features after restarting
  - Number of choices
  - Number of types of learnt nogoods
  - Number of deleted nogoods
  - Average backjump length
  - ...

All in all $32 + 25 \cdot 2$ features are calculated

claspre features

- Plain instance features
  - Number of atoms
  - Number of rule types
  - ...
- Features after preprocessing
  - Tightness
  - Equivalences between atoms and bodies
  - Number of constraint types
  - ...
- Search features after restarting
  - Number of choices
  - Number of types of learnt nogoods
  - Number of deleted nogoods
  - Average backjump length
  - ...

All in all $32 + 25 \cdot 2$ features are calculated

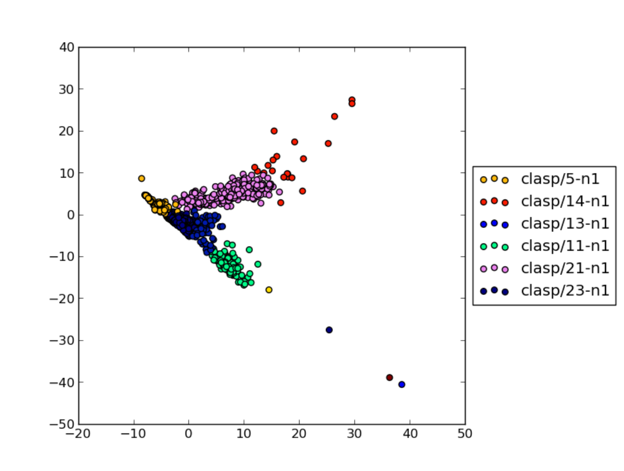
claspre features

- Plain instance features
  - Number of atoms
  - Number of rule types
  - ...
- Features after preprocessing
  - Tightness
  - Equivalences between atoms and bodies
  - Number of constraint types
  - ...
- Search features after restarting
  - Number of choices
  - Number of types of learnt nogoods
  - Number of deleted nogoods
  - Average backjump length
  - ...

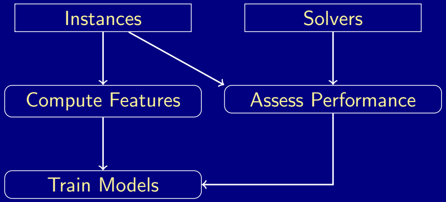
All in all $32 + 25 \cdot 2$ features are calculated

Potassco

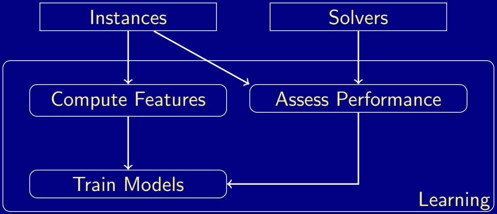
Feature space in practice



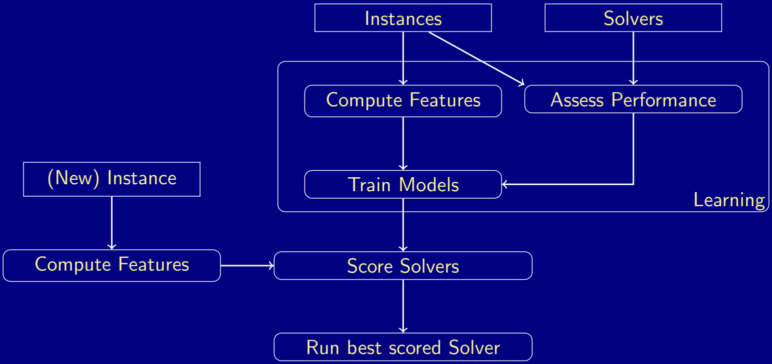
claspfolio's architecture



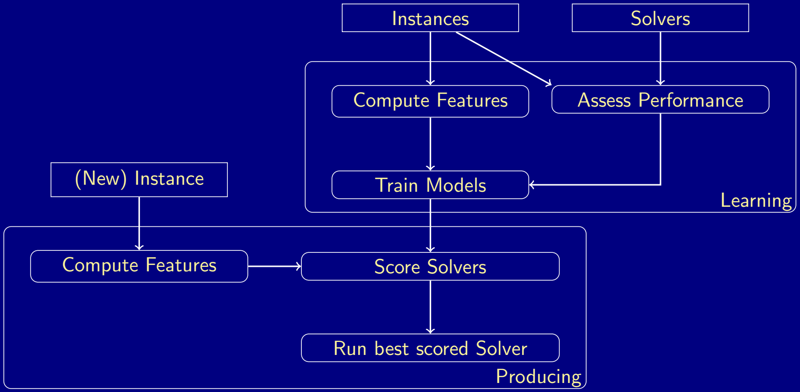
claspfolio's architecture



claspfolio's architecture



claspfolio's architecture



Outline

hclasp

- *hclasp* allows for incorporating domain-specific heuristics
  - input language for expressing domain-specific heuristics
  - solving capacities for integrating domain-specific heuristics

- Example
  - Extend your encoding, enc.lp, by a heuristic rule like

```
_heuristic(occ(A,T),factor,T) :- action(A),time(T).
```

and the heuristic information via a #show statement

Ground the program (as usual) and make hclasp notice your heuristic modifications

```
$ gringo enc.lp | hclasp --heuristic=domain
```

hclasp

- *hclasp* allows for incorporating domain-specific heuristics
  - input language for expressing domain-specific heuristics
  - solving capacities for integrating domain-specific heuristics

- Example
  - Extend your encoding, `enc.lp`, by a heuristic rule like

    ```
    _heuristic(occ(A,T),factor,T) :- action(A),time(T).
    ```

    and the heuristic information via a `#show` statement

  - Ground the program (as usual) and make `hclasp` notice your heuristic modifications

    ```
    $ gringo enc.lp | hclasp --heuristic=domain
    ```

hclasp

- *hclasp* allows for incorporating domain-specific heuristics
  - input language for expressing domain-specific heuristics
  - solving capacities for integrating domain-specific heuristics

- Example
  - Extend your encoding, `enc.lp`, by a heuristic rule like

    ```
    _heuristic(occ(A,T),factor,T) :- action(A),time(T).
    ```

    and the heuristic information via a `#show` statement
  - Ground the program (as usual) and make `hclasp` notice your heuristic modifications

    ```
    $ gringo enc.lp | hclasp --heuristic=domain
    ```

Basic CDCL decision algorithm

loop

    propagate // compute deterministic consequences

    if no conflict **then**

        if all variables assigned **then return** variable assignment

        else decide // non-deterministically assign some literal

    else

        if top-level conflict **then return** unsatisfiable

        else

            analyze // analyze conflict and add a conflict constraint

            backjump // undo assignments until conflict constraint is unit

Basic CDCL decision algorithm

loop

    propagate   // compute deterministic consequences
    if no conflict **then**
        if all variables assigned **then return** variable assignment
        else decide   // non-deterministically assign some literal
    else
        if top-level conflict **then return** unsatisfiable
        else
            analyze   // analyze conflict and add a conflict constraint
            backjump   // undo assignments until conflict constraint is unit

Inside *decide*

- Heuristic functions

$$h : \mathcal{A} \rightarrow [0, +\infty) \quad \text{and} \quad s : \mathcal{A} \rightarrow \{\mathbf{T}, \mathbf{F}\}$$

- Algorithmic scheme

1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
2. $U := \mathcal{A} \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}})$
3. $C := argmax_{a \in U} h(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto s(a)\}$

Potassco

Inside *decide*

- Heuristic functions

$$h : \mathcal{A} \rightarrow [0, +\infty) \quad \text{and} \quad s : \mathcal{A} \rightarrow \{\mathbf{T}, \mathbf{F}\}$$

- Algorithmic scheme

1. $h(a) := \alpha \times h(a) + \beta(a)$ — for each $a \in \mathcal{A}$
2. $U := \mathcal{A} \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}})$
3. $C := argmax_{a \in U} h(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto s(a)\}$

Potassco

Inside *decide*

- Heuristic functions

$$h : \mathcal{A} \rightarrow [0, +\infty) \quad \text{and} \quad s : \mathcal{A} \rightarrow \{\mathbf{T}, \mathbf{F}\}$$

- Algorithmic scheme

1. $h(a) := \alpha \times h(a) + \beta(a)$ for each $a \in \mathcal{A}$
2. $U := \mathcal{A} \setminus (A^{\mathbf{T}} \cup A^{\mathbf{F}})$
3. $C := argmax_{a \in U} h(a)$
4. $a := \tau(C)$
5. $A := A \cup \{a \mapsto s(a)\}$

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, *a*, and integer, *v*)

  `init` for initializing the heuristic value of *a* with *v*
  `factor` for amplifying the heuristic value of *a* by factor *v*
  `level` for ranking all atoms; the rank of *a* is *v*
  `sign` for attributing the sign of *v* as truth value to *a*

- Heuristic atoms

```
_heuristic(occurs(move),factor,5)
```

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, *a*, and integer, *v*)
  - `init` for initializing the heuristic value of *a* with *v*
  - `factor` for amplifying the heuristic value of *a* by factor *v*
  - `level` for ranking all atoms; the rank of *a* is *v*
  - `sign` for attributing the sign of *v* as truth value to *a*
- Heuristic atoms

```
_heuristic(occurs(move),factor,5)
```

Heuristic language elements

- Heuristic predicate `_heuristic`
- Heuristic modifiers (atom, *a*, and integer, *v*)

`init`	for initializing the heuristic value of *a* with *v*
`factor`	for amplifying the heuristic value of *a* by factor *v*
`level`	for ranking all atoms; the rank of *a* is *v*
`sign`	for attributing the sign of *v* as truth value to *a*

- Heuristic atoms

```
_heuristic(occurs(move),factor,5)
```

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).
```

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).

_heuristic(occurs(A,T),factor,2) :- action(A), time(T).
```

Potassco

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).

_heuristic(occurs(A,T),level,1) :- action(A), time(T).
```

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).

_heuristic(occurs(A,T),factor,T) :- action(A), time(T).
```

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).

_heuristic(A,level,V) :- _heuristic(A,true, V).
_heuristic(A,sign, 1) :- _heuristic(A,true, V).
```

Simple STRIPS planner

```
time(1..t).

holds(P,0) :- init(P).

1 { occurs(A,T) : action(A) } 1 :- time(T).
 :- occurs(A,T), pre(A,F), not holds(F,T-1).

holds(F,T) :- holds(F,T-1), not nolds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nolds(F,T) :- occurs(A,T), del(A,F).

 :- query(F), not holds(F,t).

_heuristic(A,level,V) :- _heuristic(A,false,V).
_heuristic(A,sign,-1) :- _heuristic(A,false,V).
```

Planning Competition Benchmarks

```
_heuristic(holds(F,T-1),true, t-T+1) :- holds(F,T).
_heuristic(holds(F,T-1),false,t-T+1) :-
                  fluent(F), time(T), not holds(F,T).
```

Problem	*base configuration*		`_heuristic`		*base c. (SAT)*		`_heur.` *(SAT)*	
Blocks'00	134.4*s*	(180/61)	9.2*s*	(239/3)	163.2*s*	(59)	2.6*s*	(0)
Elevator'00	3.1*s*	(279/0)	0.0*s*	(279/0)	3.4*s*	(0)	0.0*s*	(0)
Freecell'00	288.7*s*	(147/115)	184.2*s*	(194/74)	226.4*s*	(47)	52.0*s*	(0)
Logistics'00	145.8*s*	(148/61)	115.3*s*	(168/52)	113.9*s*	(23)	15.5*s*	(3)
Depots'02	400.3*s*	(51/184)	297.4*s*	(115/135)	389.0*s*	(64)	61.6*s*	(0)
Driverlog'02	308.3*s*	(108/143)	189.6*s*	(169/92)	245.8*s*	(61)	6.1*s*	(0)
Rovers'02	245.8*s*	(138/112)	165.7*s*	(179/79)	162.9*s*	(41)	5.7*s*	(0)
Satellite'02	398.4*s*	(73/186)	229.9*s*	(155/106)	364.6*s*	(82)	30.8*s*	(0)
Zenotravel'02	350.7*s*	(101/169)	239.0*s*	(154/116)	224.5*s*	(53)	6.3*s*	(0)
Total	252.8*s*	(1225/1031)	158.9*s*	(1652/657)	187.2*s*	(430)	17.1*s*	(3)

Planning Competition Benchmarks

```
_heuristic(holds(F,T-1),true, t-T+1) :- holds(F,T).
_heuristic(holds(F,T-1),false,t-T+1) :-
                fluent(F), time(T), not holds(F,T).
```

Problem	*base configuration*		`_heuristic`		*base c. (SAT)*		`_heur.` *(SAT)*	
Blocks'00	134.4*s*	(180/61)	9.2*s*	(239/3)	163.2*s*	(59)	2.6*s*	(0)
Elevator'00	3.1*s*	(279/0)	0.0*s*	(279/0)	3.4*s*	(0)	0.0*s*	(0)
Freecell'00	288.7*s*	(147/115)	184.2*s*	(194/74)	226.4*s*	(47)	52.0*s*	(0)
Logistics'00	145.8*s*	(148/61)	115.3*s*	(168/52)	113.9*s*	(23)	15.5*s*	(3)
Depots'02	400.3*s*	(51/184)	297.4*s*	(115/135)	389.0*s*	(64)	61.6*s*	(0)
Driverlog'02	308.3*s*	(108/143)	189.6*s*	(169/92)	245.8*s*	(61)	6.1*s*	(0)
Rovers'02	245.8*s*	(138/112)	165.7*s*	(179/79)	162.9*s*	(41)	5.7*s*	(0)
Satellite'02	398.4*s*	(73/186)	229.9*s*	(155/106)	364.6*s*	(82)	30.8*s*	(0)
Zenotravel'02	350.7*s*	(101/169)	239.0*s*	(154/116)	224.5*s*	(53)	6.3*s*	(0)
Total	252.8*s*	(1225/1031)	158.9*s*	(1652/657)	187.2*s*	(430)	17.1*s*	(3)

Potassco

Planning Competition Benchmarks

```
_heuristic(holds(F,T-1),true, t-T+1) :- holds(F,T).
_heuristic(holds(F,T-1),false,t-T+1) :-
                fluent(F), time(T), not holds(F,T).
```

Problem	*base configuration*		`_heuristic`		*base c.* *(SAT)*		`_heur.` *(SAT)*	
Blocks'00	134.4*s*	(180/61)	9.2*s*	(239/3)	163.2*s*	(59)	2.6*s*	(0)
Elevator'00	3.1*s*	(279/0)	0.0*s*	(279/0)	3.4*s*	(0)	0.0*s*	(0)
Freecell'00	288.7*s*	(147/115)	184.2*s*	(194/74)	226.4*s*	(47)	52.0*s*	(0)
Logistics'00	145.8*s*	(148/61)	115.3*s*	(168/52)	113.9*s*	(23)	15.5*s*	(3)
Depots'02	400.3*s*	(51/184)	297.4*s*	(115/135)	389.0*s*	(64)	61.6*s*	(0)
Driverlog'02	308.3*s*	(108/143)	189.6*s*	(169/92)	245.8*s*	(61)	6.1*s*	(0)
Rovers'02	245.8*s*	(138/112)	165.7*s*	(179/79)	162.9*s*	(41)	5.7*s*	(0)
Satellite'02	398.4*s*	(73/186)	229.9*s*	(155/106)	364.6*s*	(82)	30.8*s*	(0)
Zenotravel'02	350.7*s*	(101/169)	239.0*s*	(154/116)	224.5*s*	(53)	6.3*s*	(0)
Total	252.8*s*	(1225/1031)	158.9*s*	(1652/657)	187.2*s*	(430)	17.1*s*	(3)

Potassco

Outline

Challenge three (or: one+two)

Fact

Many real-world applications involve optimization

Challenge

Theory and Tools for versatile optimization methods

Challenge three (or: one+two)

Fact

Many real-world applications involve optimization

Challenge

Theory and Tools for versatile optimization methods

Alternative ways of optimization

- Branch-and-Bound optimization in *clasp*
- Hierarchical Branch-and-Bound optimization in *clasp*
- Unsatisfiability-based optimization in *unclasp*
- Incremental optimization in *iclingo*

- Saturation-based optimization in *metasp* (via *claspD*)
- Heuristic-driven optimization in *hclasp*

Alternative ways of optimization

- Branch-and-Bound optimization in *clasp*
- Hierarchical Branch-and-Bound optimization in *clasp*
- Unsatisfiability-based optimization in *unclasp*
- Incremental optimization in *iclingo*

- Saturation-based optimization in *metasp* (via *claspD*)
- Heuristic-driven optimization in *hclasp*

Alternative ways of optimization

- Branch-and-Bound optimization in *clasp*
  - **SAT ... SAT UNSAT**
- Hierarchical Branch-and-Bound optimization in *clasp*
- Unsatisfiability-based optimization in *unclasp*
- Incremental optimization in *iclingo*

- Saturation-based optimization in *metasp* (via *claspD*)
- Heuristic-driven optimization in *hclasp*

Alternative ways of optimization

- Branch-and-Bound optimization in *clasp*
  - **SAT . . . SAT UNSAT**
- Hierarchical Branch-and-Bound optimization in *clasp*
  - **SAT . . . SAT UNSAT SAT . . . SAT UNSAT SAT . . . SAT UNSAT**
- Unsatisfiability-based optimization in *unclasp*
  - **(UNSAT UNSAT . . .) SAT**
- Incremental optimization in *iclingo*
  - **UNSAT . . . UNSAT SAT**

- Saturation-based optimization in *metasp* (via *claspD*)
- Heuristic-driven optimization in *hclasp*

Alternative ways of optimization

- Branch-and-Bound optimization in *clasp*
  - **SAT . . . SAT UNSAT**
- Hierarchical Branch-and-Bound optimization in *clasp*
  - **SAT . . . SAT UNSAT SAT . . . SAT UNSAT SAT . . . SAT UNSAT**
- Unsatisfiability-based optimization in *unclasp*
  - **(UNSAT UNSAT . . .) SAT**
- Incremental optimization in *iclingo*
  - **UNSAT . . . UNSAT SAT**

- Saturation-based optimization in *metasp* (via *claspD*)
  - **(SAT$\circ$UNSAT . . .) SAT$\circ$UNSAT**
- Heuristic-driven optimization in *hclasp*
  - **SAT**

Potassco

Outline

Challenge four (or: one+one+two)

Fact

Intelligence is build around us and in our pockets

Challenge

Incremental and reactive ASP solving technology

Challenge four (or: one+one+two)

Fact

Intelligence is build around us and in our pockets

Challenge

Incremental and reactive ASP solving technology

Potassco

Going online

- Planning and reasoning about action with *iclingo*

- Sliding windows in stream reasoning with *oclingo*
- Interactive query-answering with *oclingo*
- Cognitive robotics with *ROSoClingo*

Going online

- Planning and reasoning about action with *iclingo*

- Sliding windows in stream reasoning with *oclingo*
- Interactive query-answering with *oclingo*
- Cognitive robotics with *ROSoClingo*

"Ke Jia" robots
(X. Chen, UST China)

Going online

- Planning and reasoning about action with *iclingo*

- Sliding windows in stream reasoning with *oclingo*
- Interactive query-answering with *oclingo*
- Cognitive robotics with *ROSoClingo*

"Ke Jia" robots
(X. Chen, UST China)

Potassco

Outline

Summary

Declarativity

ASP separates a problem's representation from the algorithms used for solving it

Scalability

There is no free lunch !

Challenges

- Modeling
- Solving
- Optimizing
- Reacting

Visit us !

`potassco.sourceforge.net`

- free ASP systems
- open source software
- teaching material

Potassco

Summary

Declarativity

ASP separates a problem's representation from the algorithms used for solving it

Scalability

There is no free lunch !

Challenges

- Modeling
- Solving
- Optimizing
- Reacting

Visit us !

potassco.sourceforge.net

- free ASP systems
- open source software
- teaching material

Potassco

Summary

Declarativity

ASP separates a problem's representation from the algorithms used for solving it

Scalability

There is no free lunch !

Challenges

- Modeling
- Solving
- Optimizing
- Reacting

Visit us !

potassco.sourceforge.net

- free ASP systems
- open source software
- teaching material

Potassco

Summary

Declarativity

ASP separates a problem's representation from the algorithms used for solving it

Scalability

There is no free lunch !

Challenges

- Modeling
- Solving
- Optimizing
- Reacting

Visit us !

potassco.sourceforge.net

- free ASP systems
- open source software
- teaching material

Potassco

Summary

Declarativity

ASP separates a problem's representation from the algorithms used for solving it

Scalability

There is no free lunch !

Challenges

- Modeling
- Solving
- Optimizing
- Reacting

Visit us !

`potassco.sourceforge.net`

- free ASP systems
- open source software
- teaching material

Potassco

Potassco is a composition of people

Benjamin Andres ◦ Christian Anger ◦ Farid Benhammadi ◦ Philippe Besnard ◦ Paul Borchert ◦ Christian Drescher ◦ Steve Dworschak ◦ Johannes Fichte ◦ André Flöter ◦ Martin Gebser ◦ Mona Gharib ◦ Susanne Grell ◦ Jean Gressmann ◦ Torsten Grote ◦ Holger Jost ◦ Roland Kaminski ◦ Benjamin Kaufmann ◦ Kathrin Konczak ◦ Murat Knecht ◦ Arne König ◦ Thomas Linke ◦ Benjamin Lüpfert ◦ Oliver Matheis ◦ André Neumann ◦ Pascal Nicolas ◦ Philipp Obermeier ◦ Max Ostrowski ◦ Javier Romero ◦ Orkunt Sabuncu ◦ Vladimir Sarsakov ◦ Marius Schneider ◦ Sven Thiele ◦ Richard Tichy ◦ Santiago Videla ◦ Philippe Veber ◦ Kewen Wang ◦ Philipp Wanko ◦ Matthias Weise ◦ Peter-Uwe Zettiér ◦ Stefan Ziller

Dankeschön! Et merci!

Potassco

Potassco is a composition of people

Benjamin Andres ◦ Christian Anger ◦ Farid Benhammadi ◦ Philippe Besnard ◦ Paul Borchert ◦ Christian Drescher ◦ Steve Dworschak ◦ Johannes Fichte ◦ André Flöter ◦ Martin Gebser ◦ Mona Gharib ◦ Susanne Grell ◦ Jean Gressmann ◦ Torsten Grote ◦ Holger Jost ◦ Roland Kaminski ◦ Benjamin Kaufmann ◦ Kathrin Konczak ◦ Murat Knecht ◦ Arne König ◦ Thomas Linke ◦ Benjamin Lüpfert ◦ Oliver Matheis ◦ André Neumann ◦ Pascal Nicolas ◦ Philipp Obermeier ◦ Max Ostrowski ◦ Javier Romero ◦ Orkunt Sabuncu ◦ Vladimir Sarsakov ◦ Marius Schneider ◦ Sven Thiele ◦ Richard Tichy ◦ Santiago Videla ◦ Philippe Veber ◦ Kewen Wang ◦ Philipp Wanko ◦ Matthias Weise ◦ Peter-Uwe Zettiér ◦ Stefan Ziller

Dankeschön! Et merci!

Potassco