Experiencing Answer Set Programming at Work Today and Tomorrow

Torsten Schaub
University of Potsdam

Potassco
1 Introduction
2 Modeling
3 Solving
   • Conflict-driven search
   • Solver configurations
   • Parallel solving
   • Automatic solver engineering
   • Domain-specific heuristics
4 Optimizing
5 Reacting
6 Summary
1 Introduction

2 Modeling

3 Solving
   - Conflict-driven search
   - Solver configurations
   - Parallel solving
   - Automatic solver engineering
   - Domain-specific heuristics

4 Optimizing

5 Reacting

6 Summary
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?”

Problem -> Computer

Solution

Output
Traditional programming

“What is the problem?” versus “How to solve the problem?”

Programming

Problem

Program

Executing

Solution

Interpreting

Output
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?”

- Problem
  - Modeling
  - Representation

- Solution
  - Interpreting
  - Output

Solving
Declarative problem solving

“What is the problem?” versus “How to solve the problem?”
Introduction

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 \textit{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
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
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
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
# Model Generation based Problem Solving

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>stable models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>expansions</td>
</tr>
<tr>
<td>first-order theories</td>
<td>extensions</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td></td>
</tr>
<tr>
<td>default theories</td>
<td></td>
</tr>
</tbody>
</table>
# Model Generation based Problem Solving

## Representation
- constraint satisfaction problem
- propositional horn theories
- propositional theories
- propositional programs
- first-order theories
- first-order theories
- first-order theories
- auto-epistemic theories
- default theories
- :

## Solution
- assignment
- smallest model
- models
- minimal models
- stable models
- minimal models
- supported models
- stable models
- minimal models
- stable models
- Herbrand models
- expansions
- extensions
- :

---

*Potassco*

Torsten Schaub (KRR@UP)  
Experiencing ASP at Work
# Model Generation based Problem Solving

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>stable models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>expansions</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td>extensions</td>
</tr>
<tr>
<td>default theories</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
</tr>
</tbody>
</table>

SAT
## Model Generation based Problem Solving

<table>
<thead>
<tr>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
</tr>
<tr>
<td>propositional horn theories</td>
</tr>
<tr>
<td>propositional theories</td>
</tr>
<tr>
<td>propositional theories</td>
</tr>
<tr>
<td>propositional programs</td>
</tr>
<tr>
<td>propositional programs</td>
</tr>
<tr>
<td>propositional programs</td>
</tr>
<tr>
<td>first-order theories</td>
</tr>
<tr>
<td>first-order theories</td>
</tr>
<tr>
<td>first-order theories</td>
</tr>
<tr>
<td>first-order theories</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
</tr>
<tr>
<td>default theories</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignment</td>
</tr>
<tr>
<td>smallest model</td>
</tr>
<tr>
<td>models</td>
</tr>
<tr>
<td>minimal models</td>
</tr>
<tr>
<td>stable models</td>
</tr>
<tr>
<td>minimal models</td>
</tr>
<tr>
<td>supported models</td>
</tr>
<tr>
<td>stable models</td>
</tr>
<tr>
<td>minimal models</td>
</tr>
<tr>
<td>stable models</td>
</tr>
<tr>
<td>Herbrand models</td>
</tr>
<tr>
<td>expansions</td>
</tr>
<tr>
<td>extensions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
</tr>
<tr>
<td>NMR</td>
</tr>
</tbody>
</table>

Torsten Schaub (KRR@UP)  
Experiencing ASP at Work
Introduction

Model Generation based Problem Solving

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>minimal models</td>
<td></td>
</tr>
<tr>
<td>stable models</td>
<td></td>
</tr>
<tr>
<td>supported models</td>
<td></td>
</tr>
<tr>
<td>stable models</td>
<td></td>
</tr>
<tr>
<td>minimal models</td>
<td></td>
</tr>
<tr>
<td>supported models</td>
<td></td>
</tr>
<tr>
<td>stable models</td>
<td></td>
</tr>
<tr>
<td>minimal models</td>
<td></td>
</tr>
<tr>
<td>supported models</td>
<td></td>
</tr>
<tr>
<td>stable models</td>
<td></td>
</tr>
<tr>
<td>Herbrand models</td>
<td></td>
</tr>
<tr>
<td>expansions</td>
<td></td>
</tr>
<tr>
<td>extensions</td>
<td></td>
</tr>
</tbody>
</table>

SAT
ASP
NMR
NMR

Torsten Schaub (KRR@UP)
Experiencing ASP at Work
# Answer Set Programming in general

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td><strong>propositional theories</strong></td>
<td><strong>stable models</strong></td>
</tr>
<tr>
<td>propositional programs</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
</tr>
<tr>
<td><strong>propositional programs</strong></td>
<td><strong>stable models</strong></td>
</tr>
<tr>
<td>first-order theories</td>
<td>models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td><strong>first-order theories</strong></td>
<td><strong>stable models</strong></td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td>expansions</td>
</tr>
<tr>
<td>default theories</td>
<td>extensions</td>
</tr>
<tr>
<td><strong>:</strong></td>
<td><strong>:</strong></td>
</tr>
</tbody>
</table>

ASP
# Answer Set Programming *in general*

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td><strong>propositional theories</strong></td>
<td><strong>stable models</strong> ASP</td>
</tr>
<tr>
<td>propositional programs</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
</tr>
<tr>
<td><strong>propositional programs</strong></td>
<td><strong>stable models</strong> ASP</td>
</tr>
<tr>
<td>first-order theories</td>
<td>models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td><strong>first-order theories</strong></td>
<td><strong>stable models</strong> ASP</td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td>expansions</td>
</tr>
<tr>
<td>default theories</td>
<td>extensions</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Answer Set Programming *in practice*

<table>
<thead>
<tr>
<th>Representation</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint satisfaction problem</td>
<td>assignment</td>
</tr>
<tr>
<td>propositional horn theories</td>
<td>smallest model</td>
</tr>
<tr>
<td>propositional theories</td>
<td>models</td>
</tr>
<tr>
<td>propositional theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>stable models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>minimal models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>supported models</td>
</tr>
<tr>
<td>propositional programs</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>minimal models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>stable models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>Herbrand models</td>
</tr>
<tr>
<td>first-order theories</td>
<td>expansions</td>
</tr>
<tr>
<td>auto-epistemic theories</td>
<td>extensions</td>
</tr>
<tr>
<td>default theories</td>
<td></td>
</tr>
<tr>
<td>first-order programs</td>
<td>stable Herbrand models</td>
</tr>
</tbody>
</table>
### ASP versus LP

<table>
<thead>
<tr>
<th></th>
<th>ASP</th>
<th>Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model generation</td>
<td>Model generation</td>
<td>Query orientation</td>
</tr>
<tr>
<td>Bottom-up</td>
<td>Bottom-up</td>
<td>Top-down</td>
</tr>
<tr>
<td>Modeling language</td>
<td>Modeling language</td>
<td>Programming language</td>
</tr>
</tbody>
</table>

**Rule-based format**

- Instantiation
- Flat terms
- \((Turing +)^{NP(NP)}\)
- Unification
- Nested terms
- Turing
<table>
<thead>
<tr>
<th>ASP</th>
<th>SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model generation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Bottom-up</strong></td>
<td></td>
</tr>
<tr>
<td>Constructive Logic</td>
<td>Classical Logic</td>
</tr>
<tr>
<td>Closed (and open) world reasoning</td>
<td>Open world reasoning</td>
</tr>
<tr>
<td>Modeling language</td>
<td></td>
</tr>
<tr>
<td>Complex reasoning modes</td>
<td>Satisfiability testing</td>
</tr>
<tr>
<td>Satisfiability</td>
<td>Satisfiability</td>
</tr>
<tr>
<td>Enumeration/Projection</td>
<td></td>
</tr>
<tr>
<td>Intersection/Union</td>
<td></td>
</tr>
<tr>
<td>Optimization</td>
<td></td>
</tr>
<tr>
<td>((\text{Turing } +) \ NP^{(NP)})</td>
<td>(NP)</td>
</tr>
</tbody>
</table>
ASP solving

- Problem
- Logic Program
- Grounder
- Solver
- Stable Models
- Solution

Modeling

Interpreting

Solving
Rooting ASP solving

Modeling → KR → Logic Program → Grounder → Solver → Stable Models

LP → DB → SAT → DB+KR+LP

Solving
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

\[ \text{ASP} = \text{DB} + \text{LP} + \text{KR} + \text{SAT} \]
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

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

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

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 combinatorialics.
2. Boolean constraint technology is rather sensitive to search parameters.

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

Outline

1. Introduction
2. Modeling
3. Solving
   - Conflict-driven search
   - Solver configurations
   - Parallel solving
   - Automatic solver engineering
   - Domain-specific heuristics
4. Optimizing
5. Reacting
6. Summary
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
```prolog
{ 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.
```
Advanced encoding
queensA.lp

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

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

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).
### Grounding size via `wc --lines`

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

1. Introduction
2. Modeling
3. Solving
   - Conflict-driven search
   - Solver configurations
   - Parallel solving
   - Automatic solver engineering
   - Domain-specific heuristics
4. Optimizing
5. Reacting
6. Summary
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 \textit{clasp}
Multi-threaded architecture of *clasp*

- **Preprocessing**
  - Program Builder
  - Preprocessor

- **Coordination**
  - SharedContext
    - Propositional Variables
    - Atoms → Bodies
    - Static Nogoods
    - Implication Graph

- **Solver 1...n**
  - Conflict Resolution
  - Decision Heuristic
  - Assignment Atoms/Bodies

- **Recorded Nogoods**

- **ParallelContext**
  - Threads: \( S_1, S_2, \ldots, S_n \)
  - Counter: \( T, W, \ldots, S \)
  - Queue: \( P_1, P_2, \ldots, P_n \)

- **Nogood Distributor**

- **Logic Program**
  - Preprocessing Program Builder
  - Preprocessor
Multi-threaded architecture of clasp

Preprocessing
- Program Builder
- Preprocessor

Solving
- Program

Conflict-driven search
- Decision Heuristic
- Conflict Resolution
- Assignment
- Record Nogoods
- Propagation
- Unit Propagation
- Post Propagation

Coordination
- Shared Context
  - Propositional Variables
  - Atoms
  - Bodies
  - Static Nogoods
  - Implication Graph

- Nogood Distributor

- Enumerator
  - Threads
  - Counter
  - Queue

- Parallel Context
  - Shared Nogoods

Solver 1...n
- Decision Heuristic
  - Assignment
  - Recorded Nogoods

Logic Program

Torsten Schaub (KRR@UP) Experiencing ASP at Work
Multi-threaded architecture of clasp
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
Fact

Boolean constraint technology is rather sensitive to search parameters

Challenge

Robust ASP solving technology — *Taming the oracle!*
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
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

Like the pictures...?

👉 Check out Arne König’s talk on Tuesday at 16:00+ during TechComm 3
Outline

1. Introduction
2. Modeling
3. Solving
   - Conflict-driven search
   - Solver configurations
   - Parallel solving
   - Automatic solver engineering
   - Domain-specific heuristics
4. Optimizing
5. Reacting
6. Summary
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

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

(times in seconds, cut-off at 60 seconds)
## Comparing configurations
### on queensA.lp

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

(times in seconds, cut-off at 60 seconds)
Comparing configurations on queensA.lp

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

(times in seconds, cut-off at 60 seconds)
Comparing configurations
on queensA.lp

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

(times in seconds, cut-off at 60 seconds)
Comparing configurations on queensA.lp

<table>
<thead>
<tr>
<th>n</th>
<th>--frumpy</th>
<th>--jumpy</th>
<th>--handy</th>
<th>--crafty</th>
<th>--trendy</th>
<th>--chatty</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.063</td>
<td>0.023</td>
<td>3.416</td>
<td>0.030</td>
<td>1.805</td>
<td>0.061</td>
</tr>
<tr>
<td>100</td>
<td>0.099</td>
<td>0.099</td>
<td>7.891</td>
<td>0.136</td>
<td>7.321</td>
<td>0.121</td>
</tr>
<tr>
<td>150</td>
<td>0.212</td>
<td>0.212</td>
<td>14.522</td>
<td>0.271</td>
<td>19.883</td>
<td>0.347</td>
</tr>
<tr>
<td>200</td>
<td>0.415</td>
<td>0.415</td>
<td>15.026</td>
<td>0.667</td>
<td>32.476</td>
<td>0.753</td>
</tr>
<tr>
<td>500</td>
<td>3.199</td>
<td>3.199</td>
<td>60.000</td>
<td>7.471</td>
<td>60.000</td>
<td>6.104</td>
</tr>
</tbody>
</table>

(times in seconds, cut-off at 60 seconds)
Comparing configurations
on queensA.lp

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

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

1. Introduction
2. Modeling
3. Solving
   - Conflict-driven search
   - Solver configurations
   - Parallel solving
   - Automatic solver engineering
   - Domain-specific heuristics
4. Optimizing
5. Reacting
6. Summary
clasp’s default portfolio for parallel solving via clasp --print-portfolio

- 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

- 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

---

Torsten Schaub (KRR@UP)
clasp's default portfolio for parallel solving

via clasp --print-portfolio

- 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

1. Introduction
2. Modeling
3. Solving
   - Conflict-driven search
   - Solver configurations
   - Parallel solving
   - Automatic solver engineering
   - Domain-specific heuristics
4. Optimizing
5. Reacting
6. Summary
Correlation of *clasp* configurations
Algorithm engineering

- Algorithm Configuration: piclasp
- Algorithm Schedules: aspeed
- Algorithm Selection: claspfolio
Algorithm engineering

Algorithm Configuration: \textit{piclasp}

Algorithm Schedules: \textit{aspeed}

Algorithm Selection: \textit{claspfolio}
Task

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

Approach

Use an algorithm configurator (e.g., SMAC or ParamILS) for finding a well-performing configuration.
Task

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

Approach

Use an algorithm configurator (e.g., 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)
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
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

![Diagram showing a schedule with time on the x-axis and various solver strategies on the y-axis, labeled as default, clasp-vsids, clasp-prepro, and clasp-luby.]
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
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
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
Solving Automatic solver engineering

**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
Solving Automatic solver engineering

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
Feature space in practice
claspfolio’s architecture

Instances → Compute Features → Train Models → Assess Performance → Solvers
claspfolio’s architecture

- Instances
- Compute Features
- Train Models
- Solvers
- Assess Performance
- Learning

Solving Automatic solver engineering

Torsten Schaub (KRR@UP) Experiencing ASP at Work
claspfolio’s architecture

- (New) Instance
  - Compute Features
- Instances
  - Compute Features
  - Train Models
- Solvers
  - Assess Performance
  - Learning
- Score Solvers
  - Run best scored Solver
claspfolio’s architecture

- Instances
- Compute Features
- Train Models
- Score Solvers
- Run best scored Solver
- (New) Instance
- Solvers
- Assess Performance
- Learning
- Producing

Solving - Automatic solver engineering
Outline

1. Introduction
2. Modeling
3. Solving
   - Conflict-driven search
   - Solver configurations
   - Parallel solving
   - Automatic solver engineering
   - **Domain-specific heuristics**
4. Optimizing
5. Reacting
6. Summary
**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
  ```prolog
  _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
  ```bash
  $ 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
  
  ```prolog
  _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

  ```bash
  $ 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
    
    ```prolog
    _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
    
    ```bash
    $ 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

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

### Algorithmic scheme

1. \[ h(a) := \alpha \times h(a) + \beta(a) \] for each \( a \in A \)
2. \[ U := A \setminus (A^T \cup A^F) \]
3. \[ C := \arg \max_{a \in U} h(a) \]
4. \[ a := \tau(C) \]
5. \[ A := A \cup \{a \mapsto s(a)\} \]
Inside \textit{decide}

- **Heuristic functions**

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

- **Algorithmic scheme**

  1. \[ h(a) := \alpha \times h(a) + \beta(a) \]
  2. \[ U := \mathcal{A} \setminus (\mathcal{A}^T \cup \mathcal{A}^F) \]
  3. \[ C := \text{argmax}_{a \in U} h(a) \]
  4. \[ a := \tau(C) \]
  5. \[ A := A \cup \{a \mapsto s(a)\} \] for each \( a \in \mathcal{A} \)
Solving Domain-specific heuristics

Inside `decide`

- **Heuristic functions**

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

- **Algorithmic scheme**

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

  for each \( a \in \mathcal{A} \)
Heuristic language elements

- **Heuristic predicate**  \_heuristic

- **Heuristic modifiers**  \( \text{init} \) for initializing the heuristic value of \( a \) with \( v \)
  \( \text{factor} \) for amplifying the heuristic value of \( a \) by factor \( v \)
  \( \text{level} \) for ranking all atoms; the rank of \( a \) is \( v \)
  \( \text{sign} \) for attributing the sign of \( v \) as truth value to \( a \)

- **Heuristic atoms**

  \_heuristic(occurs(move),factor,5)
Heuristic language elements

- Heuristic predicate \texttt{heuristic}

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

- Heuristic atoms

\texttt{heuristic(occurs(move),factor,5)}
Solving Domain-specific heuristics

Heuristic language elements

- **Heuristic predicate**  \_heuristic

- **Heuristic modifiers**  \( (\text{atom, } a, \text{ and integer, } v) \)
  - `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**

\[
\text{\_heuristic(occurs(move),factor,5)}
\]
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 nholds(F,T), time(T).
holds(F,T) :- occurs(A,T), add(A,F).
nholds(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).
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).

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

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).

\[ :- \ \text{query}(F), \ \text{not} \ \text{holds}(F,t). \]

_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).
\quad :- 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).

\quad :- 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).

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

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>base configuration</th>
<th>_heuristic</th>
<th>base c. (SAT)</th>
<th>_heur. (SAT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks'00</td>
<td>134.4s (180/61)</td>
<td>9.2s (239/3)</td>
<td>163.2s (59)</td>
<td>2.6s (0)</td>
</tr>
<tr>
<td>Elevator'00</td>
<td>3.1s (279/0)</td>
<td>0.0s (279/0)</td>
<td>3.4s (0)</td>
<td>0.0s (0)</td>
</tr>
<tr>
<td>Freecell'00</td>
<td>288.7s (147/115)</td>
<td>184.2s (194/74)</td>
<td>226.4s (47)</td>
<td>52.0s (0)</td>
</tr>
<tr>
<td>Logistics'00</td>
<td>145.8s (148/61)</td>
<td>115.3s (168/52)</td>
<td>113.9s (23)</td>
<td>15.5s (3)</td>
</tr>
<tr>
<td>Depots'02</td>
<td>400.3s (51/184)</td>
<td>297.4s (115/135)</td>
<td>389.0s (64)</td>
<td>61.6s (0)</td>
</tr>
<tr>
<td>Driverlog'02</td>
<td>308.3s (108/143)</td>
<td>189.6s (169/92)</td>
<td>245.8s (61)</td>
<td>6.1s (0)</td>
</tr>
<tr>
<td>Rovers'02</td>
<td>245.8s (138/112)</td>
<td>165.7s (179/79)</td>
<td>162.9s (41)</td>
<td>5.7s (0)</td>
</tr>
<tr>
<td>Satellite'02</td>
<td>398.4s (73/186)</td>
<td>229.9s (155/106)</td>
<td>364.6s (82)</td>
<td>30.8s (0)</td>
</tr>
<tr>
<td>Zenotravel'02</td>
<td>350.7s (101/169)</td>
<td>239.0s (154/116)</td>
<td>224.5s (53)</td>
<td>6.3s (0)</td>
</tr>
<tr>
<td>Total</td>
<td>252.8s (1225/1031)</td>
<td>158.9s (1652/657)</td>
<td>187.2s (430)</td>
<td>17.1s (3)</td>
</tr>
</tbody>
</table>
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 \textit{clasp}
- Hierarchical Branch-and-Bound optimization in \textit{clasp}
- Unsatisfiability-based optimization in \textit{unclasp}
- Incremental optimization in \textit{iclingo}

- Saturation-based optimization in \textit{metasp} (via \textit{claspD})
- Heuristic-driven optimization in \textit{hclasp}
Alternative ways of optimization

- Branch-and-Bound optimization in \textit{clasp}
- Hierarchical Branch-and-Bound optimization in \textit{clasp}
- Unsatisfiability-based optimization in \textit{unclasp}
- Incremental optimization in \textit{iclingo}
- Saturation-based optimization in \textit{metasp} (via \textit{claspD})
- Heuristic-driven optimization in \textit{hclasp}
Alternative ways of optimization

- Branch-and-Bound optimization in \textit{clasp}
  - \textbf{SAT \ldots SAT UNSAT}
- Hierarchical Branch-and-Bound optimization in \textit{clasp}
- Unsatisfiability-based optimization in \textit{unclasp}
- Incremental optimization in \textit{iclingo}

- Saturation-based optimization in \textit{metasp} (via \textit{claspD})
- Heuristic-driven optimization in \textit{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 o UNSAT ...) SAT o UNSAT
- Heuristic-driven optimization in *hclasp*
  - SAT
Outline

1. Introduction
2. Modeling
3. Solving
   - Conflict-driven search
   - Solver configurations
   - Parallel solving
   - Automatic solver engineering
   - Domain-specific heuristics
4. Optimizing
5. Reacting
6. Summary
Challenge four (or: one+one+two)

Fact

Intelligence is build around us and in our pockets

Challenge

Incremental and reactive ASP solving technology
Reacting

Challenge four (or: one+one+two)

Fact

Intelligence is build around us and in our pockets

Challenge

Incremental and reactive ASP solving technology
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
Reacting

Going online

- Planning and reasoning about action with *iclino*
- 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)
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
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
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
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
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 is a composition of people


Dankeschön! Et merci!
Potassco is a composition of people


Dankeschön! Et merci!