Answer Set Solving in Practice

Martin Gebser and Torsten Schaub
University of Potsdam
torsten@cs.uni-potsdam.de

Rough Roadmap

1. Introduction
2. Language
3. Modeling
4. Grounding
5. Foundations
6. Solving
7. Systems
8. Applications

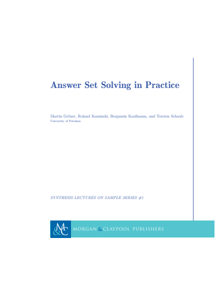
Potassco

Resources

- Course material
  - http://www.cs.uni-potsdam.de/wv/lehre
  - http://moodle.cs.uni-potsdam.de
  - http://potassco.sourceforge.net/teaching.html
- Systems
  - clasp http://potassco.sourceforge.net
  - dlv http://www.dlvsystem.com
  - smodels http://www.tcs.hut.fi/Software/smodels
  - gringo http://potassco.sourceforge.net
  - lparse http://www.tcs.hut.fi/Software/smodels
  - clingo http://potassco.sourceforge.net
  - iclingo http://potassco.sourceforge.net
  - oclingo http://potassco.sourceforge.net
  - asparagus http://asparagus.cs.uni-potsdam.de

The Potassco Book

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



Resources

- http://potassco.sourceforge.net/book.html
- http://potassco.sourceforge.net/teaching.html

Potassco

Literature

Books [4], [29], [53]

Surveys [50], [2], [39], [21], [11]

Articles [41], [42], [6], [61], [54], [49], [40], etc.

Incremental Grounding and Solving: Overview

Outline

Motivation

- Many real-world applications, having exponential state spaces, like
  - bio-informatics,
  - planning,
  - model checking,
  - etc

  have associated PSPACE-decision problems

- Example
  - The plan existence problem of deterministic planning is PSPACE-complete
    the problem of whether there is a plan having a length bounded by a given polynomial is in NP

Potassco

Motivation

- Many real-world applications, having exponential state spaces, like
  - bio-informatics,
  - planning,
  - model checking,
  - etc

  have associated PSPACE-decision problems

- Example
  - The plan existence problem of deterministic planning is PSPACE-complete
  - But the problem of whether there is a plan having a length bounded by a given polynomial is in NP

Motivation

- Many real-world applications, having exponential state spaces, like
  - bio-informatics,
  - planning,
  - model checking,
  - etc

  have associated PSPACE-decision problems

- Example
  - The plan existence problem of deterministic planning is PSPACE-complete
  - But the problem of whether there is a plan having a length bounded by a given polynomial is in NP

Motivation

- Iterative deepening search

  Consider one problem instance after another by gradually increasing the bound on the solution size

  - Problem This approach
    - is prone to redundancies (in grounding and solving), and
    - cannot harness modern look-back techniques in conflict-driven learning and heuristics

- Incremental approach

  - Idea Avoid redundancy by gradually processing the extensions to a problem rather than repeatedly re-processing the entire extended problem
  - ASP An incremental approach to both grounding and solving is needed

Potassco

Motivation

- Iterative deepening search
  Consider one problem instance after another by gradually increasing the bound on the solution size
  - Problem This approach
    - is prone to redundancies (in grounding and solving), and
    - cannot harness modern look-back techniques in conflict-driven learning and heuristics

- Incremental approach
  - Idea Avoid redundancy by gradually processing the extensions to a problem rather than repeatedly re-processing the entire extended problem
  - ASP An incremental approach to both grounding and solving is needed

Potassco

Motivation

- Iterative deepening search

  Consider one problem instance after another by gradually increasing the bound on the solution size

  - Problem This approach
    - is prone to redundancies (in grounding and solving), and
    - cannot harness modern look-back techniques in conflict-driven learning and heuristics

- Incremental approach
  - Idea Avoid redundancy by gradually processing the extensions to a problem rather than repeatedly re-processing the entire extended problem
  - ASP An incremental approach to both grounding and solving is needed

Incremental program

- An incremental program is a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over the natural numbers
  - The base program B is meant to describe static knowledge, independent of parameter k
  - The role of P is to capture knowledge accumulating with increasing k
  - The rules in Q are specific for each value of k
- We sometimes denote P and Q by $P[k]$ and $Q[k]$
- Intuitively, we want to decide whether the program

$$R[k/i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$$

has a stable model for some (minimum) integer $i \geq 1$

We write $R[i]$ rather than $R[k/i]$ whenever clear from the context

Potassco

Incremental program

- An incremental program is a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over the natural numbers
  - The base program B is meant to describe static knowledge, independent of parameter k
  - The role of P is to capture knowledge accumulating with increasing k
  - The rules in Q are specific for each value of k
- We sometimes denote P and Q by $P[k]$ and $Q[k]$
- Intuitively, we want to decide whether the program

$$R[k/i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$$

has a stable model for some (minimum) integer $i \geq 1$

We write $R[i]$ rather than $R[k/i]$ whenever clear from the context

Incremental program

- An incremental program is a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over the natural numbers
  - The base program B is meant to describe static knowledge, independent of parameter k
  - The role of P is to capture knowledge accumulating with increasing k
  - The rules in Q are specific for each value of k
- We sometimes denote P and Q by $P[k]$ and $Q[k]$
- Intuitively, we want to decide whether the program

$$R[k/i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$$

has a stable model for some (minimum) integer $i \geq 1$

We write $R[i]$ rather than $R[k/i]$ whenever clear from the context

Potassco

Incremental program

- An incremental program is a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over the natural numbers
  - The base program B is meant to describe static knowledge, independent of parameter k
  - The role of P is to capture knowledge accumulating with increasing k
  - The rules in Q are specific for each value of k
- We sometimes denote P and Q by $P[k]$ and $Q[k]$
- Intuitively, we want to decide whether the program

$$R[k/i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$$

has a stable model for some (minimum) integer $i \geq 1$

We write $R[i]$ rather than $R[k/i]$ whenever clear from the context

Potassco

Incremental program

- An incremental program is a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over the natural numbers
  - The base program B is meant to describe static knowledge, independent of parameter k
  - The role of P is to capture knowledge accumulating with increasing k
  - The rules in Q are specific for each value of k
- We sometimes denote P and Q by $P[k]$ and $Q[k]$
- Intuitively, we want to decide whether the program

$$R[k/i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$$

has a stable model for some (minimum) integer $i \geq 1$

We write $R[i]$ rather than $R[k/i]$ whenever clear from the context

Incremental program

- An incremental program is a triple (B, P, Q) of logic programs, among which P and Q contain a (single) parameter k ranging over the natural numbers
  - The base program B is meant to describe static knowledge, independent of parameter k
  - The role of P is to capture knowledge accumulating with increasing k
  - The rules in Q are specific for each value of k
- We sometimes denote P and Q by $P[k]$ and $Q[k]$
- Intuitively, we want to decide whether the program

$$R[k/i] = B \cup \bigcup_{1 \leq j \leq i} P[k/j] \cup Q[k/i]$$

has a stable model for some (minimum) integer $i \geq 1$

We write $R[i]$ rather than $R[k/i]$ whenever clear from the context

Grounding and Solving, iteratively

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$ ✗
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$ ✗
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ ✗

i etc. until a stable model is obtained ✔

Potassco

Grounding and Solving, iteratively

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$ ✘
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$ ✘
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and
   Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ ✘

i etc. until a stable model is obtained ✔

Potassco

Grounding and Solving, iteratively

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$ ✘
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$ ✘
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ ✘

i. etc. until a stable model is obtained ✔

Potassco

Grounding and Solving, iteratively

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$ ✘
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$ ✘
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ ✘

i etc. until a stable model is obtained ✔

Potassco

Grounding and Solving, iteratively

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$ ✘
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$ ✘
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and
   Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ ✘

i etc. until a stable model is obtained ✔

Potassco

Grounding and Solving, iteratively

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$ ✘
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$ ✘
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and
   Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ ✘

i etc. until a stable model is obtained ✔

Potassco

Grounding and Solving, iteratively

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$ ✘
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$ ✘
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and
   Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ ✘

i etc. until a stable model is obtained ✔

Potassco

Grounding and Solving, iteratively

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground $B \cup P[1] \cup Q[1]$ and Solve $B \cup P[1] \cup Q[1]$ ✘
2. Ground $B \cup P[1] \cup P[2] \cup Q[2]$ and Solve $B \cup P[1] \cup P[2] \cup Q[2]$ ✘
3. Ground $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ and
   Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$ ✘

i etc. until a stable model is obtained ✔

Grounding and Solving, incrementally

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$ ✗
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$ ✗
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, ✗
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc, until a stable model is obtained ✔

Potassco

Grounding and Solving, incrementally

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$ ✗
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$ ✗
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, ✗
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i etc, until a stable model is obtained ✔

Potassco

Grounding and Solving, incrementally

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$ ✗
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$ ✗
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, ✗
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i etc, until a stable model is obtained ✔

Potassco

Grounding and Solving, incrementally

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $B \cup P[1] \cup Q[1]$ ✘
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $B \cup P[1] \cup P[2] \cup Q[2]$ ✘
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $B \cup P[1] \cup P[2] \cup P[3] \cup Q[3]$, ✘
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i etc, until a stable model is obtained ✔

Potassco

Grounding and Solving, incrementally

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$ ✗
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$ ✗
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, ✗
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i etc, until a stable model is obtained ✓

Potassco

Grounding and Solving, incrementally

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$ ✘
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$ ✘
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, ✘
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i etc, until a stable model is obtained ✔

Potassco

Grounding and Solving, incrementally

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$ ✘
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$ ✘
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, ✘
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc, until a stable model is obtained ✔

Potassco

Grounding and Solving, incrementally

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$ ✘
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$ ✘
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, ✘
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc, until a stable model is obtained ✔

Potassco

Grounding and Solving, incrementally

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$ ✘
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$ ✘
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, ✘
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc, until a stable model is obtained ✔

Potassco

Grounding and Solving, incrementally

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$ ✘
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$ ✘
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, ✘
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i etc, until a stable model is obtained ✔

Potassco

Grounding and Solving, incrementally

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$ ✘
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$ ✘
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, ✘
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i. etc, until a stable model is obtained ✔

Potassco

Grounding and Solving, incrementally

- Input An incremental program $R[k] = (B, P[k], Q[k])$
- Output A non-empty set of stable models of $R[k/i]$

1. Ground B and Keep B
2. Ground $P[1] \cup Q[1]$, Solve $\underline{B} \cup P[1] \cup Q[1]$ ✘
   Keep $B \cup P[1]$, and Discard $Q[1]$
3. Ground $P[2] \cup Q[2]$, Solve $\underline{B \cup P[1]} \cup P[2] \cup Q[2]$ ✘
   Keep $B \cup P[1] \cup P[2]$, and Discard $Q[2]$
4. Ground $P[3] \cup Q[3]$, Solve $\underline{B \cup P[1] \cup P[2]} \cup P[3] \cup Q[3]$, ✘
   Keep $B \cup P[1] \cup P[2] \cup P[3]$, and Discard $Q[3]$

i etc, until a stable model is obtained ✔

Potassco

Outline

Potassco

Module

- A module $\mathbb{P}$ is a triple (P, I, O) consisting of
  - a (ground) program P over $grd(\mathcal{A})$ and
  - sets $I, O \subseteq grd(\mathcal{A})$ such that
    - $I \cap O = \emptyset$,
    - $atom(P) \subseteq I \cup O$, and
    - $head(P) \subseteq O$
- The elements of I and O are called input and output atoms
  - denoted by $I(\mathbb{P})$ and $O(\mathbb{P})$
- Similarly, we refer to (ground) program P by $P(\mathbb{P})$

Potassco

Module

- A module $\mathbb{P}$ is a triple (P, I, O) consisting of
  - a (ground) program P over $grd(\mathcal{A})$ and
  - sets $I, O \subseteq grd(\mathcal{A})$ such that
    - $I \cap O = \emptyset$,
    - $atom(P) \subseteq I \cup O$, and
    - $head(P) \subseteq O$
- The elements of I and O are called input and output atoms
  - denoted by $I(\mathbb{P})$ and $O(\mathbb{P})$
- Similarly, we refer to (ground) program P by $P(\mathbb{P})$

Potassco

Ground Instantiation

- The ground instantiation of a program P is defined as

$$grd(P) = \{r\theta \mid r \in P, \theta : var(r) \rightarrow \mathcal{T}, var(r\theta) = \emptyset\}$$

  where $\mathcal{T}$ is a set of variable-free terms
- Analogously, $grd(\mathcal{A}) = \{a \in \mathcal{A} \mid var(a) = \emptyset\}$ is the set of ground atoms

- Note that in an incremental setting $\mathcal{T}$ includes the natural numbers !

Ground Instantiation

- The ground instantiation of a program P is defined as

$$grd(P) = \{r\theta \mid r \in P, \theta : var(r) \to \mathcal{T}, var(r\theta) = \emptyset\}$$

  where $\mathcal{T}$ is a set of variable-free terms
- Analogously, $grd(\mathcal{A}) = \{a \in \mathcal{A} \mid var(a) = \emptyset\}$ is the set of ground atoms

- Note that in an incremental setting $\mathcal{T}$ includes the natural numbers !

Formal Setting

- For a program P over $grd(\mathcal{A})$ and a set $X \subseteq grd(\mathcal{A})$, define

$$P|_X = \{head(r) \leftarrow body(r)^+ \cup L \mid r \in P, \\ body(r)^+ \subseteq X,\ L = \{\sim c \mid c \in body(r)^- \cap X\}\}$$

- Note $P|_X$ projects the bodies of rules in P to the atoms of X

- For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, define $\mathbb{P}(I)$ as the module

$$(\, grd(P)|_Y, I, head(grd(P)|_X)\,)$$

where $X = I \cup head(grd(P))$ and $Y = I \cup head(grd(P)|_X)$

- For $\mathbb{P}(I) = (P', I, O)$, we have $O \subseteq grd(\mathcal{A})$ and $atom(P') \subseteq I \cup O$

Potassco

Formal Setting

- For a program P over $grd(\mathcal{A})$ and a set $X \subseteq grd(\mathcal{A})$, define

$$P|_X = \{head(r) \leftarrow body(r)^+ \cup L \mid r \in P,\; body(r)^+ \subseteq X,\; L = \{\sim c \mid c \in body(r)^- \cap X\}\}$$

- Note $P|_X$ projects the bodies of rules in P to the atoms of X

- For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, define $\mathbb{P}(I)$ as the module

$$(\,grd(P)|_Y, I, head(grd(P)|_X)\,)$$

  where $X = I \cup head(grd(P))$ and $Y = I \cup head(grd(P)|_X)$

- For $\mathbb{P}(I) = (P', I, O)$, we have $O \subseteq grd(\mathcal{A})$ and $atom(P') \subseteq I \cup O$

Formal Setting

- For a program P over $grd(\mathcal{A})$ and a set $X \subseteq grd(\mathcal{A})$, define

$$P|_X = \{head(r) \leftarrow body(r)^+ \cup L \mid r \in P,\ body(r)^+ \subseteq X,\ L = \{\sim c \mid c \in body(r)^- \cap X\}\}$$

- Note $P|_X$ projects the bodies of rules in P to the atoms of X

- For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, define $\mathbb{P}(I)$ as the module

$$(\,grd(P)|_Y, I, head(grd(P)|_X)\,)$$

  where $X = I \cup head(grd(P))$ and $Y = I \cup head(grd(P)|_X)$

- For $\mathbb{P}(I) = (P', I, O)$, we have $O \subseteq grd(\mathcal{A})$ and $atom(P') \subseteq I \cup O$

Potassco

Formal Setting

- For a program P over $grd(\mathcal{A})$ and a set $X \subseteq grd(\mathcal{A})$, define

$$P|_X = \{head(r) \leftarrow body(r)^+ \cup L \mid r \in P, \quad body(r)^+ \subseteq X,\ L = \{\sim c \mid c \in body(r)^- \cap X\}\}$$

- Note $P|_X$ projects the bodies of rules in P to the atoms of X

- For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, define $\mathbb{P}(I)$ as the module

$$(\, grd(P)|_Y, I, head(grd(P)|_X)\,)$$

where $X = I \cup head(grd(P))$ and $Y = I \cup head(grd(P)|_X)$

- For $\mathbb{P}(I) = (P', I, O)$, we have $O \subseteq grd(\mathcal{A})$ and $atom(P') \subseteq I \cup O$

A Simple Example

- Consider

$$P[k] = \{p(k) \leftarrow p(Y), {\sim}p(2) \qquad p(k) \leftarrow p(2)\}$$

and note that $grd(P[1])$ is infinite !

- For $P[1]$ and $I = \{p(0)\}$, we get the module

$$(\, grd(P[1])|_{\{p(0),p(1)\}}\, ,\, \{p(0)\}\, ,\, \{p(1)\}\,)$$

where $grd(P[1])|_{\{p(0),p(1)\}} = \{p(1) \leftarrow p(0)\ ,\ p(1) \leftarrow p(1)\}$

Potassco

A Simple Example

- Consider

$$P[k] = \{p(k) \leftarrow p(Y), \sim p(2) \qquad p(k) \leftarrow p(2)\}$$

  and note that $grd(P[1])$ is infinite !

- For $P[1]$ and $I = \{p(0)\}$, we get the module

$$(\ grd(P[1])|_{\{p(0),p(1)\}}\ ,\ \{p(0)\}\ ,\ \{p(1)\}\ )$$

  where $grd(P[1])|_{\{p(0),p(1)\}} = \{p(1) \leftarrow p(0)\ ,\ p(1) \leftarrow p(1)\}$

Modular Incremental Program

- Define the join, $\mathbb{P} \sqcup \mathbb{Q}$, of two modules $\mathbb{P}$ and $\mathbb{Q}$ as the module

$$(\; P(\mathbb{P}) \cup P(\mathbb{Q}) \, , \; I(\mathbb{P}) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P})) \, , \; O(\mathbb{P}) \cup O(\mathbb{Q}) \;) \; ,$$

  provided that $(I(\mathbb{P}) \cup O(\mathbb{P})) \cap O(\mathbb{Q}) = \emptyset$

- Note
  - Recursion between two modules to be joined is disallowed
  - Recursion is allowed within each module

- An incremental program $(B, P[k], Q[k])$ is modular, if the modules

$$\mathbb{P}_i = \mathbb{P}_{i-1} \sqcup \mathbb{P}[i](O(\mathbb{P}_{i-1})) \quad \text{and} \quad \mathbb{Q}_i = \mathbb{P}_i \sqcup \mathbb{Q}[i](O(\mathbb{P}_i))$$

  are defined for $i \geq 1$, where $\mathbb{P}_0 = \mathbb{B}(\emptyset)$

Potassco

Modular Incremental Program

- Define the join, $\mathbb{P} \sqcup \mathbb{Q}$, of two modules $\mathbb{P}$ and $\mathbb{Q}$ as the module

$$(\; P(\mathbb{P}) \cup P(\mathbb{Q}) \, , \; I(\mathbb{P}) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P})) \, , \; O(\mathbb{P}) \cup O(\mathbb{Q}) \;) \; ,$$

provided that $(I(\mathbb{P}) \cup O(\mathbb{P})) \cap O(\mathbb{Q}) = \emptyset$

- Note
  - Recursion between two modules to be joined is disallowed
  - Recursion is allowed within each module
- An incremental program $(B, P[k], Q[k])$ is modular, if the modules

$$\mathbb{P}_i = \mathbb{P}_{i-1} \sqcup \mathbb{P}[i](O(\mathbb{P}_{i-1})) \quad \text{and} \quad \mathbb{Q}_i = \mathbb{P}_i \sqcup \mathbb{Q}[i](O(\mathbb{P}_i))$$

are defined for $i \geq 1$, where $\mathbb{P}_0 = \mathbb{B}(\emptyset)$

Potassco

Modular Incremental Program

- Define the join, $\mathbb{P} \sqcup \mathbb{Q}$, of two modules $\mathbb{P}$ and $\mathbb{Q}$ as the module

$$(\; P(\mathbb{P}) \cup P(\mathbb{Q}) \, , \; I(\mathbb{P}) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P})) \, , \; O(\mathbb{P}) \cup O(\mathbb{Q}) \;) \; ,$$

  provided that $(I(\mathbb{P}) \cup O(\mathbb{P})) \cap O(\mathbb{Q}) = \emptyset$

- Note
  - Recursion between two modules to be joined is disallowed
  - Recursion is allowed within each module

- An incremental program $(B, P[k], Q[k])$ is modular, if the modules

$$\mathbb{P}_i = \mathbb{P}_{i-1} \sqcup \mathbb{P}[i](O(\mathbb{P}_{i-1})) \quad \text{and} \quad \mathbb{Q}_i = \mathbb{P}_i \sqcup \mathbb{Q}[i](O(\mathbb{P}_i))$$

  are defined for $i \geq 1$, where $\mathbb{P}_0 = \mathbb{B}(\emptyset)$

Modular Incremental Program

- Define the join, $\mathbb{P} \sqcup \mathbb{Q}$, of two modules $\mathbb{P}$ and $\mathbb{Q}$ as the module

$$(P(\mathbb{P}) \cup P(\mathbb{Q}) , \; I(\mathbb{P}) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P})) , \; O(\mathbb{P}) \cup O(\mathbb{Q})) ,$$

  provided that $(I(\mathbb{P}) \cup O(\mathbb{P})) \cap O(\mathbb{Q}) = \emptyset$

- Note
  - Recursion between two modules to be joined is disallowed
  - Recursion is allowed within each module
- An incremental program $(B, P[k], Q[k])$ is modular, if the modules

$$\mathbb{P}_i = \mathbb{P}_{i-1} \sqcup \mathbb{P}[i](O(\mathbb{P}_{i-1})) \quad \text{and} \quad \mathbb{Q}_i = \mathbb{P}_i \sqcup \mathbb{Q}[i](O(\mathbb{P}_i))$$

  are defined for $i \geq 1$, where $\mathbb{P}_0 = \mathbb{B}(\emptyset)$

Potassco

Modular Incremental Program

- Define the join, $\mathbb{P} \sqcup \mathbb{Q}$, of two modules $\mathbb{P}$ and $\mathbb{Q}$ as the module

$$(P(\mathbb{P}) \cup P(\mathbb{Q}), \; I(\mathbb{P}) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P})), \; O(\mathbb{P}) \cup O(\mathbb{Q})) ,$$

  provided that $(I(\mathbb{P}) \cup O(\mathbb{P})) \cap O(\mathbb{Q}) = \emptyset$

- Note
  - Recursion between two modules to be joined is disallowed
  - Recursion is allowed within each module
- An incremental program $(B, P[k], Q[k])$ is modular, if the modules

$$\mathbb{P}_i = \mathbb{P}_{i-1} \sqcup \mathbb{P}[i](O(\mathbb{P}_{i-1})) \quad \text{and} \quad \mathbb{Q}_i = \mathbb{P}_i \sqcup \mathbb{Q}[i](O(\mathbb{P}_i))$$

  are defined for $i \geq 1$, where $\mathbb{P}_0 = \mathbb{B}(\emptyset)$

Modular Incremental Program

- Define the join, $\mathbb{P} \sqcup \mathbb{Q}$, of two modules $\mathbb{P}$ and $\mathbb{Q}$ as the module

$$(\ P(\mathbb{P}) \cup P(\mathbb{Q}) \, , \ I(\mathbb{P}) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P})) \, , \ O(\mathbb{P}) \cup O(\mathbb{Q}) \ ) \ ,$$

  provided that $(I(\mathbb{P}) \cup O(\mathbb{P})) \cap O(\mathbb{Q}) = \emptyset$
- Note
  - Recursion between two modules to be joined is disallowed
  - Recursion is allowed within each module
- An incremental program $(B, P[k], Q[k])$ is modular, if the modules

$$\mathbb{P}_i = \mathbb{P}_{i-1} \sqcup \mathbb{P}[i](O(\mathbb{P}_{i-1})) \quad \text{and} \quad \mathbb{Q}_i = \mathbb{P}_i \sqcup \mathbb{Q}[i](O(\mathbb{P}_i))$$

  are defined for $i \geq 1$, where $\mathbb{P}_0 = \mathbb{B}(\emptyset)$

Modular Incremental Program

- Define the join, $\mathbb{P} \sqcup \mathbb{Q}$, of two modules $\mathbb{P}$ and $\mathbb{Q}$ as the module

$$(\ P(\mathbb{P}) \cup P(\mathbb{Q}) \, , \ I(\mathbb{P}) \cup (I(\mathbb{Q}) \setminus O(\mathbb{P})) \, , \ O(\mathbb{P}) \cup O(\mathbb{Q}) \ ) \ ,$$

  provided that $(I(\mathbb{P}) \cup O(\mathbb{P})) \cap O(\mathbb{Q}) = \emptyset$
- Note
  - Recursion between two modules to be joined is disallowed
  - Recursion is allowed within each module
- An incremental program $(B, P[k], Q[k])$ is modular, if the modules

$$\mathbb{P}_i = \mathbb{P}_{i-1} \sqcup \mathbb{P}[i](O(\mathbb{P}_{i-1})) \quad \text{and} \quad \mathbb{Q}_i = \mathbb{P}_i \sqcup \mathbb{Q}[i](O(\mathbb{P}_i))$$

  are defined for $i \geq 1$, where $\mathbb{P}_0 = \mathbb{B}(\emptyset)$

Potassco

A pragmatic approach

- An incremental program $(B, P[k], Q[k])$ is modular, if
  - atoms defined in B comprise dedicated predicates or 0 as argument,
  - atoms defined in $P[k]$ comprise k as argument, and
  - atoms defined in $Q[k]$ comprise dedicated predicates and k as argument

- The above conditions can be formalized as follows:
  - $atom(grd(B)) \cap \left(\bigcup_{1 \leq i} head(grd(P[i] \cup Q[i]))\right) = \emptyset$,
  - $\left(\bigcup_{1 \leq i} atom(grd(P[i]))\right) \cap \left(\bigcup_{1 \leq j} head(grd(Q[j]))\right) = \emptyset$,
  - $atom(grd(P[i])) \cap \left(\bigcup_{i < j} head(grd(P[j]))\right) = \emptyset$ for all $1 \leq i$, and
  - $atom(grd(Q[i])) \cap \left(\bigcup_{i < j} head(grd(Q[j]))\right) = \emptyset$ for all $1 \leq i$

Potassco

A pragmatic approach

- An incremental program $(B, P[k], Q[k])$ is modular, if
  - atoms defined in B comprise dedicated predicates or 0 as argument,
  - atoms defined in $P[k]$ comprise k as argument, and
  - atoms defined in $Q[k]$ comprise dedicated predicates and k as argument
- The above conditions can be formalized as follows:
  - $atom(grd(B)) \cap \left(\bigcup_{1\leq i} head(grd(P[i] \cup Q[i]))\right) = \emptyset$,
  - $\left(\bigcup_{1\leq i} atom(grd(P[i]))\right) \cap \left(\bigcup_{1\leq j} head(grd(Q[j]))\right) = \emptyset$,
  - $atom(grd(P[i])) \cap \left(\bigcup_{i<j} head(grd(P[j]))\right) = \emptyset$ for all $1 \leq i$, and
  - $atom(grd(Q[i])) \cap \left(\bigcup_{i<j} head(grd(Q[j]))\right) = \emptyset$ for all $1 \leq i$

Potassco

Outline

Incremental ASP Solving (made very easy)

- Grounding For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, an incremental grounder is a partial function

  $$\textsc{Ground} : (P, I) \mapsto (P', O) \ ,$$

  where P' is a program over $grd(\mathcal{A})$ and $O \subseteq grd(\mathcal{A})$
- Solving For programs R, R' over $grd(\mathcal{A})$ and a set L of literals over $grd(\mathcal{A})$, an incremental solver is a pair of total functions

  $$\textsc{Add} : R \mapsto R' \quad \text{and} \quad \textsc{Solve} : L \mapsto \chi,$$

  where χ is a subset of the power set of $grd(\mathcal{A})$

Incremental ASP Solving (made very easy)

- Grounding For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, an incremental grounder is a partial function

$$\textsc{Ground} : (P, I) \mapsto (P', O) \ ,$$

where P' is a program over $grd(\mathcal{A})$ and $O \subseteq grd(\mathcal{A})$

- Solving For programs R, R' over $grd(\mathcal{A})$ and a set L of literals over $grd(\mathcal{A})$, an incremental solver is a pair of total functions

$$\textsc{Add} : R \mapsto R' \quad \text{and} \quad \textsc{Solve} : L \mapsto \chi,$$

where χ is a subset of the power set of $grd(\mathcal{A})$

Potassco

Incremental ASP Solving (made very easy)

- Grounding For a program P over $\mathcal{A}$ and $I \subseteq grd(\mathcal{A})$, an incremental grounder is a partial function

$$\textsc{Ground} : (P, I) \mapsto (P', O) \ ,$$

where P' is a program over $grd(\mathcal{A})$ and $O \subseteq grd(\mathcal{A})$

- Solving For programs R, R' over $grd(\mathcal{A})$ and a set L of literals over $grd(\mathcal{A})$, an incremental solver is a pair of total functions

$$\textsc{Add} : R \mapsto R' \quad \text{and} \quad \textsc{Solve} : L \mapsto \chi,$$

where χ is a subset of the power set of $grd(\mathcal{A})$

Making rules volatile

- For a program Q over $grd(\mathcal{A})$ and a new atom $\alpha \notin grd(\mathcal{A})$, define

$$Q(\alpha) = \{head(r) \leftarrow body(r) \cup \{\alpha\} \mid r \in Q\}$$

- Deletion is provoked by adding the integrity constraint $\leftarrow \alpha$
- Note No modification to internal data structures upon deletion

Making rules volatile

- For a program Q over $grd(\mathcal{A})$ and a new atom $\alpha \notin grd(\mathcal{A})$, define

$$Q(\alpha) = \{head(r) \leftarrow body(r) \cup \{\alpha\} \mid r \in Q\}$$

- Deletion is provoked by adding the integrity constraint $\leftarrow \alpha$
- Note No modification to internal data structures upon deletion

Making rules volatile

- For a program Q over $grd(\mathcal{A})$ and a new atom $\alpha \notin grd(\mathcal{A})$, define

$$Q(\alpha) = \{head(r) \leftarrow body(r) \cup \{\alpha\} \mid r \in Q\}$$

- Deletion is provoked by adding the integrity constraint $\leftarrow \alpha$
- Note No modification to internal data structures upon deletion

Potassco

Algorithm 1: iSolve

Input : An incremental program $(B, P[k], Q[k])$
Output : A nonempty set of stable models
Internal : A grounder Grounder
Internal : A solver Solver

$i \leftarrow 0$
$(P_0, O) \leftarrow$ Grounder.$\textsc{Ground}(B, \emptyset)$
Solver.$\textsc{Add}(P_0)$
loop
  $i \leftarrow i + 1$
  $(P_i, O_i) \leftarrow$ Grounder.$\textsc{Ground}(P[i], O)$
  Solver.$\textsc{Add}(P_i)$
  $O \leftarrow O \cup O_i$
  $(Q_i, O_i') \leftarrow$ Grounder.$\textsc{Ground}(Q[i], O)$
  Solver.$\textsc{Add}(Q_i(\alpha_i) \cup \{\{\alpha_i\} \leftarrow\} \cup \{\leftarrow \alpha_{i-1}\})$
  $\chi \leftarrow$ Solver.$\textsc{Solve}(\{\alpha_i\})$
  if $\chi \neq \emptyset$ **then return** $\{X \setminus \{\alpha_i\} \mid X \in \chi\}$

[1] C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub.
The nomore++ approach to answer set solving.
In G. Sutcliffe and A. Voronkov, editors, *Proceedings of the Twelfth International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, volume 3835 of *Lecture Notes in Artificial Intelligence*, pages 95–109. Springer-Verlag, 2005.

[2] C. Anger, K. Konczak, T. Linke, and T. Schaub.
A glimpse of answer set programming.
Künstliche Intelligenz, 19(1):12–17, 2005.

[3] Y. Babovich and V. Lifschitz.
Computing answer sets using program completion.
Unpublished draft, 2003.

[4] C. Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2003.

[5] C. Baral, G. Brewka, and J. Schlipf, editors.
Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07), volume 4483 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2007.

[6] C. Baral and M. Gelfond.
Logic programming and knowledge representation.
Journal of Logic Programming, 12:1–80, 1994.

[7] S. Baselice, P. Bonatti, and M. Gelfond.
Towards an integration of answer set and constraint solving.
In M. Gabbrielli and G. Gupta, editors, *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 52–66. Springer-Verlag, 2005.

[8] A. Biere.
Adaptive restart strategies for conflict driven SAT solvers.

In H. Kleine Büning and X. Zhao, editors, *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer-Verlag, 2008.

[9] A. Biere.
PicoSAT essentials.
Journal on Satisfiability, Boolean Modeling and Computation, 4:75–97, 2008.

[10] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors.
Handbook of Satisfiability, volume 185 of *Frontiers in Artificial Intelligence and Applications*.
IOS Press, 2009.

[11] G. Brewka, T. Eiter, and M. Truszczyński.
Answer set programming at a glance.
Communications of the ACM, 54(12):92–103, 2011.

[12] K. Clark.
Negation as failure.

In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[13] M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors.
Handbook of Tableau Methods.
Kluwer Academic Publishers, 1999.

[14] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov.
Complexity and expressive power of logic programming.
In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity (CCC'97)*, pages 82–101. IEEE Computer Society Press, 1997.

[15] M. Davis, G. Logemann, and D. Loveland.
A machine program for theorem-proving.
Communications of the ACM, 5:394–397, 1962.

[16] M. Davis and H. Putnam.
A computing procedure for quantification theory.
Journal of the ACM, 7:201–215, 1960.

[17] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub.
Conflict-driven disjunctive answer set solving.
In G. Brewka and J. Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 422–432. AAAI Press, 2008.

[18] C. Drescher, M. Gebser, B. Kaufmann, and T. Schaub.
Heuristics in conflict resolution.
In M. Pagnucco and M. Thielscher, editors, *Proceedings of the Twelfth International Workshop on Nonmonotonic Reasoning (NMR'08)*, number UNSW-CSE-TR-0819 in School of Computer Science and Engineering, The University of New South Wales, Technical Report Series, pages 141–149, 2008.

[19] N. Eén and N. Sörensson.
An extensible SAT-solver.
In E. Giunchiglia and A. Tacchella, editors, *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability*

Testing (SAT'03), volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer-Verlag, 2004.

[20] T. Eiter and G. Gottlob.
On the computational cost of disjunctive logic programming: Propositional case.
Annals of Mathematics and Artificial Intelligence, 15(3-4):289–323, 1995.

[21] T. Eiter, G. Ianni, and T. Krennwallner.
Answer Set Programming: A Primer.
In S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M. Rousset, and R. Schmidt, editors, *Fifth International Reasoning Web Summer School (RW'09)*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer-Verlag, 2009.

[22] F. Fages.
Consistency of Clark's completion and the existence of stable models.
Journal of Methods of Logic in Computer Science, 1:51–60, 1994.

[23] P. Ferraris.

Answer sets for propositional theories.
In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*, pages 119–131. Springer-Verlag, 2005.

[24] P. Ferraris and V. Lifschitz.
Mathematical foundations of answer set programming.
In S. Artëmov, H. Barringer, A. d'Avila Garcez, L. Lamb, and J. Woods, editors, *We Will Show Them! Essays in Honour of Dov Gabbay*, volume 1, pages 615–664. College Publications, 2005.

[25] M. Fitting.
A Kripke-Kleene semantics for logic programs.
Journal of Logic Programming, 2(4):295–312, 1985.

[26] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`.

[27] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele.
Engineering an incremental ASP solver.
In M. Garcia de la Banda and E. Pontelli, editors, *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, pages 190–205. Springer-Verlag, 2008.

[28] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
On the implementation of weight constraint rules in conflict-driven ASP solvers.
In Hill and Warren [44], pages 250–264.

[29] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
Answer Set Solving in Practice.
Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

[30] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.

clasp: A conflict-driven answer set solver.
In Baral et al. [5], pages 260–265.

[31] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set enumeration.
In Baral et al. [5], pages 136–148.

[32] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set solving.
In Veloso [68], pages 386–392.

[33] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Advanced preprocessing for answer set solving.
In M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, pages 15–19. IOS Press, 2008.

[34] M. Gebser, B. Kaufmann, and T. Schaub.
The conflict-driven answer set solver clasp: Progress report.

In E. Erdem, F. Lin, and T. Schaub, editors, *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Artificial Intelligence*, pages 509–514. Springer-Verlag, 2009.

[35] M. Gebser, B. Kaufmann, and T. Schaub.
Solution enumeration for projected Boolean search problems.
In W. van Hoeve and J. Hooker, editors, *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2009.

[36] M. Gebser, M. Ostrowski, and T. Schaub.
Constraint answer set solving.
In Hill and Warren [44], pages 235–249.

[37] M. Gebser and T. Schaub.
Tableau calculi for answer set programming.

Potassco

In S. Etalle and M. Truszczyński, editors, *Proceedings of the Twenty-second International Conference on Logic Programming (ICLP'06)*, volume 4079 of *Lecture Notes in Computer Science*, pages 11–25. Springer-Verlag, 2006.

[38] M. Gebser and T. Schaub.
Generic tableaux for answer set programming.
In V. Dahl and I. Niemelä, editors, *Proceedings of the Twenty-third International Conference on Logic Programming (ICLP'07)*, volume 4670 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 2007.

[39] M. Gelfond.
Answer sets.
In V. Lifschitz, F. van Harmelen, and B. Porter, editors, *Handbook of Knowledge Representation*, chapter 7, pages 285–316. Elsevier Science, 2008.

[40] M. Gelfond and N. Leone.

Logic programming and knowledge representation — the A-Prolog perspective.
Artificial Intelligence, 138(1-2):3–38, 2002.

[41] M. Gelfond and V. Lifschitz.
The stable model semantics for logic programming.
In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.

[42] M. Gelfond and V. Lifschitz.
Logic programs with classical negation.
In D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming (ICLP'90)*, pages 579–597. MIT Press, 1990.

[43] E. Giunchiglia, Y. Lierler, and M. Maratea.
Answer set programming based on propositional satisfiability.
Journal of Automated Reasoning, 36(4):345–377, 2006.

[44] P. Hill and D. Warren, editors.
Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09), volume 5649 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[45] J. Huang.
The effect of restarts on the efficiency of clause learning.
In Veloso [68], pages 2318–2323.

[46] K. Konczak, T. Linke, and T. Schaub.
Graphs and colorings for answer set programming.
Theory and Practice of Logic Programming, 6(1-2):61–106, 2006.

[47] J. Lee.
A model-theoretic counterpart of loop formulas.
In L. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 503–508. Professional Book Center, 2005.

[48] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello.
The DLV system for knowledge representation and reasoning.
ACM Transactions on Computational Logic, 7(3):499–562, 2006.

[49] V. Lifschitz.
Answer set programming and plan generation.
Artificial Intelligence, 138(1-2):39–54, 2002.

[50] V. Lifschitz.
Introduction to answer set programming.
Unpublished draft, 2004.

[51] V. Lifschitz and A. Razborov.
Why are there so many loop formulas?
ACM Transactions on Computational Logic, 7(2):261–268, 2006.

[52] F. Lin and Y. Zhao.
ASSAT: computing answer sets of a logic program by SAT solvers.
Artificial Intelligence, 157(1-2):115–137, 2004.

[53] V. Marek and M. Truszczyński.
Nonmonotonic logic: context-dependent reasoning.
Artifical Intelligence. Springer-Verlag, 1993.

[54] V. Marek and M. Truszczyński.
Stable models and an alternative logic programming paradigm.
In K. Apt, V. Marek, M. Truszczyński, and D. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[55] J. Marques-Silva, I. Lynce, and S. Malik.
Conflict-driven clause learning SAT solvers.
In Biere et al. [10], chapter 4, pages 131–153.

[56] J. Marques-Silva and K. Sakallah.
GRASP: A search algorithm for propositional satisfiability.
IEEE Transactions on Computers, 48(5):506–521, 1999.

[57] V. Mellarkod and M. Gelfond.
Integrating answer set reasoning with constraint solving techniques.

Potassco

In J. Garrigue and M. Hermenegildo, editors, *Proceedings of the Ninth International Symposium on Functional and Logic Programming (FLOPS'08)*, volume 4989 of *Lecture Notes in Computer Science*, pages 15–31. Springer-Verlag, 2008.

[58] V. Mellarkod, M. Gelfond, and Y. Zhang.
Integrating answer set programming and constraint logic programming.
Annals of Mathematics and Artificial Intelligence, 53(1-4):251–287, 2008.

[59] D. Mitchell.
A SAT solver primer.
Bulletin of the European Association for Theoretical Computer Science, 85:112–133, 2005.

[60] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik.
Chaff: Engineering an efficient SAT solver.
In *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*, pages 530–535. ACM Press, 2001.

[61] I. Niemelä.
Logic programs with stable model semantics as a constraint programming paradigm.
Annals of Mathematics and Artificial Intelligence, 25(3-4):241–273, 1999.

[62] R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T).
Journal of the ACM, 53(6):937–977, 2006.

[63] K. Pipatsrisawat and A. Darwiche.
A lightweight component caching scheme for satisfiability solvers.
In J. Marques-Silva and K. Sakallah, editors, *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer-Verlag, 2007.

[64] L. Ryan.
Efficient algorithms for clause-learning SAT solvers.

Master's thesis, Simon Fraser University, 2004.

[65] P. Simons, I. Niemelä, and T. Soininen.
Extending and implementing the stable model semantics.
Artificial Intelligence, 138(1-2):181–234, 2002.

[66] T. Syrjänen.
Lparse 1.0 user's manual.

[67] A. Van Gelder, K. Ross, and J. Schlipf.
The well-founded semantics for general logic programs.
Journal of the ACM, 38(3):620–650, 1991.

[68] M. Veloso, editor.
Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07). AAAI/MIT Press, 2007.

[69] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik.
Efficient conflict driven learning in a Boolean satisfiability solver.
In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285. ACM Press, 2001.