Answer Set Solving in Practice

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

Potassco

Preferences and optimization: Overview

Outline

Potassco

Motivation

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

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

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

Motivation

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

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

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

Potassco

Motivation

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

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

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

Motivation

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

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

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

Outline

Approach

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

Approach

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

Approach

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

Approach

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

Example

$\#preference(costs, less(weight))\{40 : sauna, 70 : dive\}$

$\#preference(fun, superset)\{sauna, dive, hike, \sim bunji\}$

$\#preference(temps, aso)\{dive > sauna \parallel hot, sauna > dive \parallel \neg hot\}$

$\#preference(all, pareto)\{name(costs), name(fun), name(temps)\}$

$\#optimize(all)$

Outline

Preference

- A strict partial order $\succ$ on the stable models of a logic program
  That is, $X \succ Y$ means that X is preferred to Y
- A stable model X is $\succ$-preferred, if there is no other stable model Y such that $Y \succ X$

- A preference type is a (parametric) class of preference relations

Preference

- A strict partial order $\succ$ on the stable models of a logic program
  That is, $X \succ Y$ means that X is preferred to Y
- A stable model X is $\succ$-preferred, if there is no other stable model Y such that $Y \succ X$
- A preference type is a (parametric) class of preference relations

Preference

- A strict partial order $\succ$ on the stable models of a logic program
  That is, $X \succ Y$ means that X is preferred to Y
- A stable model X is $\succ$-preferred, if there is no other stable model Y such that $Y \succ X$
- A preference type is a (parametric) class of preference relations

Potassco

Preference

- A strict partial order $\succ$ on the stable models of a logic program
  That is, $X \succ Y$ means that X is preferred to Y
- A stable model X is $\succ$-preferred, if there is no other stable model Y such that $Y \succ X$

- A preference type is a (parametric) class of preference relations

Outline

Language

- weighted formula $w_1, \dots, w_l : \phi$
  where each w_i is a term and ϕ is a Boolean formula
- naming atom $name(s)$
  where s is the name of a preference
- preference element $\Phi_1 > \cdots > \Phi_m \parallel \Phi$
  where each Φ_r is a set of weighted formulas and Φ is a non-weighted formula
- preference statement $\#preference(s,t)\{e_1, \dots, e_n\}$
  where s and t represent the preference statement and its type
  and each e_j is a preference element

- optimization directive $\#optimize(s)$
  where s is the name of a preference

- preference specification is a set S of preference statements and a directive $\#optimize(s)$ such that S is an acyclic, closed, and $s \in S$

Language

- weighted formula $w_1, \dots, w_l : \phi$
  where each w_i is a term and ϕ is a Boolean formula
- naming atom $name(s)$
  where s is the name of a preference
- preference element $\Phi_1 > \cdots > \Phi_m \parallel \Phi$
  where each Φ_r is a set of weighted formulas and Φ is a non-weighted formula
- preference statement $\#preference(s, t)\{e_1, \dots, e_n\}$
  where s and t represent the preference statement and its type
  and each e_j is a preference element
- optimization directive $\#optimize(s)$
  where s is the name of a preference
- preference specification is a set S of preference statements and a directive $\#optimize(s)$ such that S is an acyclic, closed, and $s \in S$

Language

- weighted formula $w_1, \dots, w_l : \phi$
  where each w_i is a term and ϕ is a Boolean formula
- naming atom $name(s)$
  where s is the name of a preference
- preference element $\Phi_1 > \cdots > \Phi_m \parallel \Phi$
  where each Φ_r is a set of weighted formulas and Φ is a non-weighted formula
- preference statement $\#preference(s, t)\{e_1, \dots, e_n\}$
  where s and t represent the preference statement and its type
  and each e_j is a preference element
- optimization directive $\#optimize(s)$
  where s is the name of a preference
- preference specification is a set S of preference statements and a directive $\#optimize(s)$ such that S is an acyclic, closed, and $s \in S$

Preference type

- A preference type t is a function mapping a set of preference elements, E, to a (strict) preference relation, $t(E)$, on sets of atoms
- The domain of t, $dom(t)$, fixes its admissible preference elements
- Example $less(cardinality)$

  $(X, Y) \in less(cardinality)(E)$

  $\quad\quad$ if $|\{l \in E \mid X \models l\}| < |\{l \in E \mid Y \models l\}|$

  $dom(less(cardinality)) = \mathcal{P}(\{a, \neg a \mid a \in \mathcal{A}\})$

  (where $\mathcal{P}(X)$ denotes the power set of X)

Potassco

Preference type

- A preference type t is a function mapping a set of preference elements, E, to a (strict) preference relation, $t(E)$, on sets of atoms
- The domain of t, $dom(t)$, fixes its admissible preference elements
- Example $less(cardinality)$

$$(X, Y) \in less(cardinality)(E)$$
$$\text{if } |\{l \in E \mid X \models l\}| < |\{l \in E \mid Y \models l\}|$$

$$dom(less(cardinality)) = \mathcal{P}(\{a, \neg a \mid a \in \mathcal{A}\})$$

(where $\mathcal{P}(X)$ denotes the power set of X)

Preference type

- A preference type t is a function mapping a set of preference elements, E, to a (strict) preference relation, $t(E)$, on sets of atoms
- The domain of t, $dom(t)$, fixes its admissible preference elements
- Example $less(cardinality)$
  - $(X, Y) \in less(cardinality)(E)$
    $$\text{if } |\{l \in E \mid X \models l\}| < |\{l \in E \mid Y \models l\}|$$
  - $dom(less(cardinality)) = \mathcal{P}(\{a, \neg a \mid a \in \mathcal{A}\})$
    (where $\mathcal{P}(X)$ denotes the power set of X)

Preference type

- A preference type t is a function mapping a set of preference elements, E, to a (strict) preference relation, $t(E)$, on sets of atoms
- The domain of t, $dom(t)$, fixes its admissible preference elements
- Example $less(cardinality)$
  - $(X, Y) \in less(cardinality)(E)$
    if $|\{l \in E \mid X \models l\}| < |\{l \in E \mid Y \models l\}|$
  - $dom(less(cardinality)) = \mathcal{P}(\{a, \neg a \mid a \in \mathcal{A}\})$
    (where $\mathcal{P}(X)$ denotes the power set of X)

Preference type

- A preference type t is a function mapping a set of preference elements, E, to a (strict) preference relation, $t(E)$, on sets of atoms
- The domain of t, $dom(t)$, fixes its admissible preference elements
- Example $less(cardinality)$
  - $(X, Y) \in less(cardinality)(E)$
    if $|\{l \in E \mid X \models l\}| < |\{l \in E \mid Y \models l\}|$
  - $dom(less(cardinality)) = \mathcal{P}(\{a, \neg a \mid a \in \mathcal{A}\})$
    (where $\mathcal{P}(X)$ denotes the power set of X)

More examples

- *more*(*weight*) is defined as
  - $(X,Y) \in more(weight)(E)$ if $\sum_{(w:l)\in E, X\models l} w > \sum_{(w:l)\in E, Y\models l} w$
  - $dom(more(weight)) = \mathcal{P}(\{w : a, w : \neg a \mid w \in \mathbb{Z}, a \in \mathcal{A}\})$; and
- *subset* is defined as
  - $(X,Y) \in subset(E)$ if $\{l \in E \mid X \models l\} \subset \{l \in E \mid Y \models l\}$
  - $dom(less(cardinality)) = \mathcal{P}(\{a, \neg a \mid a \in \mathcal{A}\})$.
- *pareto* is defined as
  - $(X,Y) \in pareto(E)$ if $\bigwedge_{name(s)\in E}(X \succeq_s Y) \wedge \bigvee_{name(s)\in E}(X \succ_s Y)$
  - $dom(pareto) = \mathcal{P}(\{n \mid n \in N\})$;
- *lexico* is defined as
  - $(X,Y) \in lexico(E)$ if $\bigvee_{w:name(s)\in E}((X \succ_s Y) \wedge \bigwedge_{v:name(s')\in E, v<w}(X =_{s'} Y))$
  - $dom(lexico) = \mathcal{P}(\{w : n \mid w \in \mathbb{Z}, n \in N\})$.

Potassco

Preference relation

- A preference relation is obtained by applying a preference type to an admissible set of preference elements
- $\#preference(s,t)\,E$ declares preference relation $t(E)$ denoted by $\succ_s$
- Example $\#preference(1, less(cardinality))\{a, \neg b, c\})$ declares

$$X \succ_1 Y \text{ as } |\{l \in \{a, \neg b, c\} \mid X \models l\}| < |\{l \in \{a, \neg b, c\} \mid Y \models l\}|$$

where $\succ_1$ stands for $less(cardinality)(\{a, \neg b, c\})$

Potassco

Preference relation

- A preference relation is obtained by applying a preference type to an admissible set of preference elements
- $\#preference(s,t)\,E$ declares preference relation $t(E)$ denoted by $\succ_s$
- Example $\#preference(1, less(cardinality))\{a, \neg b, c\})$ declares

$$X \succ_1 Y \text{ as } |\{l \in \{a, \neg b, c\} \mid X \models l\}| < |\{l \in \{a, \neg b, c\} \mid Y \models l\}|$$

where $\succ_1$ stands for $less(cardinality)(\{a, \neg b, c\})$

Preference relation

- A preference relation is obtained by applying a preference type to an admissible set of preference elements
- $\#preference(s, t)\, E$ declares preference relation $t(E)$ denoted by $\succ_s$
- Example $\#preference(1, less(cardinality))\{a, \neg b, c\})$ declares

$$X \succ_1 Y \text{ as } |\{l \in \{a, \neg b, c\} \mid X \models l\}| < |\{l \in \{a, \neg b, c\} \mid Y \models l\}|$$

where $\succ_1$ stands for $less(cardinality)(\{a, \neg b, c\})$

Outline

Preference program

- Reification $H_X = \{holds(a) \mid a \in X\}$ and $H'_X = \{holds'(a) \mid a \in X\}$

- Preference program Let s be a preference statement declaring $\succ_s$ and let P_s be a logic program

  We define P_s as a preference program for s, if for all sets $X, Y \subseteq \mathcal{A}$, we have

  $$X \succ_s Y \quad \text{iff} \quad P_s \cup H_X \cup H'_Y \text{ is satisfiable}$$

- Note P_s usually consists of an encoding E_{t_s} of t_s, facts F_s representing the preference statement, and auxiliary rules A

- Note Dynamic versions of H_X and H_Y must be used for optimization

Preference program

- Reification $H_X = \{holds(a) \mid a \in X\}$ and $H'_X = \{holds'(a) \mid a \in X\}$
- Preference program Let s be a preference statement declaring $\succ_s$ and let P_s be a logic program

  We define P_s as a preference program for s, if for all sets $X, Y \subseteq \mathcal{A}$, we have

  $$X \succ_s Y \quad \text{iff} \quad P_s \cup H_X \cup H'_Y \text{ is satisfiable}$$

- Note P_s usually consists of an encoding E_{t_s} of t_s, facts F_s representing the preference statement, and auxiliary rules A
- Note Dynamic versions of H_X and H_Y must be used for optimization

Potassco

Preference program

- Reification $H_X = \{holds(a) \mid a \in X\}$ and $H'_X = \{holds'(a) \mid a \in X\}$
- Preference program Let s be a preference statement declaring $\succ_s$ and let P_s be a logic program

  We define P_s as a preference program for s, if for all sets $X, Y \subseteq \mathcal{A}$, we have

  $$X \succ_s Y \quad \text{iff} \quad P_s \cup H_X \cup H'_Y \text{ is satisfiable}$$

- Note P_s usually consists of an encoding E_{t_s} of t_s, facts F_s representing the preference statement, and auxiliary rules A
- Note Dynamic versions of H_X and H_Y must be used for optimization

Preference program

- Reification $H_X = \{holds(a) \mid a \in X\}$ and $H'_X = \{holds'(a) \mid a \in X\}$
- Preference program Let s be a preference statement declaring $\succ_s$ and let P_s be a logic program

  We define P_s as a preference program for s, if for all sets $X, Y \subseteq \mathcal{A}$, we have

  $$X \succ_s Y \quad \text{iff} \quad P_s \cup H_X \cup H'_Y \text{ is satisfiable}$$

- Note P_s usually consists of an encoding E_{t_s} of t_s, facts F_s representing the preference statement, and auxiliary rules A
- Note Dynamic versions of H_X and H_Y must be used for optimization

$\#preference(3, subset)\{a, \neg b, c\}$

$E_{subset} =$

```
better(P) :- preference(P,subset),
             holds'(X) : preference(P,_,_,for(X),_), holds(X);
             1 #sum { 1,X : not holds(X), holds'(X),
                            preference(P,_,_,for(X),_) }.
```

$F_3 =$

```
preference(3,subset). preference(3,1,1,for(a),()).
                      preference(3,2,1,for(neg(b)),()).
                      preference(3,3,1,for(c),()).
```

$A =$

```
holds(neg(A)) :- not holds(A), preference(_,_,_,for(neg(A)),_).
holds'(neg(A)) :- not holds'(A),preference(_,_,_,for(neg(A)),_).
```

$H_{\{a,b\}} =$

```
holds(a). holds(b).
```

$H'_{\{a\}} =$

```
holds'(a).
```

We get a stable model containing `better(3)` indicating that $\{a, b\} \succ_3 \{a\}$, or $\{a\} \subset \{a, \neg b\}$

Potassco

$$\#preference(3, subset)\{a, \neg b, c\}$$

$E_{subset} =$
```
better(P) :- preference(P,subset),
             holds'(X) : preference(P,_,_,for(X),_), holds(X);
             1 #sum { 1,X : not holds(X), holds'(X),
                            preference(P,_,_,for(X),_) }.
```

$F_3 =$
```
preference(3,subset). preference(3,1,1,for(a),()).
                      preference(3,2,1,for(neg(b)),()).
                      preference(3,3,1,for(c),()).
```

$A =$
```
holds(neg(A)) :- not holds(A), preference(_,_,_,for(neg(A)),_).
holds'(neg(A)) :- not holds'(A),preference(_,_,_,for(neg(A)),_).
```

$H_{\{a,b\}} =$
```
holds(a). holds(b).
```

$H'_{\{a\}} =$
```
holds'(a).
```

We get a stable model containing `better(3)` indicating that $\{a, b\} \succ_3 \{a\}$, or $\{a\} \subset \{a, \neg b\}$

Basic algorithm $solveOpt(P, s)$

Input : A program P over $\mathcal{A}$ and preference statement s
Output : A $\succ_s$-preferred stable model of P, if P is satisfiable, and $\bot$ otherwise

$Y \leftarrow solve(P)$
if $Y = \bot$ **then return** $\bot$

repeat
    $X \leftarrow Y$
    $Y \leftarrow solve(P \cup E_{t_s} \cup F_s \cup R_{\mathcal{A}} \cup H'_X) \cap \mathcal{A}$
until $Y = \bot$
return X

where $R_X = \{holds(a) \leftarrow a \mid a \in X\}$

Sketched Python Implementation

```
#script (python)

from gringo import *
holds = []

def getHolds():
    global holds
    return holds

def onModel(model):
    global holds
    holds = []
    for a in model.atoms():
        if (a.name() == "_holds"): holds.append(a.args()[0])

def main(prg):
    step = 1
    prg.ground([("base", [])])
    while True:
        if step > 1: prg.ground([("doholds",[step-1]),("preference",[0,step-1])]
        ret = prg.solve(on_model=onModel)
        if ret == SolveResult.UNSAT: break
        step = step+1
#end.

#program base.                        #program doholds(m).
#show _holds(X,0) : _holds(X,0).      _holds(X,m) :- X = @getHolds().

#end.
```

Sketched Python Implementation

```
#script (python)

from gringo import *
holds = []

def getHolds():
    global holds
    return holds

def onModel(model):
    global holds
    holds = []
    for a in model.atoms():
        if (a.name() == "_holds"): holds.append(a.args()[0])

def main(prg):
    step = 1
    prg.ground([("base", [])])
    while True:
        if step > 1: prg.ground([("doholds",[step-1]),("preference",[0,step-1])])
        ret = prg.solve(on_model=onModel)
        if ret == SolveResult.UNSAT: break
        step = step+1
#end.

#program base.                     #program doholds(m).
#show _holds(X,0) : _holds(X,0).   _holds(X,m) :- X = @getHolds().

#end.
```

Vanilla `minimize` statements

- Emulating the `minimize` statement

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

in *asprin* amounts to

```
#preference(myminimize,less(weight))
          { C,(X,Y) :: cycle(X,Y) : cost(X,Y,C) }.
#optimize(myminimize).
```

- Note *asprin* separates the declaration of preferences from the actual optimization directive

Vanilla minimize statements

- Emulating the minimize statement

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

  in *asprin* amounts to

```
#preference(myminimize,less(weight))
          { C,(X,Y) :: cycle(X,Y) : cost(X,Y,C) }.
#optimize(myminimize).
```

- Note *asprin* separates the declaration of preferences from the actual optimization directive

Example

in *asprin*'s input language

```
#preference(costs,less(weight)){
  C :: sauna : cost(sauna,C);
  C ::  dive : cost(dive,C)
}.
#preference(fun,superset){ sauna; dive; hike; not bunji }.
#preference(temps,aso){
  dive > sauna ||     hot;
 sauna > dive  || not hot
}.
#preference(all,pareto){name(costs); name(fun); name(temps)}.

#optimize(all).
```

asprin's library

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

asprin's library

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

asprin's library

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

Outline

Potassco

Summary

- asprin stands for "ASP for Preference handling"
- asprin is a general, flexible, and extendable framework for preference handling in ASP
- asprin caters to
  - off-the-shelf users using the preference relations in *asprin*'s library
  - preference engineers customizing their own preference relations

Summary

- asprin stands for "ASP for Preference handling"
- asprin is a general, flexible, and extendable framework for preference handling in ASP
- asprin caters to
  - off-the-shelf users using the preference relations in *asprin*'s library
  - preference engineers customizing their own preference relations

Summary

- asprin stands for "ASP for Preference handling"
- asprin is a general, flexible, and extendable framework for preference handling in ASP
- asprin caters to
  - off-the-shelf users using the preference relations in *asprin*'s library
  - preference engineers customizing their own preference relations

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

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

[3] 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.

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

[5] 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.

[6] 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.

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

[8] 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.

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

[10] G. Brewka, I. Niemelä, and M. Truszczyński.
Answer set optimization.
In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 867–872. Morgan Kaufmann Publishers, 2003.

[11] K. Clark.
Negation as failure.
In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

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

[13] 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.

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

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

[16] E. Di Rosa, E. Giunchiglia, and M. Maratea.
Solving satisfiability problems with preferences.
Constraints, 15(4):485–515, 2010.

[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, A. Harrison, R. Kaminski, V. Lifschitz, and T. Schaub.
Abstract Gringo.
Theory and Practice of Logic Programming, 15(4-5):449–463, 2015.
Available at `http://arxiv.org/abs/1507.06576`.

[27] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, and S. Thiele.
Potassco User Guide.
University of Potsdam, second edition edition, 2015.

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

[29] 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.

[30] 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 [49], pages 250–264.

[31] 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.

[32] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
clasp: A conflict-driven answer set solver.
In Baral et al. [3], pages 260–265.

[33] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set enumeration.
In Baral et al. [3], pages 136–148.

[34] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub.
Conflict-driven answer set solving.
In Veloso [74], pages 386–392.

[35] 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.

[36] 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.

[37] 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.

[38] M. Gebser, M. Ostrowski, and T. Schaub.
Constraint answer set solving.
In Hill and Warren [49], pages 235–249.

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

[40] 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.

[41] 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.

[42] M. Gelfond and Y. Kahl.
Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach.
Cambridge University Press, 2014.

[43] M. Gelfond and N. Leone.
Logic programming and knowledge representation — the A-Prolog perspective.
Artificial Intelligence, 138(1-2):3–38, 2002.

[44] 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.

[45] 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.

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

[47] K. Gödel.
Zum intuitionistischen Aussagenkalkül.
In *Anzeiger der Akademie der Wissenschaften in Wien*, page 65–66. 1932.

[48] A. Heyting.

Die formalen Regeln der intuitionistischen Logik.
In *Sitzungsberichte der Preussischen Akademie der Wissenschaften*, page 42–56. 1930.
Reprint in Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik, Akademie-Verlag, 1986.

[49] 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.

[50] J. Huang.
The effect of restarts on the efficiency of clause learning.
In Veloso [74], pages 2318–2323.

[51] 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.

[52] 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.

[53] 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.

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

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

[56] V. Lifschitz and A. Razborov.
Why are there so many loop formulas?

ACM Transactions on Computational Logic, 7(2):261–268, 2006.

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

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

[59] 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.

[60] J. Marques-Silva, I. Lynce, and S. Malik.
Conflict-driven clause learning SAT solvers.
In Biere et al. [8], chapter 4, pages 131–153.

[61] J. Marques-Silva and K. Sakallah.

GRASP: A search algorithm for propositional satisfiability.
IEEE Transactions on Computers, 48(5):506–521, 1999.

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

[63] 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.

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

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] L. Ryan.
Efficient algorithms for clause-learning SAT solvers.
Master's thesis, Simon Fraser University, 2004.

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

[71] T. Son and E. Pontelli.
Planning with preferences using logic programming.
Theory and Practice of Logic Programming, 6(5):559–608, 2006.

[72] T. Syrjänen.
Lparse 1.0 user's manual, 2001.

[73] 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.

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

[75] 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.