Pocket calculators for hard combinatorial search and optimization problems

Torsten Schaub

University of Potsdam

Potassco

Outline

Outline

Characteristics of good pocket calculators

0. handy
1. easy to use
2. lots of operations
3. computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems

Characteristics of good pocket calculators

0. handy
1. easy to use
2. lots of operations
3. computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems

Potassco

Characteristics of good pocket calculators

0. handy — *ASP may run on your cell phone*
1. easy to use
2. lots of operations
3. computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems

Potassco

Characteristics of good pocket calculators

0. handy ✔ *ASP may run on your cell phone*
1. easy to use
2. lots of operations
3. computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems

Potassco

Characteristics of good pocket calculators

0 handy ✔ — *ASP may run on your cell phone*
1 easy to use — *ASP has a high-level modeling language*
2 lots of operations
3 computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems

Potassco

Characteristics of good pocket calculators

0. handy ✔ — *ASP may run on your cell phone*
1. easy to use ✔ — *ASP has a high-level modeling language*
2. lots of operations
3. computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems

Characteristics of good pocket calculators

0. handy ✔ — *ASP may run on your cell phone*
1. easy to use ✔ — *ASP has a high-level modeling language*
2. lots of operations — *ASP offers various reasoning modes*
3. computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems

Characteristics of good pocket calculators

0. handy ✔ — *ASP may run on your cell phone*
1. easy to use ✔ — *ASP has a high-level modeling language*
2. lots of operations ✔ — *ASP offers various reasoning modes*
3. computes effectively

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems

Potassco

Characteristics of good pocket calculators

0. handy ✔ — *ASP may run on your cell phone*
1. easy to use ✔ — *ASP has a high-level modeling language*
2. lots of operations ✔ — *ASP offers various reasoning modes*
3. computes effectively — *ASP solvers are highly effective*

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems

Characteristics of good pocket calculators

0. handy ✔ — *ASP may run on your cell phone*
1. easy to use ✔ — *ASP has a high-level modeling language*
2. lots of operations ✔ — *ASP offers various reasoning modes*
3. computes effectively ✔ — *ASP solvers are highly effective*

Claim

Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems

Characteristics of good pocket calculators

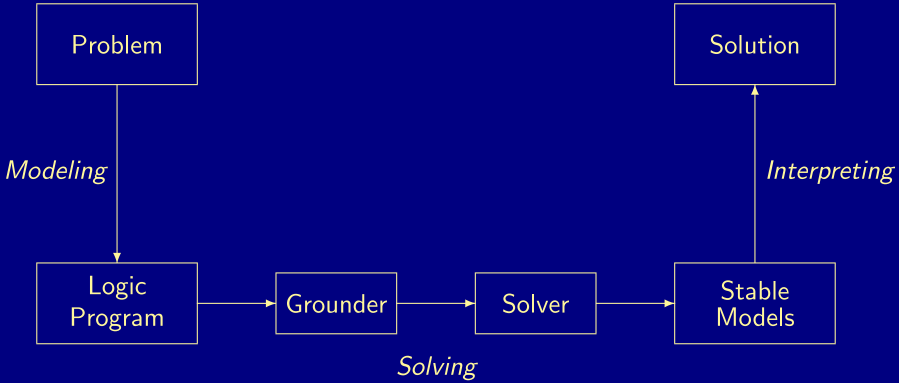
0. handy ✔ — *ASP may run on your cell phone*
1. easy to use ✔ — *ASP has a high-level modeling language*
2. lots of operations ✔ — *ASP offers various reasoning modes*
3. computes effectively ✔ — *ASP solvers are highly effective*

Claim

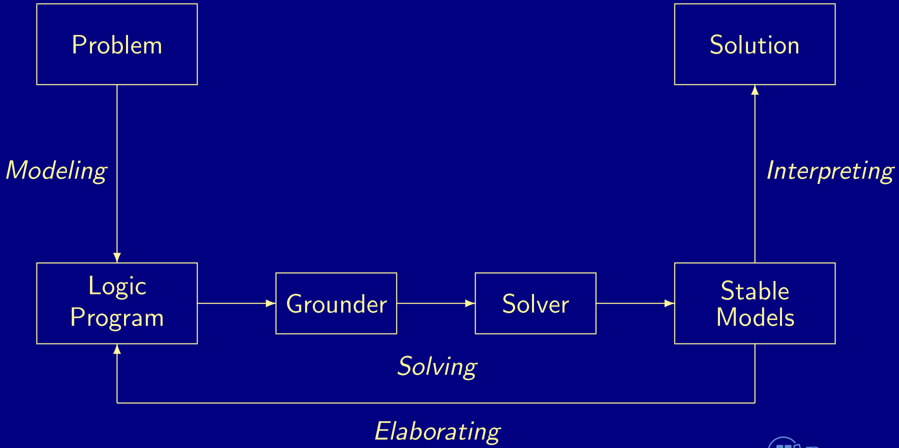
Answer Set Programming (ASP) offers good pocket calculators for hard combinatorial search and optimization problems ✔

Potassco

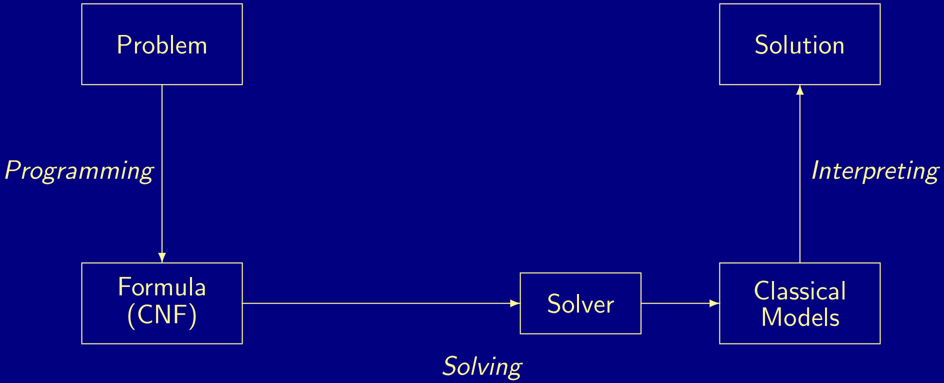
ASP solving


Problem
Solution
Modeling
Interpreting
Logic Program
Grounder
Solver
Stable Models
Solving

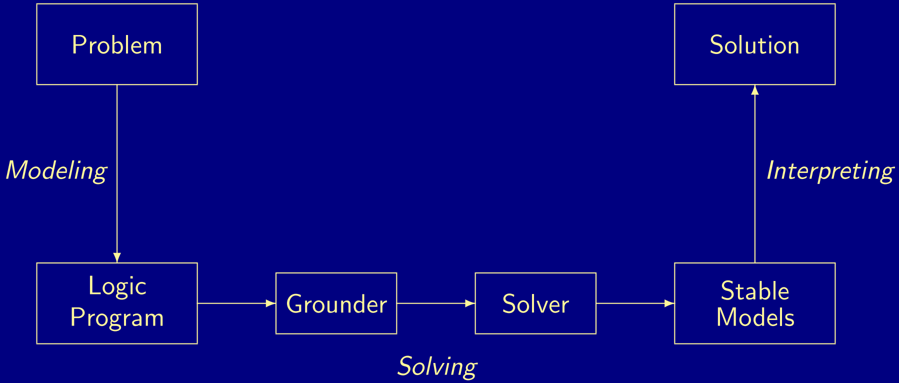
ASP solving


Problem
Solution
Modeling
Interpreting
Logic Program
Grounder
Solver
Stable Models
Solving
Elaborating

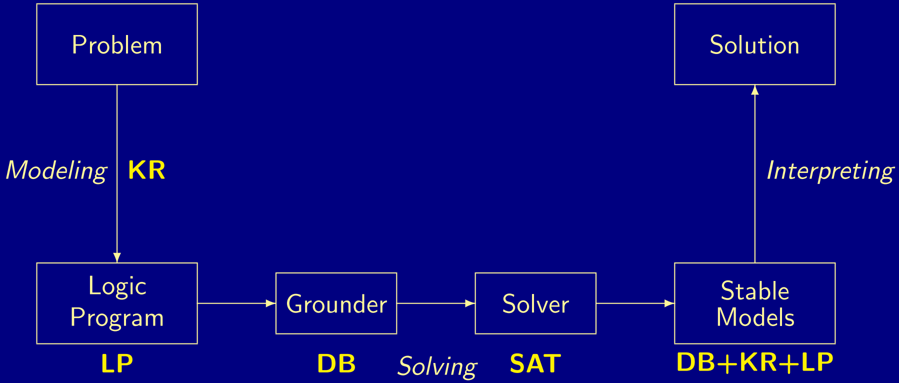
SAT solving


Problem
Solution
Programming
Interpreting
Formula
(CNF)
Solver
Classical
Models
Solving

Rooting ASP solving



Rooting ASP solving



Outline

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization

```
:~ q(X), p(X,C) [C]
#minimize { C : q(X), p(X,C) }
```

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization
  - Weak constraints `:~ q(X), p(X,C) [C]`
  - `#minimize { C : q(X), p(X,C) }`

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization
  - Weak constraints `:~ q(X), p(X,C) [C]`
  - Statements `#minimize { C : q(X), p(X,C) }`

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization
  - Weak constraints `:~ q(X), p(X,C) [C]`
  - Statements `#minimize { C : q(X), p(X,C) }`

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Optimization
  - Weak constraints `:~ q(X), p(X,C) [C]`
  - Statements `#minimize { C : q(X), p(X,C) }`

Potassco

Language constructs

- Variables `p(X) :- q(X)`
- Conditional literals `p :- q(X) : r(X)`
- Disjunction `p(X) ; q(X) :- r(X)`
- Integrity constraints `:- q(X), p(X)`
- Choice `2 { p(X,Y) : q(X) } 7 :- r(Y)`
- Aggregates `s(Y) :- r(Y), 2 #sum{ X : p(X,Y), q(X) } 7`
- Multi-objective optimization
  - Weak constraints `:~ q(X), p(X,C) [C@42]`
  - Statements `#minimize { C@42 : q(X), p(X,C) }`

Potassco

Basic methodology

Methodology

Generate and Test (or: Guess and Check)

- **Generator** Generate potential stable model candidates (typically through non-deterministic constructs)
- **Tester** Eliminate invalid candidates (typically through integrity constraints)

Peanutshell

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

Basic methodology

Methodology

Generate and Test (or: Guess and Check)

- **Generator** Generate potential stable model candidates (typically through non-deterministic constructs)
- **Tester** Eliminate invalid candidates (typically through integrity constraints)

Peanutshell

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

Satisfiability testing

$$(a \leftrightarrow b) \wedge c$$

Satisfiability testing

$(a \leftrightarrow b) \wedge c$

```
{ a ; b ; c }.

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

Maximum satisfiability testing

"$(a \leftrightarrow b) \land c$"

```
{ a ; b ; c }.

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

```
{ a ; b ; c }.

 :- not a, b.
 :~ a, not b.  [10@2]
 :~ not c.    [100@1]
```

n-Queens

Basic encoding

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

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

n-Queens

Advanced encoding

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

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

Traveling salesperson

Basic encoding

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

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

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

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


node(X) :- edge(X,_).
node(X) :- edge(_,X).

edge(1,2).   edge(1,3).   edge(1,4).
edge(2,4).   edge(2,5).   edge(2,6).   [...]
```

Traveling salesperson

Basic encoding

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

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

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

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


node(X) :- edge(X,_).
node(X) :- edge(_,X).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).  [...]
```

Traveling salesperson
Basic encoding

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

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

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

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


node(X) :- edge(X,_).
node(X) :- edge(_,X).

edge(1,2).  edge(1,3).  edge(1,4).
edge(2,4).  edge(2,5).  edge(2,6).  [...]
```

Potassco

Company Controls

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

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

Outline

Potassco

Reasoning modes

- ASP solvers offer
  - Satisfiability testing
  - Enumeration
  - Projection
  - Intersection
  - Union
  - Optimization

  - and combinations of them

- For instance, *clasp* allows for
  - ASP solving (*smodels* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)

Reasoning modes

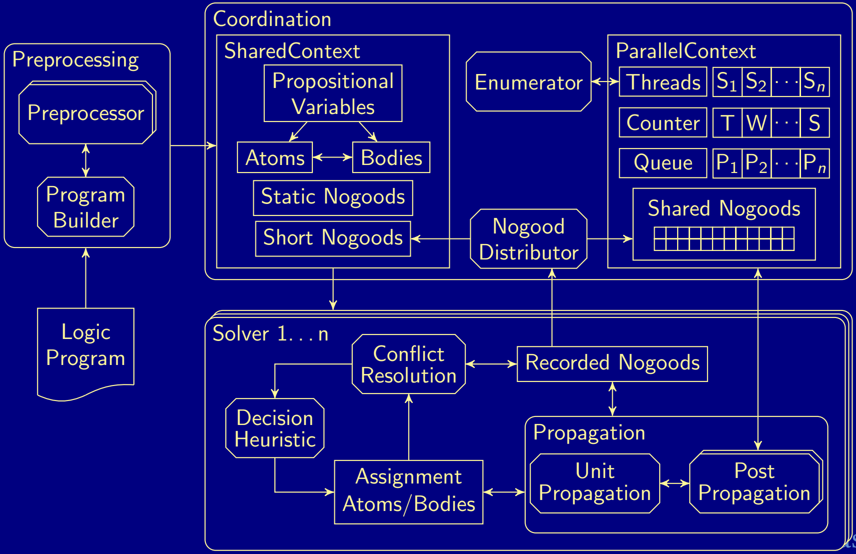
- ASP solvers offer
  - Satisfiability testing
  - Enumeration
  - Projection
  - Intersection
  - Union
  - Optimization
  - and combinations of them
- For instance, *clasp* allows for
  - ASP solving (*smodels* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)

Reasoning modes

- ASP solvers offer
  - Satisfiability testing
  - Enumeration
  - Projection
  - Intersection
  - Union
  - Optimization
  - and combinations of them
- For instance, *clasp* allows for
  - ASP solving (*smodels* format)
  - MaxSAT and SAT solving (extended *dimacs* format)
  - PB solving (*opb* and *wbo* format)

Potassco

Multi-threaded architecture of *clasp*



Outline

Potassco

Well then . . . ?

Claim

ASP offers good pocket calculators for hard combinatorial search and optimization problems

0. handy — *ASP may run on your cell phone* ✘
1. easy to use — *ASP has a high-level modeling language* ✔
2. lots of operations — *ASP offers various reasoning modes* ✔
3. computes effectively — *ASP solvers are highly effective* ✔

Potassco

Well then . . . ?

Claim

ASP offers good pocket calculators for hard combinatorial search and optimization problems

0. handy — *ASP may run on your cell phone* ✘
1. easy to use — *ASP has a high-level modeling language* ✔
2. lots of operations — *ASP offers various reasoning modes* ✔
3. computes effectively — *ASP solvers are highly effective* ✔

Well then . . . ?

Claim

ASP offers good pocket calculators for hard combinatorial search and optimization problems ✔

0 handy — *ASP may run on your cell phone* ✘
1 easy to use — *ASP has a high-level modeling language* ✔
2 lots of operations — *ASP offers various reasoning modes* ✔
3 computes effectively — *ASP solvers are highly effective* ✔

Potassco

ASPyclopedia

- Systems
  - asperix http://www.info.univ-angers.fr/pub/claire/asperix
  - assat http://assat.cs.ust.hk
  - clasp, gringo, clingo, etc. http://potassco.sourceforge.net
  - cmodels http://www.cs.utexas.edu/users/tag/cmodels
  - dlv http://www.dlvsystem.com
  - lp2* http://research.ics.aalto.fi/software/asp
  - smodels, lparse, gnt http://www.tcs.hut.fi/Software
  - wasp https://www.mat.unical.it/ricca/wasp
  - sup http://www.cs.utexas.edu/users/tag/sup
- User's guides
  - DLV Systems http://www.dlvsystem.com/html/DLV_User_Manual.html
  - Potassco http://sourceforge.net/projects/potassco/files/guide
- Literature [1, 6, 8, 14], [13, 7, 4, 3], [10, 11, 2, 16, 15, 12, 9, 5], etc.

ASPyclopedia

- Systems — *best suited for beginners*
  - clingo http://potassco.sourceforge.net
  - dlv http://www.dlvsystem.com
- User's guides
  - DLV Systems http://www.dlvsystem.com/html/DLV_User_Manual.html
  - Potassco http://sourceforge.net/projects/potassco/files/guide
- Literature [1, 6, 8, 14], [13, 7, 4, 3], [10, 11, 2, 16, 15, 12, 9, 5], etc.

ASPyclopedia

- Systems
  - asperix http://www.info.univ-angers.fr/pub/claire/asperix
  - assat http://assat.cs.ust.hk
  - clasp, gringo, clingo, etc. http://potassco.sourceforge.net
  - cmodels http://www.cs.utexas.edu/users/tag/cmodels
  - dlv http://www.dlvsystem.com
  - lp2* http://research.ics.aalto.fi/software/asp
  - smodels, lparse, gnt http://www.tcs.hut.fi/Software
  - wasp https://www.mat.unical.it/ricca/wasp
  - sup http://www.cs.utexas.edu/users/tag/sup
- User's guides
  - DLV Systems
    http://www.dlvsystem.com/html/DLV_User_Manual.html
  - Potassco
    http://sourceforge.net/projects/potassco/files/guide
- Literature [1, 6, 8, 14], [13, 7, 4, 3], [10, 11, 2, 16, 15, 12, 9, 5], etc.

Potassco

ASPyclopedia

- Systems
  - asperix http://www.info.univ-angers.fr/pub/claire/asperix
  - assat http://assat.cs.ust.hk
  - clasp, gringo, clingo, etc. http://potassco.sourceforge.net
  - cmodels http://www.cs.utexas.edu/users/tag/cmodels
  - dlv http://www.dlvsystem.com
  - lp2* http://research.ics.aalto.fi/software/asp
  - smodels, lparse, gnt http://www.tcs.hut.fi/Software
  - wasp https://www.mat.unical.it/ricca/wasp
  - sup http://www.cs.utexas.edu/users/tag/sup
- User's guides
  - DLV Systems
    http://www.dlvsystem.com/html/DLV_User_Manual.html
  - Potassco
    http://sourceforge.net/projects/potassco/files/guide
- Literature [1, 6, 8, 14], [13, 7, 4, 3], [10, 11, 2, 16, 15, 12, 9, 5], etc.

ASPyclopedia

- Systems
  - asperix http://www.info.univ-angers.fr/pub/claire/asperix
  - assat http://assat.cs.ust.hk
  - clasp, gringo, clingo, etc. http://potassco.sourceforge.net
  - cmodels http://www.cs.utexas.edu/users/tag/cmodels
  - dlv http://www.dlvsystem.com
  - lp2* http://research.ics.aalto.fi/software/asp
  - smodels, lparse, gnt http://www.tcs.hut.fi/Software
  - wasp https://www.mat.unical.it/ricca/wasp
  - sup http://www.cs.utexas.edu/users/tag/sup
- User's guides
  - DLV Systems
    http://www.dlvsystem.com/html/DLV_User_Manual.html
  - Potassco
    http://sourceforge.net/projects/potassco/files/guide
- Literature [1, 6, 8, 14], [13, 7, 4, 3], [10, 11, 2, 16, 15, 12, 9, 5], etc.

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

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

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

[4] 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.

[5] 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`.

[6] 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.

[7] 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.

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

Cambridge University Press, 2014.

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

[10] 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.

[11] 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.

[12] V. Lifschitz.

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

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

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

[15] 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.

[16] I. Niemelä.
Logic programs with stable model semantics as a constraint programming paradigm.

Potassco

Annals of Mathematics and Artificial Intelligence, 25(3-4):241–273, 1999.