Answer Set Programming and Extensions

Unit 2 – Program Evaluation

Thomas Eiter

Institut für Informationsysteme, TU Wien

VTSA Summer School 2016, Liège, August 29-September 2, 2016

- Austrian Science Fund (FWF) grants P24090, P26471, P27730

Unit Outline

1. Introduction
2. Solvers
3. Intelligent Grounding
4. Solving
5. Grounding Bottleneck

ASP Program Evaluation

Efficient evaluation of answer set programs is challenging

- NP-/Σ_2^p-completeness of consistency already in the ground (propositional) case
- different reasoning tasks
  - single solution
  - solution enumeration
  - query answering (reasoning from multiple models) projection, intersection, union
  - (multi-objective) optimization and preference
  - additional constructs (weak constraints, aggregates, function symbols, etc.)
- complex data in applications (large data volumes)

Answer Set Solvers

(see also http://en.wikipedia.org/wiki/Answer_set_programming)

ASPERIX	www.info.univ-angers.fr/pub/claire/asperix/
ASSAT	assat.cs.ust.hk/
CLASP [1]	potassco.sourceforge.net/#clasp/
CMODELS	www.cs.utexas.edu/users/tag/cmodels/
DLV [2]	www.dbai.tuwien.ac.at/proj/dlv/
ASPTOOLS	research.ics.aalto.fi/software/asp/
o ME-ASP	www.mat.unical.it/ricca/me-asp/
OMIGA	www.kr.tuwien.ac.at/research/systems/omiga
SMODELS	www.tcs.hut.fi/Software/smodels/
WASP	www.mat.unical.it/ricca/wasp/
XASP	xsb.sourceforge.net/, distributed with XSB
	....

[1] + CLASPD, CLINGO, CLINGCON etc. (http://potassco.sourceforge.net/)
[2] + DLVHEX, DLV^{DB}, DLT, DLV-COMPLEX, ONTO-DLV etc.

- Many ASP solvers are available (mostly function-free programs)
- Efforts to realize tractable fragments (*downscaling*)
- clasp was first ASP solver competitive to top SAT solvers
- ASP Solver competition (at LPNMR, since 2007)

Evaluation Approaches

- Different methods and evaluation approaches:
  - resolution-based [Bonatti *et al.*, 2008]
  - forward chaining [Lefèvre *et al.*, 2015]
  - lazy grounding: [Palù *et al.*, 2009], [Dao-Tran *et al.*, 2012], [de Cat *et al.*, 2012]
  - translation-based (see below)
  - meta-interpretation [E_ *et al.*, 2003]
- Predominant:

intelligent grounding + model search (solving)

Architecture of ASP Solvers

Common: two level architecture

1. **Intelligent Grounding**

   Given a program P, generate a (subset) of $grnd(P)$ that has the same models

2. **Solving: Model Search**

   More complicated than in SAT/CSP Solving:

   - candidate generation (classical model)
   - model checking (stability, foundedness!)
     - for SAT, model checking is in logarithmic space (in fact in ALOGTIME)
     - for normal propositional programs, model checking is PTime-complete
     - for disjunctive propositional programs, model checking is co-NP-complete

Intelligent Grounding

- Grounding is a hard problem

$$bit(0).\ \ bit(1). \tag{1}$$

$$p(X1, ..., Xn) \leftarrow bit(X1), ..., bit(Xn). \tag{2}$$

  naive: 2^n rules
- In the worst case, grounding time is exponential in the input size.
- Getting the "right" rules is difficult, already for positive programs
  - rule matching is NP-hard
  - deciding rule relevance is EXPTIME-hard
- Efficient grounding is at the heart of current systems
  - dlv's grounder (built-in);
  - lparse (smodels), gringo (clasp)
- Special techniques used:
  - *"safe rules"* (dlv): each variable in a rule occurs in the body in an unnegated atom with non-built-in predicate (exception: $X = c$)
  - *domain-restriction* (smodels)
  - Deductive DB methods: semi-naive evaluation, magic sets, ...

Grounding: Basic Ideas

- Avoid useless rule instances not applicable in any answer set

Example

$$c(1,2). \quad (3)$$

$$a(X)|b(Y) \leftarrow c(X,Y). \quad (4)$$

The full instantiation of the only rule yields

$$a(1) \mid b(1) \leftarrow c(1,1). \quad (5) \qquad a(2) \mid b(2) \leftarrow c(2,2). \quad (7)$$

$$a(2) \mid b(1) \leftarrow c(2,1). \quad (6) \qquad a(1) \mid b(2) \leftarrow c(1,2). \quad (8)$$

- (5)-(7) are useless: c(1,1), c(2,1), and c(2,2) do not occur in heads

- Basic instantiation:
  - body matching of variables (assignment)
  - backtracking on assignment
  - use *evaluation order*: $r_1 : p \leftarrow B_1. \quad r_2 : q \leftarrow p, B_2.$
    - must evaluate r_1 before r_2
    - $\Rightarrow$ use syntactic *dependency graph*

Optimizations

Intelligent grounders use a big deal of optimization techniques

- **literal ordering** in the body (= join optimization in DBs)
- **backjumping algorithm** [Perri *et al.*, 2008]
- **magic sets**
  - classic **magic sets technique** (standard Datalog): emulate goal-directed query answering (top-down) in bottom up computation
    - restrict rule instances using “magic predicates”
  - **dynamic magic sets** [Alviano *et al.*, 2012] for disjunctive programs with negation, exploit magic set information also during search [Alviano and Faber, 2011]
  - useful for hard problems in consistent query answering [Manna *et al.*, 2015]
- **parallel instantiation:**
  - exploit modern multi-core/-proc architectures with load balancing and granularity control [Perri *et al.*, 2013]
  - lparse [Pontelli *et al.*, 2003]

Grounding: Function Symbols

- Function symbols: infinite grounding, if even answer sets are finite

Example

$$p(0). \tag{9}$$

$$p(X) \leftarrow p(f(X)), \text{not } q(X). \tag{10}$$

$$q(X) \leftarrow p(f(X)), \text{not } p(X). \tag{11}$$

- dlv-grounder, gringo admit recursive function symbols
- Termination for *finitely-ground (FG) programs* [Calimeri *et al.*, 2008]
  - problem: only semi-decidable
  - Turing complete (each computable function expressible)
- **Practice:** decidable syntactic / semantic restrictions (preprocessing)
  - ω-restrictedeness [Syrjänen, 2001] (lparse)
  - λ-restrictedeness [Gebser *et al.*, 2007] (gringo x, x$<$3)
  - argument-restrictedness [Lierler and Lifschitz, 2009]
  - finite domain [Calimeri *et al.*, 2008] (dlv)
  - Γ-acyclic programs [Greco *et al.*, 2012]

Finitely-Ground (FG) Programs

Idea:

- *Modularization:* split a program P into modules $P_1 \ldots P_n$, define a proper module ordering $\prec$ for bottom up evaluation
- *Simplification:*
  - instantiate a given module P_i of P by exploiting instantiations of previous modules $\rightarrow$ "intelligent" instantiation
  - add only ground rules whose heads have a chance to be true in some answer set
- All P_i are finitely instantiated along $\prec \Rightarrow P'$ as above is found

Pros:

- FG programs are very expressive: they correspond to terminating computations of Turing Machines
- FG programs have a finite set of finite answer sets

Cons:

- Undecidable (semi-decidable) in general, not recognizable

Finite Domain (FD) Programs

- Simple, decidable subclass of FG programs [Calimeri *et al.*, 2008]
- All arguments of a program P must be *finite domain*
- Use an *argument dependency graph* $AG(P)$ of P:
  - nodes = argument positions of predicates in P; eg. $edge[1]$, $path[2]$,
  - edge $p[i] \rightarrow q[j]$, if in some rule,
    - (i) $p(t_1, \ldots, t_n)$ is in the head,
    - (ii) $q(s_1, \ldots, s_m)$ occurs positive in the body, and
    - (iii) t_i and s_j share some variable.
- Program P is *finite domain* if, for each atom $p(t_1, \ldots, t_n)$ in the head of a rule $r \in P$, and for each argument $p[i]$ of p, either
  - (i) t_i is variable-free; or
  - (ii) t_i occurs as a (sub)term of an atom not under "not " in the body; or
  - (iii) each variable occurring in t_i occurs in a positive body atom $q(s_1, \ldots, s_m)$ in some s_j s.t. $p[i]$ and $q[j]$ are not on a cycle in $AG(P)$.

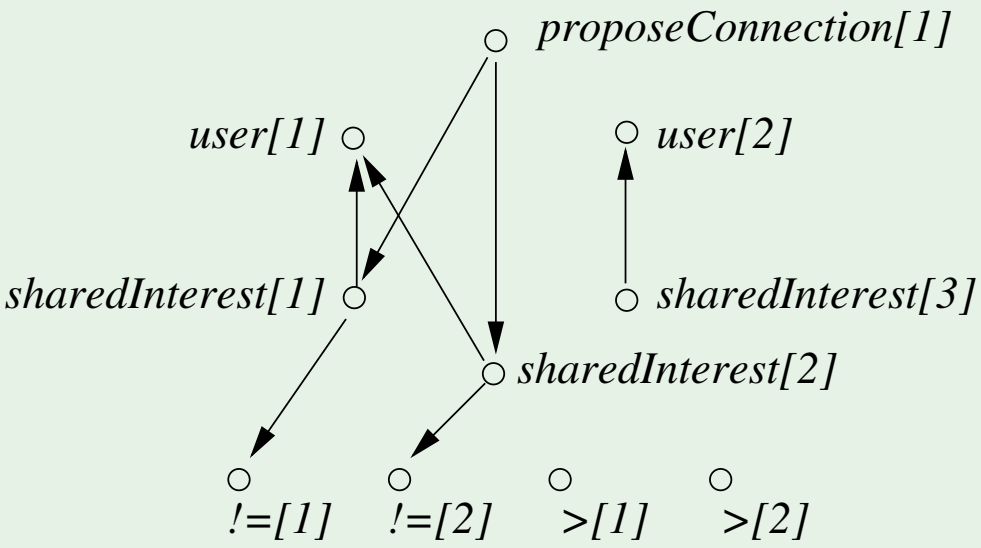
Examples

More than two shared interests

$$sharedInterests(U_1, U_2, \#intersection(S_1, S_2)) \leftarrow user(U_1, S_1), user(U_2, S_2), U_1 \neq U_2.$$

$$proposeConnection(pair(U_1, U_2)) \leftarrow sharedInterests(U_1, U_2, S), \#card(S) > 2.$$

is finite domain.

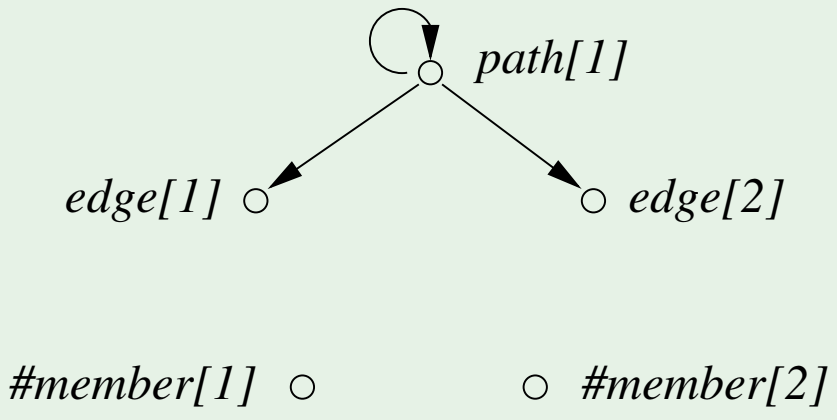


Examples (ctd.)

Paths in a graph

$$path([X,Y]) \leftarrow edge(X,Y).$$
$$path([X|[Y|W]]) \leftarrow edge(X,Y),\ path([Y|W]),\ \text{not}\ \ \#member(X,[Y|W]).$$

is not finite domain:

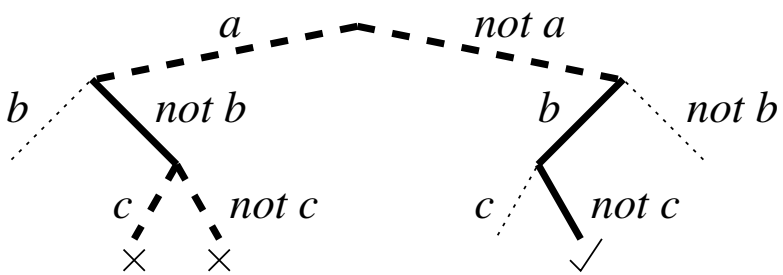


- dlv: "`Termination is not guaranteed ... Use option -nofinitecheck in order to evaluate this program anyway.`"
- still finitely-ground for finite $edge \Rightarrow$ dlv terminates
- easily modified to compute e.g. all Hamiltonian paths from s to t:

Solving: Model Search

- Applied to ground programs.
- Early solvers (e.g. smodels, dlv): native methods
  - inspired by Davis-Putnam-Logemann Loveland (DPLL) for SAT
    - 3 basic operations: decision, propagate, backtrack
  - special propagation for ASP, e.g.,
    - smodels: 5 propagators
    - dlv: *well-founded*, *must-be-true* propagation (supportedness)

$a{:-}\ not\ b.$
$b{:-}\ not\ a.$
$c{:-}\ not\ c,\ a.$



    - important: heuristics (which atom/rule is next?) [Faber *et al.*, 2001]
    - chronological backtrack-search was improved introducing backjumping and look-back heuristics [Maratea *et al.*, 2008]
    - Recent: Abstract solving [Brochenin *et al.*, 2014]
- Stability check: unfounded sets, reductions to UNSAT [Leone *et al.*, 1997b], [Koch *et al.*, 2003]

ASP Solving Approaches

- Predominant to date: modern SAT techniques
- Export of techniques for optimal answer sets to SAT
- **Genuine conflict-driven ASP solvers**
  - clasp [Gebser *et al.*, 2012], wasp [Alviano *et al.*, 2013]
- **Translation based solving:** to
  - SAT: assat [Lin and Zhao, 2004], cmodels [Lierler, 2005], lp2sat [**?**], [Giunchiglia *et al.*, 2006] (multiple SAT solver calls)
  - SMT: [Niemelä, 2009] aspmt [Bartholomew and Lee, 2014]
  - Mixed IP: [Bell *et al.*, 1994], [Liu *et al.*, 2012]; CPLEX backend
- **Cross translation:** intermediate format to ease cross translation
  - SAT modulo acyclicity [Gebser *et al.*, 2014]
    - interconnect graph based constraints with clausal constraints
    - can postpone choice of the target format to last step
- **Portfolio solvers**
  - claspfolio [Gebser *et al.*, 2011]: combines variants of clasp
  - me-asp [Maratea *et al.*, 2014]: multi-engine portfolio ASP solver

Program Decomposition

- A set S of (ground) atoms is a *splitting set* of a (ground) program P, if for each rule $r : H(r) \leftarrow B(r)$ in P either (i) $H(r) \cup B(r) \subseteq S$ or $H(r) \cap S = \emptyset$.
- $b_S(P)$ and $t_S(P)$ denotes all rules (i) and (ii), resp.

Example

E.g. $P = \left\{ \begin{array}{ll} f_1 & man(d). \\ r_1 & husband(d) \vee single(d) \leftarrow man(d). \\ r_2 & bachelor(d) \leftarrow man(d) \text{ not } husband(d). \end{array} \right\}$

$S = \{man(d), husband(d), single(d)\}$, $b_S(P) = \{f_1, r_1\}$, $t_S(P) = \{r_2\}$

Splitting Set Theorem, Lifschitz and Turner [1994]

For any splitting set S of program P, $AS(P) = \bigcup_{M \in AS(b_S(P))} AS(t_S(P) \cup M)$.

- Splitting sets allow for (de)composition and bottom up evaluation
  - $AS(b_S(P)) = \{M_1 = \{man(d), single(d)\}, M_2 = \{man(d), husband(d)\}\}$,
  - $AS(t_S(P) \cup M_1) = M_1 \cup \{bachelor(d)\}$, $AS(b_S(P) \cup M_2) = M_2$
- has also nonground versions
- solvers identify splitting sets (e.g. strongly connected components)

Characterizing Answers Sets in Classical Logic

- Answer sets of a program P are special classical models of P
  - view rule $r : a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$ (= $a \leftarrow B(r)$) as implication $b_1 \wedge \cdots \wedge b_m \wedge \neg c_1 \wedge \cdots \wedge \neg c_n \rightarrow a$ (=: $BF(r) \rightarrow a$)
- Clark [1978]: for Horn P, turn $\rightarrow$ into $\leftrightarrow$:

$$CF(P) = \{ \bigvee_{a \leftarrow B(r) \in P} BF(r) \rightarrow a \mid a \text{ is an atom } \}$$

- "Completion" $CF(P)$ captures $AS(P)$ for special normal programs

Theorem

For normal logic programs P without cyclic positive recursion (called tight programs*), $AS(P)$ is captured by $CF(P)$ (i.e., its classical models).*

Example

- $P = \{p \leftarrow q\}$: $CF(P) = \{p \leftrightarrow q,\ q \leftrightarrow \bot\}$ P is tight ✓
- $P = \{p \leftarrow q; q \leftarrow p, \text{not } r\}$: $CF(P) = \{p \leftrightarrow q,\ q \leftrightarrow p \wedge \neg r,\ r \leftrightarrow \bot\}$ P is not tight ×

Loop Formulas

$AS(P)$ is captured by adding *loop formulas* $LF(P)$ [Lin and Zhao, 2002]

Theorem

For any normal program P, $AS(P) = CF(P) \cup LF(P)$.

- Informally, $LF(P)$ contains clauses enforcing that a positive cycle C can only by true if some $r : a \leftarrow B(r)$ exists s.t. $a \in C$, $B(r)$ is satisfied and no atom from C occurs positively in $B(r)$.

Example

- $P = \{p \leftarrow q\}$: $LF(P) = \emptyset$

  $CF(P) \cup LF(P) = \{p \leftrightarrow q,\ q \leftrightarrow \bot\} \equiv \neg p \wedge \neg q$
- $P = \{p \leftarrow q,\ q \leftarrow p, \text{not } r\}$: $\quad LF(P) = \{p \wedge q \rightarrow \bot\}$

  $CF(P) \cup LF(P) = \{p \leftrightarrow q,\ q \leftrightarrow p \wedge \neg r,\ r \leftrightarrow \bot, p \wedge q \rightarrow \bot\} \equiv \neg p \wedge \neg q \wedge \neg r$

Loop Formulas /2

- Also feasible for disjunctive programs, non-propositional program [Lee and Lifschitz, 2003], [Chen *et al.*, 2006], and aggregates.
- Based on this, some ASP solvers employ SAT solvers.
- Semantically, loop formulas correspond to *unfounded-freeness* [Leone *et al.*, 1997a], [Lee, 2005]
- **Downside:** exponentially many loop formulas exist in general [Lifschitz and Razborov, 2006]
  - add loop formulas *on the fly*
- Alternative: avoid explicit loop formulas, do cautious description of reachability (*ordered completion*, [Asuncion *et al.*, 2012])

Conflict-Driven ASP Solving

- **Breakthrough in SAT**: **conflict driven clause learning (CDCL)** [Silva *et al.*, 2009]
  - represent clauses as nogoods (inadmissible assignments)
  - propagate with conflict-driven backtracking
  - clause (nogood) learning from conflicts

Example

- $(\neg a \vee b \vee c)$, $(d \vee a)$, $(d \vee \neg b) \rightsquigarrow$ nogoods $\{\mathbf{T}a, \mathbf{F}b, \mathbf{F}c\}$, $\{\mathbf{F}d, \mathbf{F}a\}$, $\{\mathbf{F}d, \mathbf{T}b\}$
- assignment $\{\mathbf{F}c\}$; set $\mathbf{F}d \Rightarrow$ derive $\mathbf{T}a$, $\mathbf{F}b$: conflict!
- learn noogod $\{\mathbf{F}c, \mathbf{F}d\}$

- **ASP: key idea** [Gebser *et al.*, 2012]
  - exploit $AS(P) = CF(P) + LF(P)$
  - use completion nogoods for $CF(P)$, loop nogoods for $LF(P)$
  - map inferences in ASP onto unit propagation on nogoods
  - enable learning

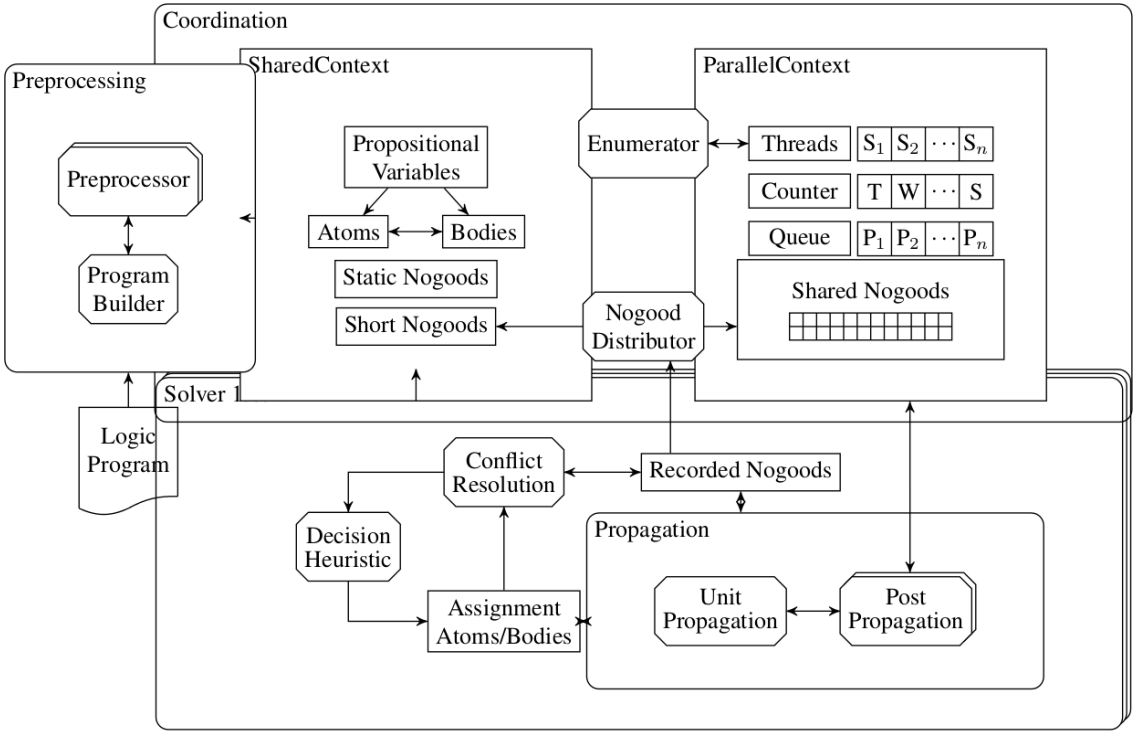
A Glimpse on clasp

- Basic CDCL algorithm

```
(I) initialize                                                    \\ initialize watches
    loop
        propagate completion, loop, and recorded nogoods \\ deterministically assign literals
(C)     if no conflict then
            if all variables assigned then
                return variable assignment                        \\ answer set found
(P)         else
                if no nogood recorded then decide                 \\ non-det. assign literal
        else
            if top-level conflict then return unsatisfiable
            else
                analyze                          \\ resolve conflict and record a conflict
(U)             backjump                 \\ undo assignments until conflict constraint is unit
```

- use unit propagation for completion nogoods (to a fixpoint)
- special loop nogood propagator
- Workflow:
  1. $P \Rightarrow P'$ simplify
  2. $P' \Rightarrow CF(P')$ (which is static)
  3. construct nogoods for $LF(P')$ *on demand*, using a dependency graph

A Glimpse on clasp /2



- multi-threading: parallel execution of clasp (search space splitting, competitive strategies)
  - learned conflict nogoods can be exchanged
- further post propagation is usable for theory-specific propagation
  - exploited e.g. by solvers clingcon, dlvhex

Preference and Optimization

- Two classes [Delgrande *et al.*, 2004]
  1. **prescriptive**: condition on how rules are/must be applied

     implementation by compilation [Delgrande *et al.*, 2003] or meta-interpretation [E_ *et al.*, 2003]
  2. **descriptive**: how answer sets relate (comparison)

     preferences among the answer sets of a program [Brewka *et al.*, 2003], [Sakama and Inoue, 2000], [Son and Pontelli, 2006]
- Class 2) has usually higher computational complexity than 1)
- Tools:
  - ASPRIN system [Brewka *et al.*, 2015]: general and flexible framework to compute optimal answer sets relative to preferences among them
  - plp tool [Delgrande *et al.*, 2000]: compiler for preferences `http://www.cs.uni-potsdam.de/~torsten/plp/`

Grounding Bottleneck

- Recall: Grounding is expensive in general
- **Practical Grounding Bottleneck** [E_ *et al.*, 2007b]
  - Unless EXPTIME = PSPACE, grounding + solving needs unavoidably exponential space in general
  - Certain programs are evaluable in polynomial space, while the top grounders still produce an exponentially large ground programs
- Example: programs P with predicate arities bounded by a constant

Theorem [E *et al.*, 2007a]

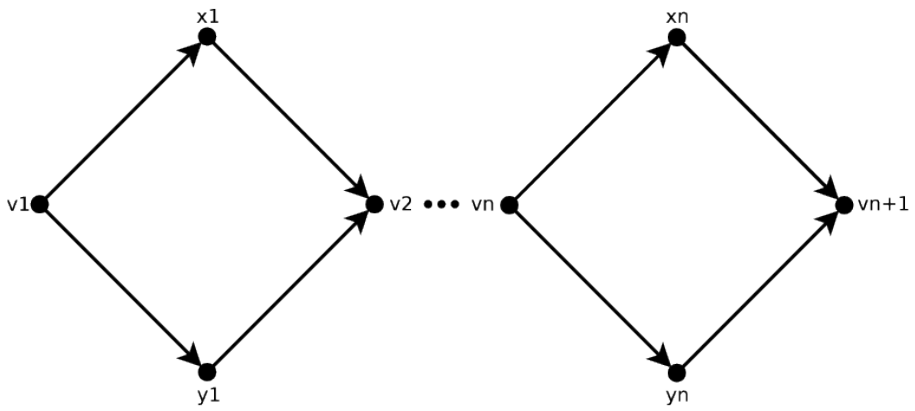
In the datalog case, deciding whether a bounded-arity program P has some answer set is (i) Σ_2^p-complete if P is normal, and (ii) Σ_2^p-complete if P is disjunctive.

- For some families $\mathbf{P}_n = P_1, P_2, \ldots$ of bounded arity programs, the grounders produce exponentially growing output

Examples

- reachability over paths of length k in a directed graph G, given by $e(v_1, v_2)$:

$$p_k(X_1, X_k) \leftarrow e(X_1, X_2), \ldots, e(X_{k-1}, X_k).$$
$$reachable(X, Y) \leftarrow p_k(X, Y).$$
$$reachable(X, Y) \leftarrow reachable(X, Z), p_k(Z, Y).$$



  - for $k = 2n + 1$ there are 2^n paths from v_1 to $v_{n+1} \Rightarrow p_k(v_1, v_{n+1})$ has 2^n derivations
  - if G is nondeterministically chosen by other rules (possibly depending on p_k) grounders don't optimize

Examples /2

- Monadic choice:

$$
\begin{array}{ll}
b(0).\ \ b(1). & (1) \\
p(X) \leftarrow b(X), \text{not}\ \ q(X). & (2) \\
q(X) \leftarrow b(X), \text{not}\ \ p(X). & (3) \\
r(X_1) \leftarrow p(X_1), \ldots, p(X_k), b(X_1), \ldots b(X_k). & (4)
\end{array}
$$

  - as above, exponential grounding result
  - modular evaluation (1),(2) before (3) can be spoiled e.g. with adding $q(X) \leftarrow r(X)$

Nonground ASP: Special Techniques

Techniques to address grounding explosion

- **lazy grounding**
- for **bounded predicate arities**
  - few generating rules property [E_ *et al.*, 2010]
  - answer set size and number of supporting rules is polynomial
  - can run in polynomial space
- **rule decomposition** [Morak and Woltran, 2012], [Bichler *et al.*, 2016]
  - split (long) rule $r : H \leftarrow B$ using decomposition of its hypergraph $H_r = (V_r, E_r)$
  - V_r = variables in B, E_r = literals (labeled with predicate)
  - the decomposition effect relies on the *treewidth of* H_r, $tw(H_r)$ (“treelikeness”)
  - avoids explosion for $tw(H_r) \leq k$, constant k (thus bounded arities)
  - tool (for full ASP-syntax) `http://dbai.tuwien.ac.at/research/project/lpopt/`

Example

- Single rule $r: \quad h(X,W) \leftarrow e(X,Y), e(Y,Z), \text{not } e(Z,W), e(W,X)$
  - $V_r = \{X, Y, X, W\}$,
  - $E_r = \{e_1{=}\{X,W\}, e_2{=}\{X,Y\}, e_3{=}\{Y,Z\}, e_4{=}\{Z,W\}, e_5{=}\{W,X\}\}$
- Tree Decomposition: $\mathcal{T} = (T, \chi)$, $T = (\{n_1, n_2\}, \{n_1 \rightarrow n_2\})$

$$\begin{array}{ll} n_1 \bullet & \chi(n_1) = \{X,Y,W\}, \quad \text{covers } h(X,W), e(X,Y), e(W,X) \\ \downarrow & \\ n_2 \bullet & \chi(n_2) = \{Y,Z,W\}, \quad \text{covers } e(Y,Z), \text{not } e(Z,W) \end{array}$$

- Decomposition algorithm yields (fresh predicate t_1, dom_W):

$$\begin{array}{l} h(X,W) \leftarrow e(X,Y), e(X,W), t_1(Y,W). \\ dom_W(W) \leftarrow e(W,X). \\ t_1(Y,W) \leftarrow e(Y,Z), \text{not } e(Z,W), dom_W(W). \end{array}$$

- Runs for bounded treewidth, i.e. $\max_{n \in T} |\chi(n_i)| - 1 = O(1)$ in polynomial space.

Application: Propositional ASP and QSAT

Bichler *et al.* [2016] (ICLP 2016)

- Transform prop. disjunctive ASP P into normal ASP w/ bounded arities
  - represent P as facts
  - few fixed rules (independent of P): guess M, check $M \models P$
  - few long rules: check foundedness of M
- Customized Encoding: 2-QSAT ($\forall\exists$-QBF satisfiability)
- Similarly, encode Σ_3^p problems into disjunctive ASP w/ bounded arities
  - 3-QSAT ($\exists\forall\exists$-QBF satisfiability)
  - stable cautious abductive reasoning [E_ *et al.*, 1997]
- For 2-QSAT, results are quite competitive to state-of-the-art solver DepQBFSolver [Lonsing and Egly, 2015]; for 3-QSAT problems, complementary
- Also good results for abductive reasoning

QSAT Encodings

QBF $\Phi = \forall x_1, \ldots, x_m \exists y_1, \ldots, y_n (c_1 \wedge \cdots \wedge c_k)$, where $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$

- "classical" encoding:

$$
\begin{array}{ll}
 & var(x_i). \text{ for all } x_i \\
 & var(y_j). \quad exists(y_j). \quad \text{for all } y_j \\
 & occ_h(c_i, z, 0). \text{ s.t. } \ell_{i,h} = \neg z, \qquad occ_h(c_i, z, 1). \text{ s.t. } \ell_{i,h} = z \\
(1) & ass(X, 1) \vee ass(X, 0) \leftarrow var(X). \\
(2) & ass(Y, 0) \leftarrow sat, exists(Y). \\
(3) & ass(Y, 1) \leftarrow sat, exists(Y). \\
(4) & sat \leftarrow (occ_i(C, X_i, A_i), ass(X_i, 1 - A_i))_{i=1}^{3}. \\
(5) & \leftarrow \text{not } sat.
\end{array}
$$

  - Informally, (1) guesses an assignment for all vars
  - by (5), we must derive sat
  - (2), (3), (4): if some assignment to the y_j violates a clause, do saturate

  Note: some answer set exists iff Φ evaluates to false

- Maratea *et al.* [2008] used a propositional version of the classic encoding; dlv was competitive with state-of-the-art QBF solvers then

QSAT Encodings /2

QBF $\Phi = \forall x_1, \ldots, x_m \exists y_1, \ldots, y_n (c_1 \wedge \cdots \wedge c_k)$, where $c_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$

- Bounded arity encoding

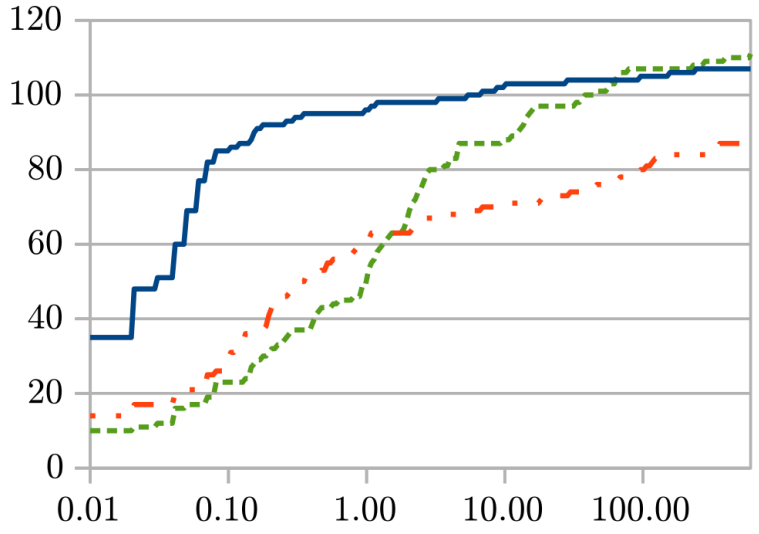
(1)	$t(x_i) \vee f(x_i) \leftarrow \ .$	for all x_i
(2)	$c_i(\vec{t}) \leftarrow t(x_j).$	for all $\vec{t} \in \vec{Y}(c_i),\ x_j \in \{\ell_{i,1}, \ell_{i,2}, \ell_{i,3}\}$
(3)	$c_i(\vec{t}) \leftarrow f(x_j).$	for all $\vec{t} \in \vec{Y}(c_i),\ \neg x_j \in \{\ell_{i,1}, \ell_{i,2}, \ell_{i,3}\}$
(4)	$c_i(\vec{t}).\ \ \vec{t} \in \vec{Y}(c_i) \setminus \{\bar{c}_i\}$	
(5)	$\leftarrow c_1(\eta(c_1)), \ldots, c_k(\eta(c_k)).$	

  - $\vec{Y}(c_i) = \{0,1\}^{h_i}$ are all assignments of 0 and 1 to the literals $\ell_{i,j_1}, \ldots, \ell_{i,j_{h_i}}$ over $y_1, \ldots, y_n$ in c_i ($h_i \leq 3$)
  - $\bar{c}_i$ is the single such assignment that makes them false
  - $\eta(c_i)$ results from $\ell_{i,j_1}, \ldots, \ell_{i,j_{h_i}}$ by replacing each y_j and $\neg y_j$ with Y_j

    Example: for $c = \neg y_1 \vee x_2 \vee y_3$, we have $\ell_{i,j_1}, \ldots, \ell_{i,j_{h_i}}$ = $\neg y_1, y_3$ and thus $\vec{Y}(c) = \{0,1\}^2$, $\bar{c} = (1,0)$, and $\eta(c) = Y_1, Y_3$
  - (1) can be replaced with "shifting" $t(x_i) \leftarrow \text{not } f(x_i).$ and $f(x_i) \leftarrow \text{not } t(x_i).$

- Lifted to QBFs $\Phi = \exists x_1, \ldots, x_l \forall x_{l+1}, \ldots, x_m \exists y_1, \ldots, y_n (c_1 \wedge \cdots \wedge c_k)$

Evaluation Results



x = time (secs), y = # solved instances

(1) classic encoding: dotted-dasher red; (2) long rule encoding:green; (3) DepQBFSolver: blue

- Publicly available 2-QBF ($\forall\exists$) competition instances
- For (1), (2), preprocessing by DepQBFSolver, (for (2), decomposition), then clingo
- For (3), DepQBFSolver
- 200 instances, 2 secs: (2) solved 111, (1) 88, and (3) 107
- Specific benchmark “stmt”: (3) $>$ (2) $>$ (1)
- 3-QBF: 151 inst. from the Eval-2012 data set of latest QBF competition
  - (3) solved 47; (2) solved ony 18, but 10 not solved by (3)

References I

Mario Alviano and Wolfgang Faber.
Dynamic magic sets and super-coherent answer set programs.
AI Commun., 24(2):125–145, 2011.

Mario Alviano, Wolfgang Faber, Gianluigi Greco, and Nicola Leone.
Magic sets for disjunctive datalog programs.
Artif. Intell., 187:156–192, 2012.

Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone, and Francesco Ricca.
WASP: A native ASP solver based on constraint learning.
In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, volume 8148 of *Lecture Notes in Computer Science*, pages 54–66. Springer, 2013.

Vernon Asuncion, Fangzhen Lin, Yan Zhang, and Yi Zhou.
Ordered completion for first-order logic programs on finite structures.
Artif. Intell., 177-179:1–24, 2012.

Michael Bartholomew and Joohyung Lee.
System aspmt2smt: Computing ASPMT theories by SMT solvers.
In Eduardo Fermé and João Leite, editors, *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 529–542. Springer, 2014.

References II

C. Bell, A. Nerode, R. Ng, and V.S. Subrahmanian.
Mixed Integer Programming Methods for Computing Non-Monotonic Deductive Databases.
Journal of the ACM, 41(6):1178–1215, November 1994.

Manuel Bichler, Michael Morak, and Stefan Woltran.
The power of non-ground rules in answer set programming.
CoRR, abs/1608.01856, 2016.
ICLP 2016, to appear.

Piero A. Bonatti, Enrico Pontelli, and Tran Cao Son.
Credulous resolution for answer set programming.
In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 418–423. AAAI Press, 2008.

Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski.
Answer set optimization.
In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 867–872. Morgan Kaufmann, 2003.

References III

Gerhard Brewka, James P. Delgrande, Javier Romero, and Torsten Schaub.
asprin: Customizing answer set preferences without a headache.
In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 1467–1474. AAAI Press, 2015.

Rémi Brochenin, Yuliya Lierler, and Marco Maratea.
Abstract disjunctive answer set solvers.
In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, pages 165–170, 2014.

Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone.
Computable functions in asp: Theory and implementation.
In *International Conference on Logic Programming (ICLP'08)*, volume 5366 of *LNCS*, pages 407–424. Springer, 2008.

Yin Chen, Fangzhen Lin, Yisong Wang, and Mingyi Zhang.
First-order loop formulas for normal logic programs.
In Patrick Doherty, John Mylopoulos, and Christopher Welty, editors, *Proceedings Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), June 2-5, Lake District of the UK*, pages 298–307. AAAI Press, 2006.

References IV

Keith L. Clark.
Negation as failure.
In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

Minh Dao-Tran, Thomas Eiter, Michael Fink, Gerald Weidinger, and Antonius Weinzierl.
OMiGA: An open minded grounding on-the-fly answer-set solver.
In Luis Fariñas del Cerro, Andreas Herzig, and Jérôme Mengin, editors, *Proceedings 13th European Conference on Logics in Artificial Intelligence (JELIA 2012)*, number 7519 in LNCS, pages 480–483. Springer, 2012.

Broes de Cat, Marc Denecker, and Peter J. Stuckey.
Lazy model expansion by incremental grounding.
In Dovier and Costa [2012], pages 201–211.

James P. Delgrande, Torsten Schaub, and Hans Tompits.
A compiler for ordered logic programs.
CoRR, cs.AI/0003024, 2000.

James P. Delgrande, Torsten Schaub, and Hans Tompits.
A framework for compiling preferences in logic programs.
TPLP, 3(2):129–187, 2003.

References V

James P. Delgrande, Torsten Schaub, Hans Tompits, and Kewen Wang.
A classification and survey of preference handling approaches in nonmonotonic reasoning.
Computational Intelligence, 20(2):308–334, 2004.

Agostino Dovier and Vítor Santos Costa, editors.
Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary, volume 17 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

Thomas Eiter, Georg Gottlob, and Nicola Leone.
Abduction From Logic Programs: Semantics and Complexity.
Theoretical Computer Science, 189(1-2):129–177, December 1997.

Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer.
Computing preferred answer sets by meta-interpretation in answer set programming.
Theory and Practice of Logic Programming, 3(4-5):463–498, 2003.

T. Eiter, W. Faber, M. Fink, and S. Woltran.
Complexity Results for Answer Set Programming with Bounded Predicate Arities.
Annals of Mathematics and Artificial Intelligence, 51(2-4):123–165, 2007.

Thomas Eiter, Wolfgang Faber, Michael Fink, and Stefan Woltran.
Complexity results for answer set programming with bounded predicate arities and implications.
Annals of Mathematics and Artificial Intelligence, 51(2-4):123–165, 2007.

References VI

Thomas Eiter, Wolfgang Faber, and Mushthofa Mushthofa.
Space efficient evaluation of asp programs with bounded predicate arities.
In *Proceedings 24th Conference on Artificial Intelligence (AAAI '10), July 11-15, 2010, Atlanta*, pages 303–308. AAAI Press, 2010.

Wolfgang Faber, Nicola Leone, and Gerald Pfeifer.
Experimenting with heuristics for answer set programming.
In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 635–640. Morgan Kaufmann, 2001.

Martin Gebser, Torsten Schaub, and Sven Thiele.
Gringo : A new grounder for answer set programming.
In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, volume 4483 of *Lecture Notes in Computer Science*, pages 266–271. Springer, 2007.

References VII

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, Marius Thomas Schneider, and Stefan Ziller.
A portfolio solver for answer set programming: Preliminary report.
In James P. Delgrande and Wolfgang Faber, editors, *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, volume 6645 of *Lecture Notes in Computer Science*, pages 352–357. Springer, 2011.

Martin Gebser, Benjamin Kaufmann, and Torsten Schaub.
Conflict-driven answer set solving: From theory to practice.
Artificial Intelligence, 187–188:52–89, August 2012.

Martin Gebser, Tomi Janhunen, and Jussi Rintanen.
Answer set programming as SAT modulo acyclicity.
In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 351–356. IOS Press, 2014.

Enrico Giunchiglia, Yuliya Lierler, and Marco Maratea.
Answer set programming based on propositional satisfiability.
J. Autom. Reasoning, 36(4):345–377, 2006.

References VIII

Sergio Greco, Francesca Spezzano, and Irina Trubitsyna.
On the termination of logic programs with function symbols.
In Dovier and Costa [2012], pages 323–333.

Christoph Koch, Nicola Leone, and Gerald Pfeifer.
Enhancing Disjunctive Logic Programming Systems by SAT Checkers.
Artificial Intelligence, 15(1–2):177–212, December 2003.

Joohyung Lee and Vladimir Lifschitz.
Loop Formulas for Disjunctive Logic Programs.
In *Proceedings of the Nineteenth International Conference on Logic Programming (ICLP-03)*, pages 451–465. Springer Verlag, December 2003.

Joohyung Lee.
A model-theoretic counterpart of loop formulas.
In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, pages 503–508. Professional Book Center, 2005.

Claire Lefèvre, Christopher Béatrix, Igor Stéphan, and Laurent Garcia.
Asperix, a first order forward chaining approach for answer set computing.
CoRR, abs/1503.07717, 2015.

N. Leone, P. Rullo, and F. Scarcello.
Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation.
Information and Computation, 135:69–112, 1997.

References IX

Nicola Leone, Pasquale Rullo, and Francesco Scarcello.
Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation.
Information and Computation, 135(2):69–112, June 1997.

Yuliya Lierler and Vladimir Lifschitz.
One more decidable class of finitely ground programs.
In *25th International Conference on Logic Programming (ICLP'09)*, volume 5649 of *LNCS*, pages 489–493. Springer, 2009.

Yuliya Lierler.
cmodels - SAT-Based Disjunctive Answer Set Solver.
In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2005)*, volume 3662 of *Lecture Notes in Computer Science*, pages 447–451. Springer, 2005.

Vladimir Lifschitz and Alexander A. Razborov.
Why are there so many loop formulas?
ACM Trans. Comput. Log., 7(2):261–268, 2006.

V. Lifschitz and H. Turner.
Splitting a Logic Program.
In *Proceedings ICLP-94*, pages 23–38, Santa Margherita Ligure, Italy, June 1994. MIT-Press.

References X

Fangzhen Lin and Yuting Zhao.
ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers.
In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, Edmonton, Alberta, Canada, 2002. AAAI Press / MIT Press.

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

Guohua Liu, Tomi Janhunen, and Ilkka Niemelä.
Answer set programming via mixed integer programming.
In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*. AAAI Press, 2012.

Florian Lonsing and Uwe Egly.
Depqbf: An incremental QBF solver based on clause groups.
CoRR, abs/1502.02484, 2015.

Marco Manna, Francesco Ricca, and Giorgio Terracina.
Taming primary key violations to query large inconsistent data via ASP.
TPLP, 15(4-5):696–710, 2015.

References XI

Marco Maratea, Francesco Ricca, Wolfgang Faber, and Nicola Leone.
Look-back techniques and heuristics in dlv: Implementation, evaluation, and comparison to qbf solvers.
J. Algorithms, 63(1-3):70–89, 2008.

Marco Maratea, Luca Pulina, and Francesco Ricca.
A multi-engine approach to answer-set programming.
TPLP, 14(6):841–868, 2014.

Michael Morak and Stefan Woltran.
Preprocessing of complex non-ground rules in answer set programming.
In Dovier and Costa [2012], pages 247–258.

Ilkka Niemelä.
Integrating answer set programming and satisfiability modulo theories.
In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*, volume 5753 of *Lecture Notes in Computer Science*, page 3. Springer, 2009.

Alessandro Dal Palù, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi.
Gasp: Answer set programming with lazy grounding.
Fundamenta Informaticae, 96(3):297–322, 2009.

References XII

Simona Perri, Francesco Scarcello, Gelsomina Catalano, and Nicola Leone.
Enhancing DLV instantiator by backjumping techniques.
Annals of Mathematics and Artificial Intelligence, 51(2-4):195–228, 2008.

Simona Perri, Francesco Ricca, and Marco Sirianni.
Parallel instantiation of ASP programs: techniques and experiments.
TPLP, 13(2):253–278, 2013.

Enrico Pontelli, Marcello Balduccini, and F. Bermudez.
Non-monotonic reasoning on beowulf platforms.
In Verónica Dahl and Philip Wadler, editors, *Practical Aspects of Declarative Languages, 5th International Symposium, PADL 2003, New Orleans, LA, USA, January 13-14, 2003, Proceedings*, volume 2562 of *Lecture Notes in Computer Science*, pages 37–57. Springer, 2003.

Chiaki Sakama and Katsumi Inoue.
Prioritized Logic Programming and its Application to Commonsense Reasoning.
Artificial Intelligence, 123(1-2):185–222, 2000.

João P. Marques Silva, Inês Lynce, and Sharad Malik.
Conflict-driven clause learning SAT solvers.
In *Handbook of Satisfiability*, pages 131–153. 2009.

References XIII

Tran Cao Son and Enrico Pontelli.
Planning with preferences using logic programming.
TPLP, 6(5):559–607, 2006.

Tommi Syrjänen.
Omega-restricted logic programs.
In Thomas Eiter, Wolfgang Faber, and Miroslaw Truszczynski, editors, *Proc. Sixth International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 2173 of *Lecture Notes in Computer Science*, pages 267–279. Springer Verlag, 2001.