OMiGA: An Open Minded Grounding On-The-Fly Answer Set Solver

Minh Dao-Tran, Thomas Eiter, Michael Fink,
Gerald Weidinger, and Antonius Weinzierl*

Institute of Information Systems, Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
{dao,eiter,fink,weidinger,weinzierl}@kr.tuwien.ac.at

Abstract. We present a new solver for Answer-Set Programs whose main features include grounding on-the-fly and readiness for use in solving distributed answer-set programs. The solver is implemented in Java and uses an underlying Rete network for propagation. Initial experimental results show the benefit of using Rete for this purpose, but also exhibit the need for learning in the presence of grounding on-the-fly.

1 Motivation

Answer-set programming (ASP), based on [2], is a paradigm where a problem is encoded as a program that is a set P of nonmonotonic rules, facts, and constraints. The program P is then given to a solver that searches for specific interpretations called answer sets of P such that all rules and constraints are satisfied.

Example 1. Consider a variant of cut-set: given a graph, remove one edge and compute the reachability of the modified graph. An ASP encoding is $P =$

$$\left\{ \begin{array}{ll} r_1\colon \mathit{del}(X,Y) \leftarrow e(X,Y), \text{not } k(X,Y). & r_4\colon \mathit{reach}(X,Y) \leftarrow k(X,Y). \\ r_2\colon \quad k(X,Y) \leftarrow e(X,Y), \mathit{del}(X_1,Y_1), X_1 \neq X. & r_5\colon \mathit{reach}(X,Z) \leftarrow \mathit{reach}(X,Y), \\ r_3\colon \quad k(X,Y) \leftarrow e(X,Y), \mathit{del}(X_1,Y_1), Y_1 \neq Y. & \qquad\qquad\qquad\qquad \mathit{reach}(Y,Z). \end{array} \right\}.$$

Rule r_1 uses negation to create a choice point for each edge; r_2 and r_3 ensure that only one edge is removed. Finally, r_4 and r_5 compute reachability among the edges kept. Observe that only r_1 requires a guess and if for one ground instance the rule is guessed applicable, an answer set can be found using only propagation.

For humans, abstract rules as above are easy to understand and process, while traditional answer-set solvers can not handle them. They apply (pre-)grounding, i.e., substituting all variables with the actual values they might take. So for each combination of edges, a new rule is generated where the variables are eliminated.

As the ground program might be very large compared to the non-ground program, techniques like intelligent grounding have been developed in order to restrict the rules which must be grounded. But for distributed systems, these techniques conflict with

* This work has been funded by the Vienna Science and Technology Fund (WWTF) project ICT 08-020 and by the Austrian Science Fund (FWF) project P20841.

information hiding and the fact that not all information is known when pre-grounding must take place. As open domains become more important, the same problem arises even in non-distributed settings.

To avoid pre-grounding the answer set solvers GASP [5] and ASPeRiX [3,4] have been developed, both based on the same evaluation technique which has similarities to the SModels [6] approach. We reconsider this technique since it can be improved with well-suited data structures like Rete [1], and we develop a solver, called OMiGA, with an eye on distributed systems.

2 Methods and Techniques

Intuitively, OMiGA keeps a partial interpretation $I = (I^+, I^-)$ containing all ground atoms considered true (resp., false) in I^+ (resp., I^-). If there is a non-ground rule r with a valuation V of variables such that all positive atoms of the grounded rule $r[V]$ are true, i.e., $B^+(r[V]) \subseteq I^+$, and all negative atoms of $r[V]$ are false, i.e., $B^-(r[V]) \subseteq I^-$, then the head of r is grounded and derived, i.e., I^+ is extended by $H(r[V])$. This *propagation* is repeated until a fixpoint is reached.

If propagation is finished and a valuation V of a rule r exists such that $B^+(r[V]) \subseteq I^+$ and $B^-(r[V]) \cap I^+ = \emptyset$, then $r[V]$ possibly is applicable in some answer set; we have to *guess* whether it is the case or not. If yes, I^- is extended by $B^-(r[V])$ and I^+ by $H(r[V])$, making $r[V]$ applicable. If not, conceptually a constraint is added ensuring that $r[V]$ does not become applicable.

At any point, if a constraint is violated or $I^+ \cap I^- \neq \emptyset$, the current guessing branch is inconsistent, backtracking is issued, and subsequently a guess is inverted.

To minimize time needed to find a valuation V of r for propagation or guessing, we employ a Rete-like network, where nodes represent parts of the input program P. Each node has an associated working memory (WM) in which all valuations that are true under I are stored. There are *basic nodes*, *join nodes*, and *head nodes*, representing facts, subsets of rule body atoms, and rule heads, respectively.

These nodes are connected according to P, e.g., for rule r_5 in Example 1 there exists one join-node connected twice to the basic node of *reach*, and it is connected to the head node for r_5. If a new fact is added to its corresponding basic node, it is propagated through the network and thus only processed where it might lead to new results. To represent I^+ and I^- there are two basic nodes for each predicate p. For guessing, the Rete network is built such that each rule has a specific join-node whose WM contains valuations V for which $B(r[V]) \subseteq I^+$, i.e., finding a rule whose applicability can be guessed amounts to selecting from that node.

3 Solver Architecture

The basic architecture of OMiGA is shown in Figure 1. Its input is an answer-set program that is parsed and rewritten to optimize guessing (in a standard way, introducing new atoms representing satisfaction of rule bodies). From these rules, the Rete Builder then creates the Rete network that is later used in the central component of our solver. Here, a Manager controls the depth-first search for answer sets as follows: Rete is repeatedly

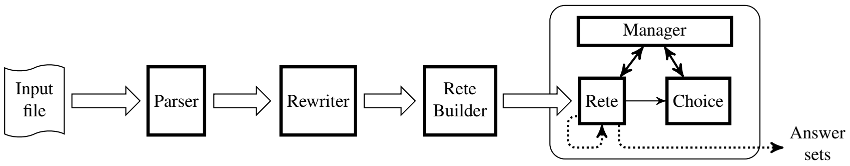
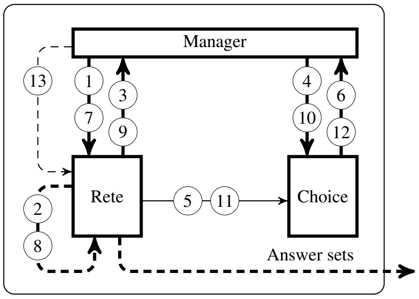


Fig. 1. System architecture



$(1, 7)$	M (Manager) triggers R (Rete) to propagate
(2)	R propagates until a fix point and informs M (3) $I^{+} = \{e(1,2), e(2,3), e(3,4), e(2,4)\}$
$(4, 10)$	M triggers C (Choice) to make a choice
(5)	C looks up Rete network, finds a choice $r_1[X/2, Y/3]$
(6)	C makes a choice: r_1 is applicable $I^{+} = I^{+} \cup \{del(2,3)\}, I^{-} = \{k(2,3)\}$
(8)	R propagates until a fix point and informs M (9) $I^{+} = I^{+} \cup \left\{ \begin{array}{l} k(1,2), k(3,4), k(2,4), reach(1,4), \\ reach(1,2), reach(3,4), reach(2,4) \end{array} \right\}$
(11)	C looks up Rete network, finds no more choice
(12)	C informs M of no more choice
(13)	M requests R to print the answer

Fig. 2. Illustration of the propagation-and-guess evaluation

triggered to evaluate all rules and derive new rule heads. If no new atoms can be derived, i.e., the propagation phase is finished, then the Choice component selects a rule whose positive body is fulfilled, a new decision level is entered and the rule is guessed to be applicable. This process of propagation and guess is repeated until either no more choices can be made, i.e., an answer set is found and printed, or an inconsistent state is reached. In the latter case, backtracking is done and the last guess is inverted. Figure 2 details a run for Example 1, on a graph with edges $E = \{e(1,2), e(2,3), e(3,4), e(2,4)\}$.

4 Evaluation

We compare OMiGA to clingo[1] 3.0.4, DLV[2] 2011-12-21, and ASPeRiX 0.2.4; we omit GASP, as it is known to be slower than ASPeRiX [4] and time measurement due to entanglement with Prolog is ambiguous. The instances are: (i) reachability, a positive program for graph reachability (from the 2009 ASP Contest), with close to $24K$ and $700K$ edges, (ii) 3-colorability on sparse graphs of size 10 and 20, (iii) locstrat, a benchmark program from [4] with 200 and 400 nodes, (iv) cutedge, our running example on a random graph with 100 nodes and close to $2.8K$ resp. $4.9K$ edges.

As Table 1 shows, the use of Rete pays off as it stores partial joins and hence reduces time for propagation. OMiGA is consistently faster than ASPeRiX, except for locstrat where ASPeRiX uses must-be-true propagation while OMiGA only propagates rule heads. Also note that due to the use of Java and the building of the Rete network, our solver has an increased startup-time. On NP-hard problems like 3-colorability, the issue

[1] http://potassco.sourceforge.net/
[2] http://www.dlvsystem.com

Table 1. Comparative systems evaluation (**c**: clingo, **d**: DLV, **a**: ASPeRiX, **o**: OMiGA)

	reachability		3-colorability				locstrat				cutedge			
	24K	**700K**	**10**		**20**		**200**		**400**		**2.8K**		**4.9K**	
c	0.33	5.00	0.00	0.00	0.00	0.00	0.46	0.46	2.06	2.05	25.85	27.34	75.06	79.26
d	0.44	4.56	0.00	0.00	0.00	0.00	5.88	5.67	46.93	47.78	107.07	214.67	301.54	600.08
a	2.84	—	0.01	1.06	—	—	0.01	0.08	0.07	0.33	1.70	16.70	4.62	46.02
o	1.20	15.53	0.16	0.35	1.97	5.37	0.38	0.65	0.61	1.32	0.77	3.05	0.85	3.53

Running time in seconds, left: first answer, right: first 10 answers (if applicable).

of missing learning from conflicts is visible as clingo (using nogood learning) and DLV (using backjumping and look-back heuristics) perform extremely well. For non-ground ASP solving, learning is an open issue.

Comparing the cutedge benchmark with reachability, the effect of intelligent pre-grounding is evident since for the positive reachability instances, the pre-grounder can efficiently evaluate the programs. If those pre-grounding strategies are not possible due to only one guessing rule like in cutedge, the propagation becomes ineffective, very much in contrast to a Rete-based propagation.

5 Ongoing Work and Conclusion

We have presented OMiGA,[3] a new grounding on-the-fly solver for answer set programs, which also can be employed in a distributed setting where nodes contain programs that can access atoms at other nodes, by exchanging only the Manager component. As our experimental results show, Rete pays off and makes OMiGA outperform clingo and DLV for propagation-intense instances. Future work on backtracking, conflict-driven learning, and extended propagation is planned.

References

1. Forgy, C.L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19(1), 17–37 (1982)
2. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3/4), 365–386 (1991)
3. Lefèvre, C., Nicolas, P.: A First Order Forward Chaining Approach for Answer Set Computing. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 196–208. Springer, Heidelberg (2009)
4. Lefèvre, C., Nicolas, P.: The First Version of a New ASP Solver: ASPeRiX. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 522–527. Springer, Heidelberg (2009)
5. Palù, A.D., Dovier, A., Pontelli, E., Rossi, G.: Gasp: Answer set programming with lazy grounding. Fundam. Inform. 96(3), 297–322 (2009)
6. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artif. Intell. 138(1-2), 181–234 (2002)

[3] `http://www.kr.tuwien.ac.at/research/systems/omiga`