NoMoRe: Non-Monotonic Reasoning with Logic Programs

Christian Anger, Kathrin Konczak, and Thomas Linke

Universität Potsdam, Institut für Informatik,
{canger,konczak,linke}@cs.uni-potsdam.de

1 Introduction

The **no**n-**mo**notonic **re**asoning system NoMoRe [2] implements answer set semantics for normal logic programs. It realizes a novel, rule-based paradigm to compute answer sets by computing non-standard graph colorings of the *block graph* associated with a given logic program (see [6, 8, 7] for details). These non-standard graph colorings are called *a-colorings* or *application-colorings* since they reflect the set of generating rules (applied rules) for an answer set. Hence NoMoRe is rule-based and not atom-based like most of the other known systems. In the recent release of NoMoRe we have newly implemented *backward propagation* of partial a-colorings and a technique called *jumping* in order to ensure full (backward) propagation [8]. Both techniques improve the search space pruning of NoMoRe. Furthermore, we have extended the syntax by integrity, weight and cardinality constraints [11, 5][1].

The NoMoRe-system is implemented in the programming language Prolog; it has been developed under the ECLiPSe Constraint Logic Programming System [1] and it was also successfully tested with SWI-Prolog [12]. The system is available at http://www.cs.uni-potsdam.de/~linke/nomore. In order to use the system, ECLiPSe- or SWI-Prolog is needed [1, 12][2].

2 Theoretical Background

The current release of the NoMoRe system implements nonmonotonic reasoning with normal logic programs under answer set semantics [4]. We consider rules r of the form $p \leftarrow q_1, \ldots, q_n, not\ \ s_1, \ldots, not\ \ s_k$ where p, q_i $(0 \leq i \leq n)$ and s_j $(0 \leq j \leq k)$ are atoms, $head(r) = p$, $body^+(r) = \{q_1, \ldots, q_n\}$, $body^-(r) = \{s_1, \ldots, s_k\}$ and $body(r) = body^+(r) \cup body^-(r)$.

The block graph of program P is a directed graph on the rules of P:

Definition 1. *([6]) Let P be a logic program and let $P' \subseteq P$ be maximal grounded[3]. The block graph $\Gamma_P = (V_P, A_P^0 \cup A_P^1)$ of P is a directed graph with vertices $V_P = P'$ and two*

[1] Observe, that we use a transformation for weight and cardinality constraints into normal logic programs, which is exponential in the worst case. That is, so far we did not consider efficiency issues when dealing with these constraints.

[2] Both Prolog systems are freely available for scientific use.

[3] A set of rules S is *grounded* iff there exists an enumeration $\langle r_i \rangle_{i \in I}$ of S such that for all $i \in I$ we have that $body^+(r_i) \subseteq head(\{r_1, \cdots, r_{i-1}\})$. A maximal grounded set P' is a grounded set that is maximal wrt set inclusion. We generalize the definition of the head of a rule to sets of rules in the usual way.

different kinds of arcs

$$A_P^0 = \{(r', r) \mid r', r \in P' \text{ and } head(r') \in body^+(r)\}$$
$$A_P^1 = \{(r', r) \mid r', r \in P' \text{ and } head(r') \in body^-(r)\}.$$

Since groundedness (by definition) ignores negative bodies, there exists a unique maximal grounded set $P' \subseteq P$ for each program P, that is, Γ_P is well-defined. Definition 1 captures the conditions under which a rule r' blocks another rule r (i.e. $(r', r) \in A^1$). We also gather all groundedness information in Γ_P, due to the restriction to rules in the maximal grounded part of P. This is essential because a block relation between two rules r' and r becomes effective only if r' is groundable through other rules. Therefore Γ_P captures all information necessary for computing the answer sets of program P.

Answer sets are characterized as special non-standard graph colorings of block graphs. Let P be a logic program and let Γ_P the corresponding block graph. For $r \in P$ let $G_r = (V_r, A_r)$ be a directed graph. Then G_r is a *grounded 0-graph for r in Γ_P* iff the following conditions hold: 1. G_r is an acyclic subgraph of Γ_P s.t. $A_r \subseteq A_P^0$, 2. $r \notin V_r$ and $body^+(r) \subseteq head(V_r)$ and 3. for each node $r' \in V_r$ and for each $q' \in body^+(r')$ there exists a node $r'' \in V_r$ s.t. $q' = head(r'')$ and $(r'', r') \in A_r$. The different grounded 0-graphs for rule r in Γ_P correspond to the different classical "proofs" for $head(r)$ in P^+, ignoring the default negations of all rules. Now let $\mathcal{C} : P \mapsto \{\ominus, \oplus\}$ be a total mapping[4]. We call r *grounded wrt Γ_P and $\mathcal{C}$* iff there exist a grounded 0-graph $G_r = (V_r, A_r)$ for r in Γ_P s.t. $\mathcal{C}(V_r) = \oplus$. r is called *blocked wrt Γ_P and $\mathcal{C}$* if there exists some $r' \in \text{Pred1}(r)$ s.t. $\mathcal{C}(r') = \oplus$. Now we are ready to define a-colorings.

Definition 2. *Let P be a logic program, let Γ_P be the corresponding block graph and let $\mathcal{C} : P \mapsto \{\ominus, \oplus\}$ be a total mapping. Then $\mathcal{C}$ is an* a-coloring *of Γ_P iff the following condition holds for each $r \in P$*

AP *$\mathcal{C}(r) = \oplus$ iff r is grounded wrt Γ_P and $\mathcal{C}$ and r is not blocked wrt Γ_P and $\mathcal{C}$.*

Observe, that there are programs (e.g. $P = \{p \leftarrow not\ p\}$) s.t. no a-coloring exists for Γ_P. Intuitively, each node of the block graph (corresponding to some rule) is colored with one of two colors, representing application ($\oplus$) or non-application ($\ominus$) of the corresponding rule. In [6] we show that there is a one-to-one correspondence between a-colorings and answer sets. That is, the set of all $\oplus$-colored rules wrt some a-coloring are the generating rules for some answer set of the corresponding program.

3 Description of the System

`NoMoRe` uses a compilation technique to compute answer sets of a logic program P in three steps (see Figure 1). At first, the block graph Γ_P is computed. Secondly, Γ_P is compiled into Prolog code in order to obtain an efficient coloring procedure. User may choose between two different kinds of compilation, one which is fast but which gives a lot of compiled code and another one which is a little bit slower but which produces less compiled code than the

[4] A mapping $\mathcal{C} : P \mapsto C$ is called *total* iff for each node $r \in P$ there exists some $\mathcal{C}(r) \in C$. Oppositely, mapping $\mathcal{C}$ is called partial if there are some $r \in P$ for which $\mathcal{C}(r)$ is undefined.

other. The second way of compiling has to be used with large logic programs, depending on the memory management of the underlying Prolog system. The compiled Prolog code (together with the example-independent code) is then used to actually compute the answer sets. To read logic programs we use a parser (eventually after running a grounder, e.g. `lparse` or `dlv`) and there is a separate part for interpretation of a-colorings into answer sets. Additionally, `noMoRe` comes with an interface to the graph drawing tool `DaVinci` [10] for visualization of block graphs. This allows for a structural analysis of programs.

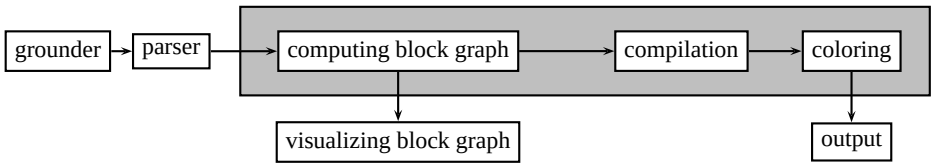

Fig. 1. The architecture of `noMoRe`.

The `noMoRe` system is used for purposes of research on the underlying paradigm. But even in this early state, usability for anybody familiar with the logic programming paradigm is given. The syntax accepted by `noMoRe` is Prolog-like. For example, a rule $a \leftarrow b, not\ c$ is represented through `a:-b,not c`. Furthermore, `noMoRe` is able to deal with integrity constraints as well as weight and cardinality constraints, which are some of the most interesting syntactical features of `smodels`.

All discussed features (e.g. backward propagation, jumping, `daVinci` output and some experimental features) can be controlled by corresponding flags (see the `noMoRe` manual [9]).

4 Evaluating the System

As benchmarks, we used some instances of NP-complete problems proposed in [3], namely, the independent set problem for circle graphs[5], the problem of finding Hamiltonian cycles in complete graphs and the problem of finding classical graph colorings. Furthermore we have tested some instances of the n-queens problem. In Table 1 we have time measurements for `noMoRe`, `smodels` and `dlv` (from left to right). For results considering the number of choices needed to compute all solution for similar examples see [8]. For Hamiltonian and n-queens problems we clearly see an improvement when using backward propagation and jumping. For independent set and coloring problems `noMoRe` becomes slower, but uses less choices than without backward propagation and jumping (see [8]). For coloring problems a similar behavior can be observed for the `smodels` system.

[5] A so-called circle graph Cir_n has n nodes $\{v_1, \cdots, v_n\}$ and arcs $A = \{(v_i, v_{i+1}) \mid 1 \leq i < n\} \cup \{(v_n, v_1)\}$.

	noMoRe			smodels		dlv
backprop	no	yes	yes	with (without)		
jumping	no	no	yes	lookahead		
ham_k_7	17.6	3.25	3.92	0.38	(0.16)	0.1
ham_k_8	328.86	35.86	44.23	7.69	(3.57)	0.71
ind_cir_40	5.84	7.18	7.24	1.31	(0.77)	1.97
ind_cir_50	106.15	127.45	128.99	23.87	(16.81)	40.62
col4x4	1.92	2.51	1.96	0.1	(0.27)	0.36
col5x5	179.38	223.42	180.35	11.54	(41.17)	24.79
queens7	0.73	0.84	0.5	0.08	(0.03)	0.05
queens8	3.82	4.3	1.86	0.32	(0.09)	0.12

Table 1. Time measurements in seconds for different problems (all solutions) on AMD 1.4 GHz.

References

1. A. Aggoun, D. Chan, P. Dufresne, et al. Eclipse user manual release 5.0. Available at http://www.icparc.ic.ac.uk/eclipse, 2000.
2. C. Anger, K. Konczak, and T. Linke. NoMoRe: A system for non-monotonic reasoning under answer set semantics. In W. Faber T. Eiter and M. Truszczyński, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, pages 406–410. Springer, 2001.
3. P. Cholewiński, V. Marek, A. Mikitiuk, and M. Truszczyński. Experimenting with nonmonotonic reasoning. In *Proceedings of the International Conference on Logic Programming*, pages 267–281. MIT Press, 1995.
4. M. Gelfond and V. Lifschitz. Classical negation in logic programs and deductive databases. *New Generation Computing*, 9:365–385, 1991.
5. P. Simons I. Niemel and T. Soininen. Stable model semantics of weight constraint rules. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, volume 1730 of *Lecture Notes in Artificial Intelligence*, pages 177–191, El Paso, Texas, USA, 1999. Springer Verlag.
6. T. Linke. Graph theoretical characterization and computation of answer sets. In B. Nebel, editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 641–645. Morgan Kaufmann Publishers, 2001.
7. T. Linke. Rule-based computation of answer sets. 2002. submitted.
8. T. Linke, C. Anger, and K. Konczak. More on nomore. In S. Benferhat and E. Giunchiglia, editors, *Proceedings of the 9th International Workshop on Nonmonotonic Reasoning*, pages 210–218, Toulouse, 2002.
9. T. Linke, K. Konzcak, and C. Anger. http//www.cs.uni-potsdam.de/~linke/nomore, 2000–2002.
10. M.Werner. davinci v2.1.x online documentation. daVinci is available at http://www.tzi.de/ davinci/doc_V2.1/, University of Bremen, 1998.
11. P. Simons. Extending the stable model semantics with more expressive rules. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, volume 1730 of *Lecture Notes in Artificial Intelligence*, pages 177–191, El Paso, Texas, USA, 1999. Springer Verlag.
12. J. Wielemaker. Swi-prolog 3.4.3 reference manual. SWI-Prolog is available at http://www.swi.psy.uva.nl/projects/SWI-Prolog/Manual/, 1990–2000.