

# NoMoRe: A System for Non-Monotonic Reasoning under Answer Set Semantics

Christian Anger, Kathrin Konczak, and Thomas Linke

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

## 1 Introduction

NoMoRe implements answer set semantics for normal logic programs. It realizes a novel paradigm to compute answer sets by computing *a-colorings* (non-standard graph colorings with two colors) of the *block graph* (a labeled digraph) associated with a given program  $P$  (see [5] for details). Intuitively, an *a-coloring* reflects the set of generating rules for an answer set, which means that noMoRe is rule-based and not atom-based like most of the other known systems. Since the core system was designed for propositional programs only, we have integrated `lparse` [8] as a grounder in order to deal with variables. Furthermore, we have included an interface to the graph drawing tool `DaVinci` [6] for visualization of block graphs. This allows for a structural analysis of programs.

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 [9]. 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 [1,9]<sup>1</sup>.

## 2 Theoretical Background

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

Look at the following normal logic program

$$P = \{a \leftarrow b, \text{not } e. \quad b \leftarrow d. \quad c \leftarrow b. \quad d \leftarrow . \quad e \leftarrow d, \text{not } f. \quad f \leftarrow a.\} \quad (1)$$

Let us call the rules of program (1)  $r_a, r_b, r_c, r_d, r_e$ , and  $r_f$ , respectively.  $P$  has the answer sets  $A_1 = \{d, b, c, a, f\}$  and  $A_2 = \{d, b, c, e\}$ . It is easy to see that the application of  $r_f$  blocks the application of  $r_e$  wrt  $A_1$ , because if  $r_f$  contributes to  $A_1$ , then  $f \in A_1$  and thus  $r_e$  cannot be applied. Analogously,  $r_e$  blocks  $r_a$  wrt answer set  $A_2$ . This observation leads us to a strictly blockage-based approach.

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

---

<sup>1</sup> Both Prolog systems are freely available for scientific use.

**Definition 1.** ([5]) Let  $P$  be a logic program and let  $P' \subseteq P$  be maximal grounded<sup>2</sup>. 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 different kinds of arcs

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

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

Figure 1 shows the block graph of program (1). Since groundedness (by definition) ignores negative bodies, there exists a unique maximal grounded set  $P' \subseteq P$  for each program  $P$ , that is,  $\Gamma_P$  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  $\Gamma_P$ , 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  $\Gamma_P$  captures all information necessary for computing the answer sets of program  $P$ .

Answer sets then are characterized as special non-standard graph colorings of block graphs. We denote 0-predecessors, 0-successors, 1-predecessors and 1-successors of  $\Gamma_P$  by  $\gamma_0^-(v)$ ,  $\gamma_0^+(v)$ ,  $\gamma_1^-(v)$  and  $\gamma_1^+(v)$  for  $v \in V$ , respectively.

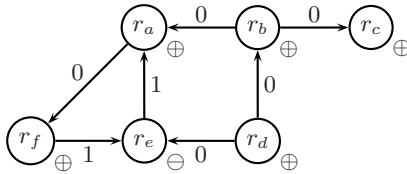
**Definition 2.** ([5]) Let  $P$  be a logic program, s.t.  $|\text{body}^+(r)| \leq 1$  for each  $r \in P$ , let  $\Gamma_P = (P, A_P^0 \cup A_P^1)$  be the corresponding block graph and let  $c : P \mapsto \{\ominus, \oplus\}$  be a mapping. Then  $c$  is an a-coloring (application-coloring) of  $\Gamma_P$  iff the following conditions hold for each  $r \in P$ <sup>3</sup>

- A1**  $c(r) = \ominus$  iff one of the following conditions holds
  - a.  $\gamma_0^-(r) \neq \emptyset$  and for each  $r' \in \gamma_0^-(r)$  we have  $c(r') = \ominus$
  - b. there is some  $r'' \in \gamma_1^-(r)$  s.t.  $c(r'') = \oplus$ .
- A2**  $c(r) = \oplus$  iff both of the following conditions hold
  - a.  $\gamma_0^-(r) = \emptyset$  or it exists grounded 0-path  $G_r$  s.t.  $c(G_r) = \oplus$
  - b. for each  $r'' \in \gamma_1^-(r)$  we have  $c(r'') = \ominus$ .

Observe, that there are programs (e.g.  $P = \{p \leftarrow \text{not } p\}$ ) s.t. no a-coloring exists for  $\Gamma_P$ . 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. The coloring presented in Figure 1 corresponds to answer set  $A_1$  of  $P$ . Node (rule)  $r_e$  has to be colored  $\ominus$  (not applied), because there is some 1-predecessor of  $r_e$  colored  $\oplus$  (applied). In other words,  $r_f$  blocks  $r_e$ .

<sup>2</sup> 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  $\text{body}^+(r_i) \subseteq \text{head}(\{r_1, \dots, r_{i-1}\})$ . 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.

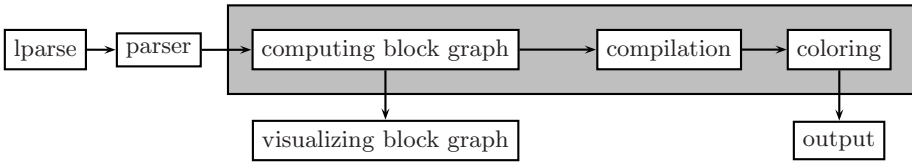
<sup>3</sup> A subset of rules  $G_r \subseteq P$  is a grounded 0-path for  $r \in P$  if  $G_r$  is a 0-path from some fact to  $r$  in  $\Gamma_P$ . For a set of rules  $S \subseteq P$  we write  $c(S) = \oplus$  or  $c(S) = \ominus$  if for each  $r \in S$  we have  $c(r) = \oplus$  or  $c(r) = \ominus$ , respectively. For the generalization of condition  $|\text{body}^+(r)| \leq 1$  see [5]. There you can also find further details on a-colorings and the algorithm to compute them.



**Fig. 1.** Block graph of program (1 with a-coloring corresponding to answer set  $A_1$ )

### 3 Description of the System

NoMoRe uses a compilation technique to compute answer sets of a logic program  $P$  in three steps (see Figure 2). At first, the block graph  $\Gamma_P$  is computed. Secondly,  $\Gamma_P$  is compiled into Prolog code in order to obtain an efficient coloring procedure. The compiled Prolog code is then used to actually compute the answer sets. To read logic programs we use a parser (eventually after running `lparse`) and there is a separate part for interpretation of a-colorings into answer sets. For information purpose there is yet another part for visualizing block graphs using the graph drawing tool DaVinci [6]. The noMoRe system is used for purposes



**Fig. 2.** The architecture of noMoRe

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, the first rule of program (1) is represented through `a :- b, not e.`

### 4 Evaluating the System

As a first benchmark, we used two NP-complete problems proposed in [2]: the problem of finding a Hamiltonian path in a graph (**Ham**) and the independent set problem (**Ind**). In terms of time used for computing answer sets, our first prolog implementation is not comparable with state of the art C/C++ implementations, e.g. `smodels` [7] and `dlv` [3]. Therefore we compare the number of used choice points, because it reflects how an algorithm deals with the exponential part of a problem. Unfortunately, only `smodels` gives information about its

**Table 1.** Number of choice points for **HAM**-problems of complete graphs with  $n$  nodes

$n =$	all solutions for <b>Ham</b> of $K_n$				one solution for <b>Ham</b> of $K_n$							
	7	8	9	10	5	6	7	8	9	10	11	12
<b>smodels</b>	4800	86364	1864470	45168575	3	4	30	8	48	1107	18118	398306
<b>noMoRe</b>	14335	115826	1160533	7864853	16	20	31	34	58	69	79	108

choice points. For this reason, we have concentrated on comparing our approach with **smodels**.

Results are given for finding all solutions of different instances of **HAM** and **Ind**. Table 1 shows results for some **HAM**-encodings of complete graphs  $K_n$  where  $n$  is the number of nodes<sup>4</sup>. Surprisingly, it turns out that **noMoRe** performs very good on this problem class. That is, with growing problem size we need less choice points (and less time) than **smodels**. This can also be seen in Table 2 which shows the corresponding time measurements. To be fair, for **Ind**-problems of graphs  $Cir_n$ <sup>5</sup> we need more choice points (and much more time) **smodels** needs. However, even with the same number of choice points **smodels** is faster than **noMoRe**, because **noMoRe** uses general backtracking of prolog, whereas **smodels** backtracking is highly specialized for computing answer sets. The same applies to **d1v**. Even so, it is clear that our approach is a very promising one.

**Table 2.** Time measurements in seconds with ECLIPSe Prolog for **HAM**- and **IND**-problems on a SUN Ultra2 with two 300MHz Sparc processors (compilation time not included)

$n =$	<b>Ham</b> of $K_n$										<b>Ind</b> of $Cir_n$			
	all solutions			one solution							all solutions			
	8	9	10	5	6	7	8	9	10	11	12	40	50	60
<b>smodels</b>	54	1334	38550	0.01	0.02	0.04	0.04	0.11	1.61	24	526	8	219	4052
<b>d1v</b>	4	50	493	0.02	0.03	0.03	0.05	0.06	0.07	0.09	0.15	13	259	4594
<b>noMoRe</b>	208	2556	21586	0.01	0.02	0.06	0.12	0.25	0.40	0.53	1.02	39	706	12767

## References

1. A. Aggoun, D. Chan, P. Dufresne, and other. Eclipse user manual release 5.0. Available at <http://www.icparc.ic.ac.uk/eclipse>, 2000. 406
2. 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. 408

<sup>4</sup> In a complete graph each node is connected to each other node.

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

3. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for nonmonotonic reasoning. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the Fourth International Conference on Logic Programming and Non-Monotonic Reasoning*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 363–374. Springer Verlag, 1997. 408
4. M. Gelfond and V. Lifschitz. Classical negation in logic programs and deductive databases. *New Generation Computing*, 9:365–385, 1991. 406
5. Th. Linke. Graph theoretical characterization and computation of answer sets. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2001. to appear. 406, 407
6. M. Werner. *davinci v2.1.x* online documentation. *daVinci* is available at [http://www.tzi.de/davinci/doc\\_V2.1/](http://www.tzi.de/davinci/doc_V2.1/), University of Bremen, 1998. 406, 408
7. I. Niemelä and P. Simons. *Smodels*: An implementation of the stable model and well-founded semantics for normal logic programs. In J. Dix, U. Furbach, and A. Nerode, editors, *Proc. of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 420–429. Springer, 1997. 408
8. T. Syrjänen. *Lparse 1.0 user's manual*. Available at <http://saturn.tcs.hut.fi/Software/smodels/>, 2000. 406
9. Jan 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. 406