More on noMoRe

Thomas Linke  Christian Anger  Kathrin Konczak
Universität Potsdam, Institut für Informatik
August-Bebel-Strasse 89, D-14482 Potsdam
{linke,anger,konczak}@cs.uni-potsdam.de

Abstract

We focus on the efficient computation of answer sets for normal logic programs. We concentrate on a recently proposed rule-based method (implemented in the noMoRe system) for computing answer sets. We show how noMoRe and its underlying method can be improved tremendously by improving its deterministic consequences. With these improvements noMoRe is able to deal with problem classes it could not handle so far.

1 Introduction

Answer set programming (ASP) is a programming paradigm, which allows for solving problems in a compact and highly declarative way. The basic idea is to specify a given problem in a declarative language, e.g. normal logic programs\footnote{The language of normal logic programs is not the only one suitable for ASP. Others are disjunctive logic programs, propositional logic or DATALOG with constraints [4].}, such that the different answer sets given by answer sets semantics [8] correspond to the different solutions of the initial problem [10]. As an example, consider the independent set problem, which is to determine if there exists a maximal (wrt set inclusion) independent subsets of nodes for a given graph. A subset \( S \subseteq V \) of nodes of a graph \( G = (V,E) \) is called independent if there are no edges between nodes in \( S \). Let

\[
P = \begin{cases}
in(a) \iff \neg in(d), \neg in(b) \\
\neg in(b), \neg in(a), \neg in(c) \\
\neg in(c), \neg in(b), \neg in(d) \\
\neg in(d), \neg in(c), \neg in(a)
\end{cases}
\]

(1)

be a logic program and let us call the rules \( r_a, r_b, r_c \) and \( r_d \), respectively. Then program (1) encodes the independent set problem for graph \( G = \{(a,b,c,d),\{(a,b),(b,c),(c,d),(d,a)\}\} \). Program (1) has two answer sets \( X_1 = \{\text{in}(a),\text{in}(c)\} \) and \( X_2 = \{\text{in}(b),\text{in}(d)\} \) corresponding to the two independent sets of graph \( G \).

Currently there are reasonably efficient implementations (e.g. smodels [15] and dlv [5]) available as well as interesting applications of answer set programming, (e.g. [3, 13, 14]). Since computation of answer sets is NP-complete for normal logic programs (and \( \Sigma^p_2 \)-complete for disjuctive logic programs), most algorithms rely on methods similar to the Davis-Putnam algorithm for SAT. That is, they contain a non-deterministic part (making choices) and a part computing deterministic consequences for these choices. Whereas in [7] different heuristics are investigated in order to make the “right” choices, in this paper we improve the deterministic consequences of the recently proposed rule-based noMoRe system [1]. For all ASP systems relying on Davis-Putnam-like algorithms (that is, for all systems mentioned so far), non-deterministic choices and deterministic consequences determine the behavior of the resulting algorithm. In particular, we redefine propagation of so-called a-colorings as introduced in [11] such that we are able to include backward propagation. Furthermore, we introduce a technique called jumping to ensure complete backward propagation and give experimental results showing the influence of the presented techniques.

2 Background

We deal with normal logic programs which contain the symbol \( \neg \) used for negation as failure. A normal logic program is a set of rules of the form \( p \leftarrow q_1, \ldots, q_m, \neg s_1, \ldots, \neg s_k \) where \( p, q_i \) (0 \( \leq i \leq m \)) and \( s_j \) (0 \( \leq j \leq k \)) are propositional atoms. A rule is a fact if \( n = k = 0 \), it is called basic if \( k = 0 \) and quasi-fact if \( n = 0 \). For a rule \( r \) like above we define head\((r) = p \) and body\((r) = \{q_1, \ldots, q_m, \neg s_1, \ldots, \neg s_k\} \). Fur-
thermore, let \( \text{body}^+(r) = \{q_1, \ldots, q_n\} \) denote the set of positive body atoms and \( \text{body}^-(r) = \{s_1, \ldots, s_l\} \) the set of negative body atoms. We denote the set of all facts of a program \( P \) by \( \text{Facts}(P) \) and the set of all atoms of \( P \) by \( \text{Atoms}(P) \).

Let \( r \) be a rule, \( r^+ \) then denotes the rule \( \text{head}(r) \leftarrow \text{body}^+(r) \), obtained from \( r \) by deleting all negative body atoms in the body of \( r \). For a logic program \( P \) let \( P^+ = \{r^+ \mid r \in P\} \). A set of atoms \( X \) is closed under a basic program \( P \) iff for any \( r \in P \), \( \text{head}(r) \in X \) whenever \( \text{body}^+(r) \subseteq X \). The smallest set of atoms which is closed under a basic program \( P \) is denoted by \( \text{Cn}(P) \). The retract, \( P^X \), of a program \( P \) relative to a set \( X \) of atoms is defined by \( P^X = \{r^+ \mid r \in P \text{ and } \text{body}^+(r) \cap X = \emptyset \} \). We say that a set \( X \) of atoms is an answer set of a program \( P \) iff \( \text{Cn}(P^X) = X \).

A set of rules \( P \) is grounded iff there exists an enumeration \( (r_i)_{i \in I} \) of \( P \) such that for all \( i \in I \) we have that \( \text{body}^+(r_i) \subseteq \text{head}(\{r_i, \ldots, r_{i-1}\}) \). Observe that there exists a unique maximal grounded set \( P' \subseteq P \) for each program \( P \). For a set of rules \( P \) and a set of atoms \( X \) we define the set of generating rules of \( P \) wrt \( X \) as \( GR(P, X) = \{r \in P \mid \text{body}^+(r) \subseteq X, \text{body}^-(r) \cap X = \emptyset \} \). Then \( X \) is an answer set of \( P \) iff \( X = \text{Cn}(GR(P, X)) \). This characterizes answer sets in terms of generating rules. Observe, that in general \( GR(P, X)^+ \neq P^X \) (take \( P = \{a \leftarrow b, b \leftarrow c\} \) and \( X = \{a\} \)).

We need some graph theoretical terminology. A directed graph (or digraph) \( G \) is a pair \( G = (V, A) \) such that \( V \) is a finite, non-empty set (vertices) and \( A \subseteq V \times V \) is a set (arcs). For a digraph \( G = (V, A) \) and a vertex \( v \in V \), we define the set of all predecessors of \( v \) as \( \text{Pred}(v) = \{u \mid (u, v) \in A\} \). Analogously, the set of all successors of \( v \) is defined as \( \text{Succ}(v) = \{u \mid (v, u) \in A\} \). A path from \( v \) to \( v' \) in \( G = (V, A) \) is a finite subset \( P_{vv'} \subseteq V \) such that \( P_{vv'} = \{v_1, \ldots, v_n\} \), \( v = v_1 \), \( v' = v_n \) and \( (v_i, v_{i+1}) \in A \) for each \( 1 \leq i < n \). Let \( G = (V, A) \) and \( G' = (V', A') \) be digraphs. Then \( G' \) is a subgraph of \( G \) if \( V' \subseteq V \) and \( A' \subseteq A \). \( G' \) is an induced subgraph of \( G \) if \( G' \) is a subgraph of \( G \) s.t. for each \( v, v' \in V' \) we have that \( (v, v') \in A' \) iff \( (v, v') \in A \).

In order to represent more information in a directed graph, we need a special kind of arc labeling. \( G = (V, A^0 \cup A^1) \) is a directed graph whose arcs \( A^0 \cup A^1 \) are labeled zero \( (A^0) \) and one \( (A^1) \). We call arcs in \( A^0 \) and \( A^1 \) 0-arcs and 1-arcs, respectively. For \( G \) we distinguish 0-successors (0-successors) from 1-predecessors (1-successors) denoted by \( \text{Pred}0(v) (\text{Succ}0(v)) \) and \( \text{Pred}1(v) (\text{Succ}1(v)) \) for \( v \in V \), respectively. A path \( P_{vv'} \) in \( G \) is called 0-path if \( \text{Arcs}(P_{vv'}) \subseteq A^0 \).

**Block Graphs for Normal Logic Programs**

Next we summarize the central definitions of block graphs for logic programs and a-colorings of block graphs (cf. [11, 12]).

**Definition 1** Let \( P \) be a logic program and let \( P' \) be the maximal grounded subset of \( P \). The block graph \( \Gamma_P = (V_P, A^0_P \cup A^1_P) \) of \( P \) is a directed graph with vertices \( V_P = P \) and two different kinds of arcs defined as follows

\[
A^0_P = \{(r', r) \mid r', r \in P' \text{ and } \text{head}(r') \in \text{body}^+(r)\},
A^1_P = \{(r', r) \mid r', r \in P' \text{ and } \text{head}(r') \in \text{body}^-(r)\}.
\]

This definition captures the conditions under which a rule \( r' \) blocks another rule \( r \) (i.e. \( (r', r) \in A^1 \)). We introduce an 1-arc \((r', r)\) in \( \Gamma_P \) if \( r' = (q \leftarrow \ldots) \) and \( r = (\ldots \leftarrow q, \ldots) \). We also gather all groundedness information in \( \Gamma_P \), because we only introduce a 0-arc \((r', r)\) (between rules \( r' = (q \leftarrow \ldots) \) and \( r = (\ldots \leftarrow q, \ldots) \)) if \( r' \) and \( r \) are in the maximal grounded subset of \( P \). Figure 1 shows the block graph of program (1). Observe, that operations \( \text{head}(r), \text{body}^+(r) \) and \( \text{body}^-(r) \) (for \( r \in P \)) are operations on the block graph, since the nodes of \( \Gamma_P \) are the rules of logic program \( P \).

In order to define so-called application colorings or a-colorings for block graphs we need the following definition.

**Definition 2** Let \( P \) be a logic program and let \( \Gamma_P = (V_P, A^0_P \cup A^1_P) \) the corresponding block graph. Furthermore, let \( x \in \text{Atoms}(P) \) and let \( G_x = (V_x, A_x) \) be a subgraph of \( \Gamma_P \). Then \( G_x \) is a grounded 0-graph for \( x \) in \( \Gamma_P \) iff the following conditions hold:

1. \( G_x \) is an acyclic subgraph of \( \Gamma_P \) s.t. \( A_x \subseteq A^0_P \)
2. \( G_x \) contains a target node \( r_x \) s.t. \( x = \text{head}(r_x) \) and from every other node there exists a 0-path to the target node
3. for each node \( r \in V_x \) we have \( \text{body}^+(r) = \emptyset \) or for each \( q' \in \text{body}^+(r) \) there exists a node \( r' \in V_x \) s.t. \( q' = \text{head}(r') \) and \((r', r) \in A_x \).

\(^2\)Here we mean sets \( P' \) which are maximal wrt set inclusion and grounded.

\(^3\)Observe, that for program \( P = \{p \leftarrow q, q \leftarrow p\} \) the maximal grounded subset of rules is empty and therefore \( \Gamma_P \) contains no 0-arcs.
Observe, that the nodes of a grounded 0-graph are grounded according to definition (see Section 2). Furthermore, the different grounded 0-graphs for atom \( x \) in \( \Gamma_P \) correspond to the different classical “proofs” for \( x \) in \( P^+ \), ignoring the default negations of all rules.

**Definition 3** Let \( P \) be a logic program, let \( C : P \mapsto \{\oplus, \ominus\} \) be a total mapping\(^4\). We call \( r \) grounded wrt \( \Gamma_P \) and \( C \) iff for each \( q \in body^+(r) \) there exist a grounded 0-graph \( G_q = (V_q, A_q) \) for \( q \) in \( \Gamma_P \) s.t. \( C(V_q) = \oplus \).

A rule \( r \) is called blocked wrt \( \Gamma_P \) and \( C \) if there exists some \( r' \in \text{Pred}(r) \) s.t. \( C(r') = \ominus \).

Now we are ready to define a-colorings.

**Definition 4** Let \( P \) be a logic program, let \( \Gamma_P \) be the corresponding block graph and let \( C : P \mapsto \{\oplus, \ominus\} \) be a total mapping. Then \( C \) is an a-coloring of \( \Gamma_P \) iff the following condition hold for each \( r \in P \)

\[
\text{AP} \quad C(r) = \oplus \iff r \text{ is grounded and } r \text{ is not blocked wrt } \Gamma_P \text{ and } C.
\]

Let \( C \) be an a-coloring of some block graph \( \Gamma_P \). Rules are then intuitively applied wrt some answer set of \( P \) if and only if they are colored \( \oplus \), that is, condition AP captures the intuition of applying a rule wrt to some answer set. Similarly, the negation of condition AP (\( r \) is not grounded or \( r \) is blocked) captures the intuition when a rule is not applicable wrt to some answer set.

Observe, that there are programs for which the corresponding block graph has no a-coloring and thus no answer set. Let \( r_p \) be rule \( p \leftarrow \neg \neg p \). Then program \( P = \{r_p\} \) has block graph \((P, \emptyset, \{(r_p, r_p)\})\), that is, \( \Gamma_P \) consists of a single 1-loop. By Definition 4 there is no a-coloring of \( \Gamma_P \). If we color \( r_p \) with \( \oplus \) we get a direct contradiction to AP, since then \( r_p \) is blocked. On the other hand, if we color \( r_p \) with \( \ominus \) then \( r_p \) is trivially grounded and not blocked. Therefore \( r_p \) has to be colored \( \oplus \) which again is a contradiction.

The main result in [11] states that Program \( P \) has an answer set \( X \) iff \( \Gamma_P \) has an a-coloring \( C \) s.t.

\[
GR(P, X) = \{r \in P \mid C(r) = \oplus\}.
\]

This result constitutes a rule-based method to compute answer sets by computing a-colorings. In Figure 1 we have depicted the two a-colorings of the block graph of program (1) left and right from \( \oplus \), respectively.

![Figure 1: Block graph and a-colorings of program (1).](image)

---

### 3 Propagation

In the nonmonotonic reasoning system noted in the last section is implemented. Let us assume that each program is grounded. In order to describe the deterministic part of the implementation and its improvements, we need some central properties of nodes. All those properties are defined wrt partial a-colorings. We call a partial mapping \( C : P \mapsto \{\oplus, \ominus\} \) a partial a-coloring if \( C \) is an a-coloring of the induced subgraph of \( \Gamma_P \) with nodes \( \text{Dom}(C) \).

**Definition 5** Let \( P \) be a logic program and let \( C \) be a partial a-coloring of \( \Gamma_P \). For each node \( r \in P \) we define the following properties wrt \( \Gamma_P \) and \( C \):

1. p-grounded(\( r \)) if \( \exists q \in \text{body}^+(\( r \)): \exists r' \in \text{Pred}(\( r \)) : q = \text{head}(\( r' \)) \text{ and } C(\( r' \)) = \oplus \)
2. p-notgrounded(\( r \)) if \( \exists q \in \text{body}^+(\( r \)): \forall r' \in \text{Pred}(\( r \)) : q \neq \text{head}(\( r' \)) \text{ or } C(\( r' \)) = \ominus \)
3. p-blocked(\( r \)) if \( \exists r' \in \text{Pred}(\( r \)) : C(\( r' \)) = \oplus \)
4. p-notblocked(\( r \)) if \( \forall r' \in \text{Pred}(\( r \)) : C(\( r' \)) = \ominus \)

Notice the difference between total and partial a-colorings. For example, if p-notgrounded(\( r \)) holds for \( r \) wrt some total a-coloring \( C \) then p-grounded(\( r \)) is the negation of p-notgrounded(\( r \)). This does not hold for partial a-coloring \( C \), since there may be nodes for which \( C \) is undefined. For this reason, we have to define both p-grounded (p-blocked) and p-notgrounded (p-notblocked), respectively, because they cannot be defined through each other wrt partial a-colorings. However, we have the following result for total a-colorings:

**Theorem 1** Let \( P \) be a logic program and let \( C \) be a total a-coloring of \( \Gamma_P \). Then for each node \( r \in P \) we have \( r \) is grounded wrt \( \Gamma_P \) and \( C \) if p-grounded(\( r \) wrt \( \Gamma_P \) and \( C \).
$\Gamma_P$ and $C$. Furthermore, we have $r$ is blocked wrt $\Gamma_P$ and $C$ iff $p$-blocked$(r)$ wrt $\Gamma_P$ and $C$.

Clearly, to be grounded (wrt $\Gamma_P$ and $C$ for node $r$) is a global concept w提醒 0 $\Gamma_P$ whereas $p$-grounded$(r)$ is defined locally wrt $\Gamma_P$. Furthermore, observe the difference between $r$ is blocked and $p$-blocked$(r)$ wrt $\Gamma_P$ and $C$. Even if the definitions of both concepts are the same (cf Definition 3), the former is defined wrt to total a-colorings, whereas the latter one is defined wrt partial a-colorings. In a situation like in Fig-

$$\begin{array}{c}
\begin{array}{c}
\bullet r' \quad 1 \quad r \\
\end{array}
\end{array}$$

Figure 2: Some block graph with partial a-coloring.

ure 2 we do not have $p$-blocked$(r')$ and we do not have $p$-not-blocked$(r')$ wrt the depicted partial a-coloring. But we always have either that $r$ is blocked or not blocked wrt total a-colorings.

Definition 6 Let $C$ be a partial a-coloring of $\Gamma_P$ and let $U$ be the set of uncolored nodes wrt $C$. Then each node $r \in U$ can be colored $\oplus$ by propagation of $C$ iff we have $p$-grounded$(r)$ and $p$-not-blocked$(r)$ wrt $C$. Node $r$ can be colored $\ominus$ by propagation of $C$ iff we have $p$-not-grounded$(r)$ or $p$-blocked$(r)$ wrt $C$.

Notice, that propagation of partial a-colorings to uncolored nodes is global wrt $\Gamma_P$, since in order to propagate $C$ as much as possible we have to check all nodes in $U$ which in general are distributed over $\Gamma_P$. According to Definitions 5 and 6 nodes colored by propagation always have colored predecessors. Therefore we obtain a more procedural way to propagate partial a-colorings by localized propagation conditions.

Definition 7 Let $P$ be a logic program, let $\Gamma_P$ be the corresponding block graph and let $C$ be a partial a-coloring of $\Gamma_P$. We define an extended mapping $C^e$of $C$ s.t. for each $r \in \text{Dom}(C)$ we have $C^e(r) = C(r)$ and for each $r, r' \in P$ the following conditions hold wrt $\Gamma_P$ and $C^e$:

(A) if $r \in \text{Succ}(r')$ and $C^e(r') = \ominus$ then $C^e(r) = \ominus$

(B) if $r \in \text{Succ}(r')$ and $C^e(r') = \ominus$ and $p$-not-blocked$(r)$ and $p$-grounded$(r)$ then $C^e(r) = \ominus$

(C) if $r \in \text{Succ}(r')$ and $C^e(r') = \ominus$ and $p$-not-blocked$(r)$ and $p$-grounded$(r)$ then $C^e(r) = \ominus$

(D) if $r \in \text{Succ}(r')$ and $C^e(r') = \ominus$ and $p$-not-grounded$(r)$ then $C^e(r) = \ominus$.

We have the following result:

**Theorem 2** Let $P$ be a logic program and let $C$ and $C^e : P \rightarrow \{\ominus, \ominus\}$ be partial mappings. Then we have if $C$ is a partial a-coloring of $\Gamma_P$ and $C^e$ is an extension of $C$ as in Definition 7 then $C^e_C \subseteq C^e_C$, $C^e_C \subseteq C^e_C$ and $C^e$ is a partial a-coloring of $\Gamma_P$.

This theorem gives the conditions for four different propagation cases in arc direction: if a node $r$ is colored $c (c \in \{\ominus, \ominus\})$ then this color can be propagated over 1- and over 0-arcs to the neighbors of $r$, according to localized propagation conditions (A), (B), (C) and (D).

Now let $P$ be some logic program. Let $C : P \rightarrow \{\ominus, \ominus\}$ be a partial a-coloring. $C$ is represented by a pair of (disjoint) sets $(C^e_C, C^e_C)$ s.t. $C^e_C = \{r \in P \mid C(r) = \ominus\}$ and $C^e_C = \{r \in P \mid C(r) = \ominus\}$. Since $C$ is not total we do not necessarily have $P = C^e_C \cup C^e_C$. We refer to mapping $C$ with the pair $(C^e_C, C^e_C)$ and vice versa. Assume that $\Gamma_P$ is a global parameter of each of the presented procedure. Let $U$ and $N$ be sets of nodes s.t. $U$ contains the currently uncolored nodes $U := P \setminus (C^e_C \cup C^e_C)$ and $N$ contains colored nodes whose color has to be propagated. Figures 3, 4 and 5 show the main procedures of nonMore in pseudo code.

```pseudo
function color_P(U, N: list; C: partial mapping)
    var r: node
    if propagate_P(N, C)
        U := U \ {C^e_C \cup C^e_C}
        if choose_P(U, C, r)
            U := U \ {r}
            if color_P(U, \{r\}, (C^e_C \cup \{r\})) then return true
        else return color_P(U, \{r\}, (C^e_C \cup \{r\}, C^e_C))
    else return propagate_P(U, (C^e_C \cup U, C^e_C))
    return false

Figure 3: Basic nonMore procedure.
```

Notice that procedures color_P and propagate_P return some extended partial mapping through parameter C or fail. Procedure choose_P either returns some uncolored node r s.t. we have p-grounded(r) wrt the current partial a-coloring or fails. Clearly, choose_P implements the non-deterministic part of color_P. Oppositely, propagate_P implements the deterministic consequences of nonMore. Let $L_1(P) = \{r \in P \mid r \in \text{Pred}(r)\}$ denote the set of all 1-loops in $\Gamma_P$. When calling color_P the first time, we start with
function propagate\(_p\)(N; list, C; partial mapping)
\[ \text{var } r' : \text{node } t : \text{boolean} \]
\[ t := \text{true} \]
\[ \textbf{while } (N \neq \emptyset \text{ and } t = \text{true}) \text{ do} \]
\[ \text{select } r' \text{ from } N \]
\[ \textbf{if } (r' \in C) \text{ then} \]
\[ (A) \quad t := (\text{propA}_p(r', C) \text{ and} \)
\[ (C) \quad \text{propC}_p(r', C) \]
\[ \textbf{else} \]
\[ (B) \quad t := (\text{propB}_p(r', C) \text{ and} \)
\[ (D) \quad \text{propD}_p(r', C) \]
\[ \text{return } t \]

Figure 4: Procedure \(\text{propagate}_p\).

\(C = (L_1(P), \text{Facts}(P)), U = P \setminus (\text{Facts}(P) \cup L_1(P))\)
and \(N = \text{Facts}(P) \cup L_1(P)\). That is, we start with all facts colored \(\oplus\) and all 1-loops colored \(\ominus\). Basically, \(\text{color}_p\) takes both a partial mapping \(C\) and a set of uncolored nodes \(U\) and aims at coloring these nodes. That is, \(\text{color}_p\) computes an extended partial mapping or if this is impossible it fails. This is done by choosing some uncolored node \(r \ (r \in U)\) with \(\text{choose}_p\) and by trying to color it \(\oplus\) first. If this does not give a solution \(\text{color}_p\) tries to color node \(r\) with \(\ominus\). If both possibilities fail \(\text{color}_p\) fails. Therefore, we say that node \(r\) is used as a choice. To be a choice is not a property of a node, because choices are dynamic wrt each solution. Therefore a node may be a choice in one run of \(\text{color}_p\) but not in every run, e.g., because there may be heuristics that uses different nodes as choices depending on the actual implementation of \(\text{choose}_p\). Observe, that all different a-colorings are obtained via backtracking over choices in \(\text{color}_p\).

Notice, that procedure \(\text{propagate}_p\) works locally according to conditions \((A)\) to \((D)\) of Definition 7. The color of a node is propagated immediately after getting colored, because the test whether the node was colored correctly is done during propagation. Observe that Theorem 4.1 in [12] implies that partial a-coloring \(C\) can not be extended to some total a-coloring if \(\text{propagate}_p\) fails. Therefore, \(\text{color}_p\) fails only during propagation. Procedures \(\text{propA}_p, \text{propB}_p, \text{propC}_p, \text{and} \text{propD}_p\) are the implementations of \((A), (B), (C)\) and \((D)\), respectively. The purpose of \(\text{propagate}_p\) is to apply the corresponding propagation cases, e.g. if \(C(r') = \oplus\) then cases \((A)\) and \((C)\) have to be applied\(^7\). According to Figure 3, partial a-colorings are propagated only in arc direction. That is, if some node is colored then we try to propagate the color of this node to its successors (if possible).

Function \(\text{propB}_p(r' ; \text{node}; C; \text{partial mapping})\)
\[ \text{var } r : \text{node } S : \text{set of nodes } t : \text{boolean} \]
\[ S := \{r \in \text{Succ1}(r') \mid \text{condition (B) holds for } r\} \]
\[ t := \text{true} \]
\[ \textbf{while } (S \neq \emptyset \text{ and } t = \text{true}) \text{ do} \]
\[ \text{select } r \text{ from } S \]
\[ \textbf{if } r \in C \text{ then} \]
\[ \text{return } \text{false} \]
\[ \textbf{else} \]
\[ \textbf{if } r \notin C \text{ then} \]
\[ t := \text{propagate}_p(r, (C,\ominus, C \cup \{r\})) \]
\[ \text{return } \text{false} \]

Figure 5: Procedure \(\text{propB}_p\).

4 Backward Propagation

In [11, 1] it is stated that the number of choices can be reduced by introducing backward propagation, that is, partial a-colorings can also be propagated in opposite arc direction. Clearly, as for propagation in arc direction, we have four backward propagation cases. However, there is a problem with defining localized conditions for backward propagation (as in Definition 7). Assume that Figure 6 depicts a part of some block graph together with some partial a-coloring.

```
1 0 r'
\Phi 1 0 ...
```

Figure 6: Part of some block graph with partial a-coloring.

Graph together with some partial a-coloring. On the one hand, we know that \((r')\) has to be colored \(\oplus\) (provided that there are no other predecessors of \((r')\), because this is the only way to block \((r)\) and if \((r)\) is not blocked there is no answer set. On the other hand, we cannot color \((r)\) with \(\oplus\), because we do not have \(p\)-grounded\((r')\) (see Definition 6). Therefore we need so-called transitory a-colorings.

\textbf{Definition 8} Let \(P\) be some logic program. We call a partial mapping \(C : P \rightarrow \{\ominus, \oplus, +\}\) a transitory a-coloring of \(\Gamma_P\) iff \(C\) is an a-coloring of the induced subgraph of \(\Gamma_P\) with nodes \(C_\ominus \cup C_\oplus\).

\(^7\) The missing three procedures also used in \(\text{propagate}_p\) can be easily implemented analogously to \(\text{propB}_p\).
That is, a transitory a-coloring is a partial a-coloring where some nodes may be uncolored or colored with +. Color + is used instead of ⊕ to color node $r'$ in situations like in Figure 6, where we do not have p-grounded($r'$) wrt the current partial a-coloring, but $r'$ can still possibly be grounded. In order to transform some transitory a-coloring (during the execution of color$_p$) to a total a-coloring color + is replaced by color ⊕, if possible. This is achieved by propagation. Whenever a node is colored, this color is propagated to all its neighbors immediately, no matter whether these already have been colored or not. In case a node already colored + (⊕) has to be colored ⊕ via propagation, propagation fails due to contradiction. When a node already colored + has to be colored ⊕ via propagation, color + is simply replaced by ⊕. That is, either every color + will become ⊕, or color$_p$ fails. We need the following properties wrt transitory a-colorings:

**Definition 9** Let $P$ be a logic program, let $C$ be a transitory a-coloring of $\Gamma_P$ and let $r \in P$ be some node. Then $r$ is groundable($r'$) wrt $\Gamma_P$ and $C$ iff $\forall q \in body^+(r') : \exists r'' \in \text{Pred}(r) \text{ with } q = head(r'') \text{ s.t. } C(r'') = \oplus \text{ or } r'' \text{ is uncolored.}$

Here groundable($r'$) means that either $r$ is groundable or that there is some uncolored 0-predecessor, which can possibly be colored ⊕ while extending $C$. For $r \in P$ and $q \in body^-(r)$ we define $S_q \subseteq \text{Pred}(r)$ as $S_q = \{ r'' | r'' \in \text{Pred}(r) \text{ and } q = head(r'') \}$. Furthermore, for a set of rules $S \subseteq P$ we define p-grounded($S$) wrt $\Gamma_P$ and transitory a-coloring $C$ iff there is some $r \in S$ s.t. $C(r) = \oplus$. Now we are ready to define the four localized backward propagation cases.

**Definition 10** Let $P$ be a logic program, let $\Gamma_P$ be the corresponding block graph and let $C$ be a transitory a-coloring of $\Gamma_P$. We define an extended mapping $C^e : P \rightarrow \{ \oplus, \ominus, + \}$ of $C$ s.t. for each $r \in Domm(C)$ we have $C^e(r) = C(r)$ and conditions (A) to (D) of Definition 7 and the following conditions hold for all $r, r' \in P$ wrt $\Gamma_P$ and $C^e$:

1. **(bA)** if $C^e(r') = \ominus$ and $r \in \text{Pred}(r')$ then $C^e(r) = \ominus$
2. **(bB)** if $C^e(r') = \ominus$ and p-grounded($r'$) and $r \in \text{Pred}(r')$ s.t. and $\forall r'' \in \text{Pred}(r') : (C^e(r'') = \ominus \text{ iff } r'' \neq r)$ then $C^e(r) = +$
3. **(bC)** if $C^e(r') = \ominus$ and there is some $q \in body^+(r')$ s.t. $q = head(r')$ for some $r \in S_q$ and groundable($r'$) and for each $r'' \in S_q : (C^e(r'') = \ominus \text{ iff } r'' \neq r)$ then $C^e(r) = +$
4. **(bD)** if $C^e(r') = \ominus$ and p-notblocked($r'$) and there is some $q \in body^+(r')$ with $q = head(r)$ for some $r \in S_q \text{ s.t. for each } r' \in body^+(r')$: (p-grounded($S_q'$)) if $S_q' \neq S_q$ then $C^e(r) = \ominus$

Intuitively, these cases ensure that an already ⊕-colored node is grounded (bC) and not blocked (bD) while an already ⊕-colored node is blocked (bB) or not grounded (bD). So in a sense, the purpose of these cases is to justify the color of a node. Observe, that cases (bB) and (bC) use color + instead of ⊕ (see Definition 8). We have the following result corresponding to Theorem 2:

**Theorem 3** Let $P$ be a logic program and let $C$ and $C^e : P \rightarrow \{ \oplus, \ominus, + \}$ be a partial mappings. Then we have if $C$ is a transitory a-coloring of $\Gamma_P$ and $C^e$ is an extension of $C$ as in Definition 10 then $C_{\ominus} \subseteq C^e_{\ominus}$, $C_{\oplus} \subseteq C^e_{\oplus}$ and $C^e$ is a transitory a-coloring of $\Gamma_P$.

Let us show how color$_p$ computes the a-colorings of the block graph of program (1) (see Figure 1). At the beginning we cannot propagate anything, because there is no fact and no 1-loop. We take $r_a$ as a choice. First, we try to color $r_a$ with ⊕ by calling color$_p(U, N, C)$ with $U = P \setminus \{ r_a \}$, $N = \{ r_a \}$ and $C = (\emptyset, \{ r_a \})$. Now, propagate$_p(N, C)$ is executed. By propagating $C(r_a) = \ominus$ with case (A) we get $C(r_b) = \ominus$ and $C(r_d) = \ominus$. Recursively, through case (B) $C(r_c) = \ominus$ is propagated. This gives $C = \{ (r_b, r_d), \{ r_a, r_c \} \}$. Since $U$ becomes the empty set, choose$_p$ fails and $C$ is the first output. So far we did not need backward propagation.

Now, we color $r_a$ with ⊕ through calling color$_p(U, N, C)$ with $U$ and $N$ as above and $C = (\emptyset, \{ r_a \})$. Since no (backward) propagation is possible we have to compute the next choice. For choose$_p$ all three uncolored nodes are possible choices s.t. CP holds. Assume $C(r_b) = \oplus$ as next choice. Through propagation case (A) we get $C(r_c) = \ominus$. This color of $r_c$ has to be propagated by executing propagate$_p(\{ r_c \}, \{ r_a, r_b, r_c \} \cup \{ r_b \} \cup \{ r_a \} \cup \{ r_d \})$. By using propagation case (B) we obtain $C(r_d) = \ominus$. Recursively, propagation of ⊕ for $r_d$ gives no contradiction and $C = \{ r_a, r_b, r_d \} \cup \{ r_a, r_d \}$ is the second a-coloring. By assuming $C(r_b) = \oplus$, that is, $C = \{ r_a, r_b, \emptyset \}$, $r_c$ is colored with ⊕ through backward propagation case (bB). By propagation of this color with case (A) node $r_d$ is colored with ⊕. By applying case (B) to the color of $r_d$ we obtain that $r_a$ has to be colored with ⊕, because it is not blocked, but this is a contradiction to $C(p_a) = \ominus$. Thus, there is no further solution and we have found the two solutions with two choices. Observe that the usage of (bB) saves one additional choice, since without backward propagation the partial coloring $C = \{ r_a, r_b \}$ could not have been
extended any more and another choice would have been necessary.

5 Jumping

procedure propagate$_P$(N:list, C:part. mapping)
var r': node;
    while N ≠ ∅ do
        select r' from N;
        if (r' C) then
            (A) if propA$_P$(r', C) fails then fail;
            (C) if propC$_P$(r', C) fails then fail;
            (bA) if backpropA$_P$(r', C) fails then fail;
            (bC) if backpropC$_P$(r', C) fails then fail;
        else
            (B) if propB$_P$(r', C) fails then fail;
            (D) if propD$_P$(r', C) fails then fail;
            (bB) if backpropB$_P$(r', C) fails then fail;
            (bD) if backpropD$_P$(r', C) fails then fail;
            if jumpC$_P$(r', C) fails then fail;
            if jumpD$_P$(r', C) fails then fail;

procedure jumpB$_P$(r': node; C:partial mapping),
var S: set of nodes;
    S := Succ(r');
    while S ≠ ∅ do
        select r' from S;
        if C(r') = ⊥ then backpropB$_P$(r', C);

Figure 7: Extended propagation procedures including backward propagation and jumping.

A further improvement for the rule-based algorithm is so-called jumping. Backward propagation according to (bB), (bC) and (bD) requires certain conditions to be fulfilled, which may not be known when a node is colored. For (bA) this is not the case, because in (bA) there is no further condition. Take the following program P and its corresponding block graph Γ$_P$ (see Figure 8):

\[
P = \begin{cases}
    a \leftarrow \neg a, \neg b, \neg d \\
    b \leftarrow \neg c, \neg d \leftarrow \neg e \\
    c \leftarrow \neg b, e \leftarrow \neg d
\end{cases}
\]

(2)

Figure 8: Block graph of program (2).

We know that C([r$_a$]) = ⊥, otherwise there would not be an answer set at all. Since r$_a$ is trivially grounded, it has to be blocked. This can be achieved by two rules, r$_b$ and r$_d$, though one is sufficient. But we do not know yet, which one should be colored ⊥. Later on, when e.g. r$_b$ has been used as a choice and is colored ⊥, this is achieved via jumping, that is, case (bB) is used again for node r$_a$ and r$_d$ is colored ⊥. Finally, with (A) node r$_e$ is colored ⊥. In this way jumping helps to avoid unnecessary choices, because without jumping we would need another choice to color nodes r$_b$ and r$_a$. Backward propagation would not be complete without jumping. In general, whenever a ⊥ is propagated along a 1-arc to an already ⊥-colored node, we check backward propagation case (bB) for this node again. Similarly, we check (bC) and (bD) again for ⊥-colored and ⊥-colored 0-successors of already colored and propagated nodes, respectively.

As an example, Figure 7 shows the implementation of procedure jumpB$_P$, which jumps to an already colored node in order to check backward propagation (bB) again. By replacing procedure propagate$_P$ in Figure 3 with propagate$_P$ in Figure 7 we obtain an algorithm for computing a-colorings including backward propagation and jumping. The four procedures for the backward propagation cases are implemented similarly to procedure propB$_P$ in Figure 3, with the exception that we aim at coloring predecessors instead of successors now.

6 Results

For a transitory mapping C : P ⊃ {⊥, ⊕, ⊖} we define the set of corresponding answer sets A$_C$ as

\[A_C = \{ X | X \text{ is answer set of } P \text{ and } C_\oplus \subseteq GR(P, X) \text{ and } C_\ominus \cap GR(P, X) = \emptyset \} .\]

If C is undefined for all nodes then A$_C$ contains all answer sets of P. If C is a total mapping s.t. no node is colored with ⊖ then A$_C$ contains exactly one answer set of P (if C is an a-coloring). With this notation we formulate the following result:

Theorem 4 Let P be a logic program, let C and C' : P ⊃ {⊥, ⊕, ⊖} be transitory mappings. Then for each \( r \in (C_\oplus \cup C_\ominus \cup C_\ominus) \) we have if propagate$_P$($\{r\}$, C) succeeds and C' is the transitory mapping after its execution then A$_C$ = A$_{C'}$.

This theorem states that propagate$_P$ neither discards nor introduces answer sets corresponding to some transitory mapping C. Hence, only nodes used as choices lead to different answer sets.

Finally, let C$_P$ be the set of all solutions of color$_P$. We obtain correctness and completeness of color$_P$.  

Theorem 5 Let \( P \) be a logic program, let \( \Gamma_P \) be its block graph, let \( C : P \rightarrow \{\ominus, \oplus\} \) be a mapping and let \( C_P \) the set of all solutions of color\( \_ \) for program \( P \). Then \( C \) is an a-coloring of \( \Gamma_P \) iff \( C \in C_P \).

7 Experiments

As benchmarks, we have used some instances of NP-complete problems proposed in [2], namely, the independent set problem for circle graphs\(^8\), the problem of finding Hamiltonian cycles in complete graphs and the problem of finding classical graph colorings. Furthermore we have tested some planning problems taken from [6] and the n-queens problem. In Table 1 we have counted the number of choices instead of measuring time, since the number of choices indicates how good an algorithm deals with a non-deterministic problem. For smodels results with and without lookahead (results in parenthesis) are shown\(^9\).

<table>
<thead>
<tr>
<th></th>
<th>noMoRe</th>
<th>smodels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>ham</td>
<td>11335</td>
<td>11335</td>
</tr>
<tr>
<td>lam, 8</td>
<td>82200</td>
<td>82200</td>
</tr>
<tr>
<td>ind, circ20</td>
<td>539</td>
<td>317</td>
</tr>
<tr>
<td>ind, circ30</td>
<td>9266</td>
<td>5264</td>
</tr>
<tr>
<td>p2, step4</td>
<td>-</td>
<td>464</td>
</tr>
<tr>
<td>p2, step6</td>
<td>-</td>
<td>13654</td>
</tr>
<tr>
<td>col, 4</td>
<td>27680</td>
<td>27680</td>
</tr>
<tr>
<td>col, 5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>queens, 4</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td>queens, 5</td>
<td>326</td>
<td>326</td>
</tr>
</tbody>
</table>

Table 1: Number of choices (all solutions) for different problems.

The influence of backward propagation and jumping on the number of choices is clearly visible. There are also some problems where we did not obtain a solution after more than 12 hours without backward propagation. Table 1 impressively shows that noMoRe with backward propagation and jumping is now comparable with smodels on several problem classes; especially if we disable the lookahead of smodels. The difference between smodels and noMoRe for planning examples and the n-queens problems seems to come from different heuristics for making choices. We have just started to investigate the influence of more elaborated heuristics for making choices.

8 Conclusion

We have shown that by introducing backward propagation together with jumping the rule-based algorithm implemented in noMoRe can be greatly improved. A related method of backward propagation wrt answer set semantics for normal logic programs was proposed in [9]. However, a lot of the obtained improvement is due to the concept of a third color \(+\). There seems to be a close relation between noMoRe’s color \(+\) and \texttt{dlv}'s must-be-true truth value [6], though this has to be studied more thoroughly, because noMoRe is rule-based and \texttt{dlv} (and smodels) is literal-based. Through the conducted experiments the impact of the improvements is shown. NoMoRe is now comparable to smodels on many different problem classes measuring the number of choices. This improvement was obtained by improving the deterministic consequences of noMoRe. However, there are still some interesting open questions. The main one is whether rule-based computation of answer sets is different from atom-based (literal-based) or not. During our experiments we have detected programs (with a large rule atom ratio) for which atom-based computations are more suitable and other programs (with a small rule atom ratio) for which rule-based computation performs better. Currently, we have no general answer to this question and a general comparison between atom-based and rules-based methods for logic programs will be necessary.

Another question for future work is whether rule-based approaches like implemented in noMoRe can lead to a system which is able to compete (considering time) against state of the art systems like smodels and \texttt{dlv}. Clearly, this question cannot be answered without having a system for the rule-based approach that is as elaborated implemented as smodels and \texttt{dlv} are. Right now for noMoRe this is not the case, because its development is in a state where smodels and \texttt{dlv} were some years ago, for example the heuristics of noMoRe still does not have a lookahead. In fact, the only very simple heuristic we use so far in noMoRe is to make choices for the color of a node \( r \) only if we have \texttt{p-grounded}(\( r \)). However, we think that our experiments show that there is a chance for rule-based methods to be able to compete against atom-based in the future. Therefore further work will also include the study of heuristics and different implementation techniques.
9 Acknowledgements

Acknowledgements

I would like to thank T. Schaub, Ph. Besnard, K. Wang, K. Konczak, Ch. Anger and S.M. Model for commenting on previous versions of this paper.

This work was partially supported by the German Science Foundation (DFG) within Project “Nichtmonotone Inferenzsysteme zur Verarbeitung konfigierender Regeln”.

References


