Using Nested Logic Programs for Answer Set Programming

Thomas Linke

Institut für Informatik, Universität Potsdam
linke@cs.uni-potsdam.de

Abstract. We present a general method to improve computation of answer sets by analyzing structural properties of normal logic programs. Therefore we use labeled directed graphs associated to normal programs, which can be utilized to compute answer sets. The basic idea is to detect special subgraphs of those graphs corresponding to structural properties of normal programs and transform them into simpler but equivalent subgraphs by applying graph transformations. It turns out that there is no characterization for these graph transformations in terms of normal logic programs. Surprisingly, nested logic programs provide a semantics for the investigated transformations. In order to demonstrate its practical usefulness, we have implemented our approach in the `noMoRe` system.

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. logic programs[1], such that the different answer sets given by answer sets semantics [11] correspond to the different solutions of the initial problem [14]. Currently there are reasonably efficient implementations (e.g. `smodels` [18] and `dlv` [6] available. However, it is still important to look for better ASP algorithms and improvements of existing ASP systems.

In this paper we present a general method to improve computation of answer sets for propositional normal logic programs (nLP) by analyzing structural properties of programs[2]. For example, if program P contains the two rules $(a \leftarrow c, \mathit{not}\ d)$ and $(b \leftarrow c, \mathit{not}\ d)$ then both rules always apply together in the construction of an answer set of P. Thus, there is no need to separately check their applicability. The aim of this paper is to detect similar situations where sets of rules behave uniformly wrt all answer sets of a given program. In particular, we utilize block graphs, which can be used to compute answer sets for normal logic programs by computing non-standard graph colorings, called *a-colorings* [16, 17].

[1] The language of logic programs is not the only one suitable for ASP. Others are propositional logic or DATALOG with constraints [5].

[2] The extension to the case where strong negation is permitted is straightforward and proceeds in the usual way.

The central contribution of this work is the definition of different a-coloring preserving block graph transformations, which reduce the number of nodes. By those simplifications of block graphs, we are able to demonstrate that a-colorings and corresponding answer sets can be computed more efficiently. Furthermore, we are able to show that there exist **no** corresponding program transformations between **normal** logic programs for the defined graph transformations. Hence block graphs are demonstrated to be more expressive than normal logic programs. However, it turns out that there are corresponding program transformations between **normal nested** logic programs (nNLP), which provide a semantics for the block graph transformations. In fact block graph transformations and program transformations between normal nested programs are shown to be equivalent. This result has two important consequences. First, it shows how nested logic programs can used for improving answer set computation of normal logic programs. And second, answer sets of normal nested logic programs can be computed by computing answer sets of normal programs.

We have implemented the presented transformations in the `noMoRe` system. Our experimental results show that there is a reasonable gain of efficiency (less time and less space). Observe that our implementation also computes answer sets of normal nested programs by using the above mentioned equivalence between graph and program transformations.

2 Background

In this paper we consider a proper subclass of propositional nested logic programs [14]. Nested logic programs generalize logic programs by allowing bodies and heads of rules to contain arbitrary nested expressions. Here an *expression* is formed from propositional atoms using the operators

$$, \quad ; \quad not$$

standing for conjunction, disjunction and default negation, respectively. *Literals* are expressions of the form p (positive literals) or $not\ p$ (negative literals), where p is some propositional atom. A *rule* r has the form

$$h_1, \ldots, h_k \leftarrow B_1; \ldots; B_n \tag{1}$$

where $h_1, \ldots, h_k$ are atoms and $B_1, \ldots, B_n$ are conjunctions of literals or $\top$ (true) or $\bot$ (false). A rule r is called a *fact* if $n = 1$ and $B_1 = \top$; r is called *normal* if $k \leq 1$ and $n = 1$. If r contains no default negation *not* at all r is a *basic* rule. A *normal nested logic program* is a finite set of rules of the form (1). A *normal logic program* is a finite set of normal rules. A program is *basic* if it contains only basic rules. Let $nNLP$ (nLP) denote the class of all normal nested logic programs (normal logic programs), respectively. For a rule r we define the *head* and the *body* of r as $Head(r) = \{h_1, \ldots, h_k\}$ and $Body(r) = \{B_1, \ldots, B_n\}$, respectively. For a set of rules P we define $Head(P) = \{Head(r) \mid r \in P\}$ and $Body(P) = \{Body(r) \mid r \in P\}$. If $B \in Body(P)$ s.t. $B = (p_1, \ldots, p_l, not\, s_1, \ldots, not\, s_m)$ we let

$B^+ = \{p_1, \ldots, p_l\}$ and $B^- = \{s_1, \ldots, s_m\}$ denote the positive and negative part of B, respectively. If $B = \top$ then we set $B^+ = B^- = \emptyset$. For a normal rule r we define $B_r^+ = B^+$ and $B_r^- = B^-$ where $r = \mathit{Head}(r) \leftarrow B$. Furthermore, $\mathit{Fact}(P)$ denotes the set of all facts of P and $\mathit{Atom}(P)$ denotes the set of all atoms of P.

Answer sets for general nested programs were first defined in [15]. Here we adapt the definition of stable models [10] (answer sets for normal programs) to normal nested logic programs. A set of atoms X is *closed under* a basic program P iff for any $r \in P$, $\mathit{Head}(r) \subseteq X$ whenever there is a $B \in \mathit{Body}(r)$ s.t. $B^+ \subseteq X$. The smallest set of atoms which is closed under a basic program P is denoted by $\mathrm{Cn}(P)$. The *reduct*, P^X, of a program P *relative to* a set X of atoms is defined in two steps. First, let $B \in \mathit{Body}(P)$ and let X be some set of atoms. Then the *reduct* B^X of B *relative to* X is defined as

$$B^X = \begin{cases} B^+ \text{ if } B^- \cap X = \emptyset \\ \bot \quad \text{otherwise.} \end{cases}$$

For a rule of the form (1) we define $r^X = h_1, \ldots, h_k \leftarrow B_1^X; \ldots; B_n^X$. Second, for a normal nested program P we define $P^X = \{r^X \mid r \in P \text{ and } \mathit{Body}(r^X) \neq \{\bot\}\}$. Then P^X is a basic program. We say that a set X of atoms is an *answer set* of a program P iff $\mathrm{Cn}(P^X) = X$. For normal logic programs this definition coincides with the definition of stable models [14]. The set of all answer sets of program P is denoted by $AS(P)$.

A *directed graph* (or *digraph*) G is a pair $G = (V, E)$ such that V is a finite, non-empty set (nodes) and $E \subseteq V \times V$ is a set (arcs). For a digraph $G = (V, E)$ and a vertex $v \in V$, we define the set of all *predecessors* (*successors*) of v as $\mathrm{Pred}(v) = \{u \mid (u, v) \in E\}$ ($\mathrm{Succ}(v) = \{u \mid (v, u) \in E\}$), respectively. Let $M \subseteq V$ be a subset of nodes of some digraph $G = (V, E)$. We define $\mathrm{Pred}(M) = \cup_{v \in M} \mathrm{Pred}(v)$ and $\mathrm{Succ}(M) = \cup_{v \in M} \mathrm{Succ}(v)$. The *connecting arcs of M wrt G* are defined as $CA(M) = ((\mathrm{Pred}(M) \times M) \cup (M \times \mathrm{Succ}(M))) \cap E$. A *path from v to v'* in $G = (V, E)$ is a finite subset $P_{vv'} \subseteq V$ such that $P_{vv'} = \{v_1, \ldots, v_n\}$, $v = v_1$, $v' = v_n$ and $(v_i, v_{i+1}) \in E$ for each $1 \leq i < n$.

Let $G = (V, E)$ and $G' = (V', E')$ be digraphs. Then G' is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. 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 $(v, v') \in E'$ iff $(v, v') \in E$.

G and G' are *isomorph* iff there exists a bijective mapping $h : V \to V'$ s.t. for all $v_1, v_2 \in V$ we have $(v_1, v_2) \in E$ iff $(h(v_1), h(v_2)) \in E'$. We write $G \approx_h G'$ if G and G' are isomorph and h is the underlying isomorphism[3].

We need a special kind of arc labeling for digraphs. $G = (V, E_0, E_1)$ is a directed graph whose arcs $E_0 \cup E_1$ are labeled zero (E_0) or one (E_1). Those graphs are called *0-1-digraphs*. The class of all 0-1-digraphs is denoted by $G_{0,1}$. We call arcs in E_0 and E_1 *0-arcs* and *1-arcs*, respectively. For G we distinguish 0-predecessors (0-successors) from 1-predecessors (1-successors) denoted by $\mathrm{Pred0}(v)$ ($\mathrm{Succ0}(v)$) and $\mathrm{Pred1}(v)$ ($\mathrm{Succ1}(v)$) for $v \in V$, respectively. For 0-1-digraphs we have $\mathrm{Pred}(v) = \mathrm{Pred1}(v) \cup \mathrm{Pred0}(v)$ and $\mathrm{Succ}(v) =$

[3] Note that h in the definition of $\approx_h$ is not unique, but the results of this paper hold for each isomorphism h.

$\mathrm{Succ0}(v) \cup \mathrm{Succ1}(v)$. The notations for 0- and 1-predecessors and 0- and 1-successors are generalized to sets of nodes as for predecessors and successors (see above).

In this paper we deal with special colorings of graphs. A *coloring* $\mathcal{C}$ of $G = (V, E_0, E_1)$ is a mapping $\mathcal{C} : V \to \{\oplus, \ominus\}$. We denote the set of all nodes colored with $\oplus$ or $\ominus$ by $\mathcal{C}_\oplus$ or $\mathcal{C}_\ominus$, respectively. Since we have $\mathcal{C}_\oplus \cap \mathcal{C}_\ominus = \emptyset$, we identify a coloring $\mathcal{C}$ with the pair $(\mathcal{C}_\oplus, \mathcal{C}_\ominus)$.

3 Block Graphs for Logic Programs

The block graph of a normal logic program was first defined in [16] and further elaborated as rule dependency graph in [13]. Those graphs contain information of dependencies between rules. Since we want to apply graph (program) transformations to block graphs which contract sets of nodes (rules) to single nodes (rules), we need so called *assignments* to remember the original sets of nodes (rules). For a set M let 2^M denote the power set of M.

Definition 1. *Let S and M be sets and let $\Phi : S \to 2^M$ be a total mapping. Then Φ is an* assignment of S to M *iff the following conditions hold*

1. *$\Phi(s) \neq \emptyset$ for each $s \in S$*
2. *for all $s, s' \in S$ we have if $\Phi(s) \neq \Phi(s')$ then $\Phi(s) \cap \Phi(s') = \emptyset$*
3. *$M = \bigcup_{s \in S} \Phi(s)$.*

For a subset $V \subseteq S$ we define $\Phi(V) = \bigcup_{v \in V} \Phi(v)$. Clearly, there is no unique assignment of S to M. Let $|M|$ denotes the number of elements in set M.

Definition 2. *Let S be some set, let $R : 2^S \to 2^S$ be a mapping s.t. Φ^R is an assignment of $R(M)$ to M for each $M \subseteq S$. Then R is a* reduction *iff the following conditions hold for each $M \subseteq S$*

1. *$|R(M)| \leq |M|$*
2. *for all $S_1, S_2 \subseteq R(M)$ we have if $S_1 \cap S_2 = \emptyset$ then $R(\Phi^R(S_1) \cup \Phi^R(S_2)) = R(\Phi^R(S_1)) \cup R(\Phi^R(S_2))$.*

Observe, that in the above definition the assignments Φ^R has to exist. Condition *2.* implies that a reduction R has to be modular wrt Φ^R, that is, subsets of M which are assigned (by Φ^R) to a single element in $R(M)$ can be mapped independently by R. If R is a reduction between programs it is called a *program reduction*. Let T be a mapping between 0-1-digraphs. Then T is called a *graph reduction* iff the restriction of T to the nodes V is a reduction for each 0-1-digraph $G_P = (V, E_0, E_1)$.

Definition 3. *Let $G = (V, E_0, E_1)$ be a 0-1-digraph and let P be a normal nested logic program. Then $G_P = (G, \Phi_P)$ is an* assigned 0-1-digraph for P *iff Φ_P is an assignment of V to P.*

In order to define correct graph transformations for block graphs we have to delete unnecessary literals and rules in logic programs.

Definition 4. *Let P be a normal (nested) logic program. A rule $r \in P$ is* captured (in P) *iff for each $B \in Body(r)$ and for each $p \in B^+ \cup B^-$ there exists $r' \in P$ s.t. $p \in Head(r')$. P is* captured *iff all rules in P are captured.*

Clearly, each normal nested logic program can be transformed to some captured program with same answer sets. Hence for the rest of the paper we assume every program to be captured.

The block graph of a normal nested logic program is a generalization of the block graph as defined in [16] for normal programs.

Definition 5. *Let P be a normal nested logic program. The* block graph $\Gamma_P = (V, E_0, E_1)$ *of P is a directed graph with nodes $V = P$ and labeled arcs*

$$E_0 = \{(r', r) \mid r', r \in P \text{ and } Head(r') \cap B^+ \neq \emptyset \text{ for some } B \in Body(r)\}$$
$$E_1 = \{(r', r) \mid r', r \in P \text{ and } Head(r') \cap B^- \neq \emptyset \text{ for some } B \in Body(r)\}.$$

If we define the mapping $I_P : P \to 2^P$ s.t. $I_P(r) = \{r\}$ for each $r \in P$ then obviously I_P is an assignment of P to P. Therefore (Γ_P, I_P) is an assigned 0-1-digraph for each normal nested program P.

In order to define so-called *application colorings* or shortened *a-colorings* for assigned 0-1-digraphs we need the following definitions.

Definition 6. *A normal nested logic program (set of rules) P is* grounded *iff there exists an enumeration $\langle r_i \rangle_{i \in I}$ of P such that for all $i \in I$ there is some $B \in Body(r_i)$ s.t. $B^+ \subseteq Head(\{r_1, \ldots, r_{i-1}\})$.*

Next we define applicability of a rule wrt to a set of rules.

Definition 7. *Let P be a normal nested program and let r be a rule (possibly not in P). Then r is* applicable wrt P *iff there exists $B \in Body(r)$ s.t. the following two conditions hold:*

1. *there exist $P_r \subseteq P$ s.t. P_r is grounded and $B^+ \subseteq Head(P_r)$*
2. *$B^- \cap Head(P) = \emptyset$.*

A set of rules S is *applicable wrt P* iff each rule in S is applicable wrt P. Now we are ready to define a-colorings for assigned 0-1-digraphs for normal nested logic programs.

Definition 8. *Let P be a normal nested logic program, let $G_P = (G, \Phi_P)$ be some assigned 0-1-digraph for P s.t. $G = (V, E_0, E_1)$ and let $\mathcal{C}$ be a total coloring of G. Then $\mathcal{C}$ is an* a-coloring *of G_P iff for each $v \in V$ we have*

AP $v \in \mathcal{C}_\oplus$ *iff $\Phi_P(v)$ is applicable wrt $\Phi_P(\mathcal{C}_\oplus)$.*

Let $\mathrm{AC}(G_P)$ denote the set of all a-colorings of G_P. Clearly, for normal programs Definition 8 coincides with the original definition of a-colorings as given in [16, 17]. However, Definition 8 generalizes the former definition in two aspects. First, Definition 8 deals with the more general class of normal nested logic programs. Additionally, a-colorings are defined for assigned 0-1-digraphs where a single node may correspond to a subset of rules of the underlying program.

Next we define equivalence between a-colorings and answer sets.

Definition 9. *Let P and P' be normal nested logic programs and let $G_P = (G, \Phi_P)$ be an assigned 0-1-digraph for P. Then we define $AC(G_P) \cong_{f,\Phi} AS(P')$ iff there exists a bijective mapping $f : AC(G_P) \rightarrow AS(P')$ and there exists an assignment Φ of P' to P s.t. $\Phi_P(\mathcal{C}_\oplus) = \Phi(GR(P', f(\mathcal{C})))$ for each $\mathcal{C} \in AC(G_P)$.*

Observe, that we omit the index f in $\cong_{f,\Phi}$ whenever there is no need to refer to the actual bijective mapping in Definition 9. The main results in [16, 17] directly imply that we have $\mathrm{AC}((\Gamma_P, I_P)) \cong_{I_P} AS(P)$ for each normal program P. Hence answer sets of normal programs can be computed by computing a-colorings. Consider program $P = \{a \leftarrow b, not\ c.\ c \leftarrow d, not\ a.\ b \leftarrow \top.\ d \leftarrow \top.\ \}$ and define $G_P = (G, \Phi_P)$ s.t. $G = (\{x, y\}, \emptyset, \{(x, y), (y, x)\})$, $\Phi_P(x) = \{a \leftarrow b, not\ c.\ , b \leftarrow \top.\ \}$ and $\Phi_P(y) = \{c \leftarrow d, not\ a.\ , d \leftarrow \top.\ \}$. Then G_P is an assigned 0-1-digraph for P. G_P has two a-colorings $C_1 = (\{x\}, \{y\})$ and $C_1 = (\{y\}, \{x\})$ corresponding to the two answer sets $X_1 = \{b, a\}$ and $X_2 = \{d, c\}$ of P, respectively. For example, for $C = C_1$ we have that condition **AP** holds for each $v \in \mathcal{C}_\oplus$ iff $\Phi_P(x)$ is applicable wrt to itself. This it true iff each rule in $\Phi_P(x)$ is applicable wrt $\Phi_P(x)$. Rule $b \leftarrow \top$ is applicable by definition, because we have if $B = \top$ then $B^+ = \emptyset$. For rule $a \leftarrow b, not\ c$ we have $B = (b, not\ c)$ and thus for $B^+ = \{b\}$ and $B^- = \{c\}$ conditions *1.* and *2.* of Definition 7 hold, because $Head(\Phi_P(x)) = \{a, b\}$.

We are able to generalize the former result in [16, 17] to normal nested programs.

Theorem 1. *For each $P \in nNLP$ we have $AC((\Gamma_P, I_P)) \cong_{I_P} AS(P)$.*

Definition 10. *Let $G_P = (G, \Phi_P)$ and $G_{P'} = (G', \Phi_{P'})$ be assigned 0-1-digraphs for normal nested programs P and P' s.t. $G = (V, E_0, E_1)$ and $G' = (V', E'_0, E'_1)$. Then we define $AC(G_P) \cong_{h,\Phi} AC(G_{P'})$ iff there exists a bijective mapping $h : AC(G_P) \rightarrow AC(G_{P'})$ and there exists an assignment Φ of V' to V. s.t. $\mathcal{C}_\oplus = \Phi(h(C)_\oplus)$ for each $C \in AC(G_P)$.*

As for Definition 9, if possible we omit the index h in $\cong_{h,\Phi}$. We have the following result relating equivalences between a-colorings and equivalences between a-colorings and answer sets.

Theorem 2. *Let G_P and $G_{P'}$ be assigned 0-1-digraphs for normal nested programs P and P', respectively, s.t. $AC(G_P) \cong_\Phi AC(G_{P'})$ and $AC(G_{P'}) \cong_{\Phi_{P'}} AS(P')$. Then we have $AC(G_P) \cong_{\Phi'} AS(P')$.*

4 Block Graph Transformations

The block graph of a logic program can be used to apply some graph transformations, which reduce the number of nodes. The basic idea is to contract a set of nodes to a single node when each node of the original set has the same color for all a-colorings. For example, two nodes corresponding to rules like $d \leftarrow c$ and $c \leftarrow a$, respectively, have the same color for all a-colorings (see Examples 2). Observe that situations like this heavily appear in many ASP-problems, e.g. blocks world planning (see Table 1 Section 6).

Formally, we define graph transformations which contract sets of nodes to single nodes as follows.

Definition 11. *Let $G = (V, E_0, E_1)$ be a 0-1-digraph and let $v, v' \in V$ be two nodes s.t. Pred(v') = Pred0(v') = $\{v\}$. Define graph transformation $T_0^{(v,v')}(G)$: $G_{0,1} \to G_{0,1}$ for G s.t. $T_0^{(v,v')}(G) = (V', E_0', E_1')$ where*

$$\begin{aligned} V' &= V \setminus \{v'\} \\ E_0' &= (E_0 \setminus CA(\{v'\})) \cup (\{v\} \times Succ0(\{v'\})) \\ E_1' &= (E_1 \setminus CA(\{v'\})) \cup (\{v\} \times Succ1(\{v'\})). \end{aligned}$$

Transformation $T_0^{(v,v')}$ contracts nodes v and v' to node v if v is the only 0-predecessor of v' and v' has no 1-predecessors. If we consider the arcs in a graph as dependencies between its nodes then v' depends on a single node v through 0-arc (v, v'). On the level of logic programs this means that a rule corresponding to v applies iff a rule corresponding to v' applies wrt some answer set.

Let $G = (V, E_0, E_1)$ be a 0-1-digraph and let $v, v' \in V$ be two nodes. We define graph transformation $T^{(v,v')}(G) : G_{0,1} \to G_{0,1}$ s.t. for $T^{(v,v')}(G) = (V', E_0', E_1')$ we have

$$\begin{aligned} V' &= V \setminus \{v'\} \\ E_0' &= (E_0 \setminus CA(\{v'\})) \cup (\mathrm{Pred0}(\{v'\}) \times \{v\}) \cup (\{v\} \times \mathrm{Succ0}(\{v'\})) \\ E_1' &= (E_1 \setminus CA(\{v'\})) \cup (\mathrm{Pred1}(\{v'\}) \times \{v\}) \cup (\{v\} \times \mathrm{Succ1}(\{v'\})). \end{aligned}$$

Definition 12. *Let $G = (V, E_0, E_1)$ be a 0-1-digraph and let $v, v' \in V$ be two nodes s.t. Pred(v) = Pred(v'). Define graph transformation $T_B^{(v,v')}(G) : G_{0,1} \to G_{0,1}$ for G as $T_B^{(v,v')}(G) = T^{(v,v')}(G)$.*

If v and v' have the same predecessors then they depend on the same set of nodes and thus are contracted to one node. For programs this means that two rules corresponding to v and v', respectively, have the same body.

Definition 13. *Let $G = (V, E_0, E_1)$ be a 0-1-digraph and let $v, v' \in V$ be two nodes s.t. Succ(v) = Succ(v') $\neq \emptyset$. Define graph transformation $T_H^{(v,v')}(G)$: $G_{0,1} \to G_{0,1}$ for G as $T_H^{(v,v')}(G) = T^{(v,v')}(G)$.*

In Definition 13 the idea is to contract two nodes if they have the same successors, because their influence on those successors is the same. We cannot contract nodes if they do not have any successors, since they do not influence other nodes. Two rules of some normal program corresponding to v and v', respectively, have to have the same head. In order to contract nodes, each of the above three definitions takes different dependencies into account.

Let (G, Φ_P) be an assigned 0-1-digraph for P s.t. $G = (V, E_0, E_1)$ and for $i \in \{0, H, B\}$ let $T_i^{(v,v')}(G) = (V', E_0', E_1')$ for nodes $v, v' \in V$. Define $\Phi_P^{(v,v')}$: $V' \to 2^P$ for each $u \in V'$ as

$$\Phi_P^{(v,v')}(u) = \begin{cases} \Phi_P(v) \cup \Phi_P(v') & \text{if } u = v \\ \Phi_P(u) & \text{otherwise.} \end{cases}$$

If (G, Φ_P) is an assigned 0-1-digraph for P then obviously $(T_i^w(G), \Phi_P^w)$ is an assigned 0-1-digraph for P for each $i \in \{0, H, B\}$ where $w = (v, v')$.

Lemma 1. *Let P be a normal nested logic program with block graph Γ_P and let $r, r' \in P$ be two nodes s.t. Pred(r') = Pred0(r') = $\{r\}$ or Pred(r) = Pred(r') or Succ(r) = Succ(r') $\neq \emptyset$. Then for each $\mathcal{C} \in AC((\Gamma_P, I_P))$ we have $\{r, r'\} \subseteq \mathcal{C}_\oplus$ or $\{r, r'\} \subseteq \mathcal{C}_\ominus$.*

This lemma justify our intuition, that nodes in a block graph which may be contracted according to definitions 11, 12 and 13 have the same color for each a-coloring.

For the rest of the paper we give all results for all three defined graph transformations simultaneously through index i. Let $G = (G_P, \Phi_P)$ be some 0-1-digraph, let $i \in \{0, B, H\}$ and let $w_j = (v_j, v'_j)$ for each $1 \leq j \leq n+1$ where v_j and v'_j are nodes of G_P. Then $T_i^{w_n} \circ \ldots \circ T_i^{w_1}$ is a *maximal graph transformation for* G iff it is the composition of graph transformations $T_i^{w_1}$ for G, ..., $T_i^{w_n}$ for $T_i^{n-1} \circ \ldots \circ T_i^{w_1}(G)$ and for the graph $T_i^{w_n} \circ \ldots \circ T_i^{w_1}(G)$ there is no further graph transformation $T_i^{w_{n+1}}$ possible. We have the following theorem stating that there is a unique maximal composition of those graph transformations.

Lemma 2. *Let G be some 0-1-digraph and let $T_i = T_i^{w_n} \circ \ldots \circ T_i^{w_1}$ be some maximal graph transformation for G where $w_j = (v_j, v'_j)$ for nodes v_j and v'_j of G ($1 \leq j \leq n$). Then $T_i(G)$ is unique for $i \in \{0, H, B\}$.*

With T_i we denote the unique maximal graph transformation $T_i^{w_n} \circ \ldots \circ T_i^{w_1}(G)$ for each $i \in \{0, H, B\}$. We define $\Phi_P^i = \Phi_P^{w_n}$. According to Lemma 2, Φ_P^i is well-defined. We have the following result.

Lemma 3. *The graph transformation $T_i : G_{0,1} \to G_{0,1}$ is a graph reduction for $i \in \{0, H, B\}$.*

Lemma 3 implies that for graph transformation T_i there exists a corresponding assignment of the nodes of $T_i(G_P)$ to nodes in G_P (see Definition 2). We denote this assignment with Φ^i for $i \in \{0, H, B\}$.

Theorem 3. *Let (G, Φ_P) be some assigned 0-1-digraph for normal nested program P. Then $(T_i(G), \Phi_P^i)$ is an assigned 0-1-digraph for P for each $i \in \{0, H, B\}$ where $\Phi_P^i = \Phi_P \circ \Phi^i$.*[4]

This theorem tells us that if we apply graph transformations T_0, T_H and T_B in any order then we end up with some assigned 0-1-digraph for P.

For example graph transformation T_0 contracts 0-paths without incoming arcs to single nodes as much as possible. For logic program

$$P = \left\{ \begin{array}{ll} r_1 : a \leftarrow not\ b. & r_2 : b \leftarrow not\ a. \\ r_3 : c \leftarrow a. & r_4 : c \leftarrow b. \\ r_5 : d \leftarrow c. & \end{array} \right\} \tag{2}$$

[4] Observe that $\Phi_P \circ \Phi^i(v) = \Phi_P(\Phi^i(v))$ for each $v \in V$ where V is the set of nodes of G.

it is possible to apply block graph transformations $T_0^{r_1,r_3}$ and $T_0^{r_2,r_4}$. Figure 1 shows the block graph Γ_P of program (2) on the left and the resulting graph $T_0(\Gamma_P)$ on the right side. Additionally, on the left and right of '/' we have depicted the two a-colorings of (Γ_P, I_P), respectively. We also have depicted the two a-colorings of $(T_0(\Gamma_P), \Phi_P^0)$. Observe, that we obtain the two corresponding answer sets $X_1 = \{a, c, d\}$ and $X_2 = \{b, c, d\}$ directly from those a-colorings by collecting the heads of rules corresponding to $\oplus$-colored nodes. Let $V' = \{v_1, v_2, v_5\}$ be the set of nodes of $T_0(\Gamma_P)$. Then for the rule assignment

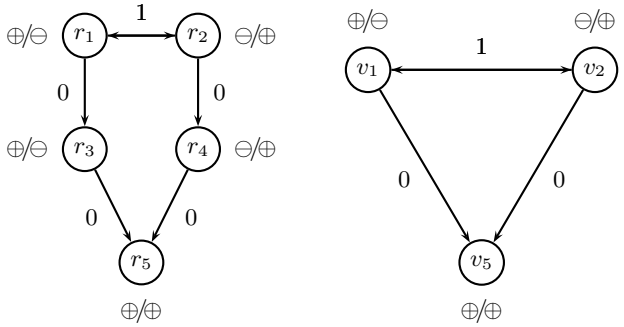

Fig. 1. Block graph Γ_P of program (2) on the left and resulting graph $T_0(\Gamma_P)$ on the right.

Φ_P^0 we have $\Phi_P^0(v_5) = \{r_5\}$, $\Phi_P^0(v_1) = \{r_1, r_3\}$ and $\Phi_P^0(v_2) = \{r_2, r_4\}$. Observe, that node v_5 is applicable wrt both a-colorings of $(T_0(\Gamma_P), \Phi_P^0)$, because $\mathit{Body}(r_5) = \{c\}$ and $c \in \mathit{Head}(\mathcal{C}_\oplus)$. Thus we have $v_5 \in \mathcal{C}_\oplus$ for both a-colorings of $(T_0(\Gamma_P), \Phi_P^0)$. We have the following result.

Theorem 4. *Let P be a normal nested logic program. Then $AC((T_i(\Gamma_P), \Phi_P^i)) \cong_{I_P} AS(P)$ for each $i \in \{0, H, B\}$.*

According to this theorem answer sets of normal programs can be computed by computing a-colorings. Similar results to Theorem 4 also hold for compositions of T_i and T_j for $i, j \in \{0, H, B\}$ (see Theorem 3).

Definition 14. *Let T be a graph reduction and let R be a program reduction. Then T and R are* corresponding to each other *iff the following conditions hold for each program $P \in nNLP$*

1. $T(\Gamma_P) \approx_h \Gamma_{R(P)}$
2. $AC((T(\Gamma_P), \Phi^T)) \cong_{h,\Phi} AC((\Gamma_{R(P)}, \Phi^R))$.

Observe, that in the above definition T and R are reductions and therefore the assignments Φ^T and Φ^R are given implicitly through Definition 2. According to Definition 14, a graph reduction corresponds to a program reduction if the transformed block graph $T(\Gamma_P)$ is isomorphic to the block graph of the transformed program $R(P)$ (cond. *1.* of Definition 14). Furthermore, the a-colorings of both graphs should be equivalent (cond. *2.* of Definition 14). Also observe that both

conditions rely on the same bijective mapping h which means that the equivalence of the a-colorings has to hold wrt graph isomorphism h. At first sight, one may think that a transformed block graph should be the block graph of some **normal** logic program which is "equivalent" to the original one. Surprisingly, this is not the case.

Theorem 5. *For the graph reduction T_i there is* **no** *corresponding program reduction $R_i : nLP \to nLP$ for $i \in \{0, H, B\}$.*

For example, let P denote program (2) and consider Γ_P and $T_0(\Gamma_P)$ (see Figure 1). Let $P' \in$ nLP be some normal program s.t. its $\Gamma_{P'}$ is isomorphic to the transformed graph $T_0(\Gamma_P)$. Then Theorem 5 says that the a-colorings of $(\Gamma_{P'}, I_{P'})$ are not equivalent to the depicted a-colorings of $(T_0(\Gamma_P), \Phi^0_P)$ in Figure 1. First, we show that the rule in P' corresponding to node v_5 in $T_0(\Gamma_P)$ cannot have a single positive head atom like in $(x \leftarrow y\ )$. Assume $(x \leftarrow y\ ) \in P'$ would be the rule corresponding to node v_5. This would imply that the two rules corresponding to nodes v_1 and v_2 would both have head x (see Definition 5). Since rules corresponding to nodes v_1 and v_2 block each other, each of them would also blocks itself. This is a contradiction, because it is not the case in the graph $T_0(\Gamma_P)$. Therefore for each **normal** program P' with block graph $T_0(\Gamma_P)$ node r_d has to correspond to a rule with two positive body atoms, lets say $\Phi_{P'}(v_5) = (x \leftarrow y, y'\ )$. Let us take normal program $P' = \{(y \leftarrow not\ \ y'\ ), (y' \leftarrow not\ \ y\ ), (x \leftarrow y, y'\ )\}$ where the first two rules correspond to nodes v_1 and v_2, respectively. Then the block graph of P' is isomorphic to $T_0(\Gamma_P)$. But it is easy to see that node v_5 (corresponding to rule $x \leftarrow y, y'\ )$ cannot be in $\mathcal{C}_\oplus$ for some $\mathcal{C} \in \mathrm{AC}((\Gamma_{P'}, I_{P'}))$. In other words, we have $v_5 \in \mathcal{C}_\ominus$ for each $\mathcal{C} \in \mathrm{AC}((\Gamma_{P'}, I_{P'}))$. The same holds for each normal program P' which has a block graph isomorphic to $T_0(\Gamma_P)$. Hence for all **normal** logic programs s.t. condition *1.* of Definition 14 holds we have that condition *2.* of Definition 14 does not hold. Similar examples can be found for the transformations T_H and T_B.

5 Program Transformations

In the last section we have seen that there are simple a-coloring preserving block graph transformations s.t. in general for the transformed block graph there is no corresponding **normal** program. Next, we will see that it is possible to define program transformations between **normal nested** logic programs corresponding to block graph transformations.

Definition 15. *Let P be a normal nested program and let $r, r' \in P$ rules. Then the mappings $R_0^{r,r'}, R_H^{r,r'}, R_B^{r,r'} : nNLP \to nNLP$* for *$P$ are defined as follows.*

If $Body(r) = \{B\}$, $B^+ \subseteq Head(r')$, $Head(P \setminus \{r'\}) \cap B^+ = \emptyset$ and $Head(P) \cap B^- = \emptyset$ then define

$$R_0^{r,r'}(P) = (P \setminus \{r, r'\}) \cup \{(Head(r) \cup Head(r')) \leftarrow Body(r')\}.$$

If $Head(r) = Head(r')$ *then define*

$$R_H^{r,r'}(P) = (P \setminus \{r, r'\}) \cup \{Head(r) \leftarrow (Body(r) \cup Body(r'))\}.$$

If $Body(r) = Body(r')$ *then define*

$$R_B^{r,r'}(P) = (P \setminus \{r, r'\}) \cup \{(Head(r) \cup Head(r')) \leftarrow Body(r)\}.$$

Let P be some normal nested program, let $i \in \{0, H, B\}$ and let $w_j = (r_j, r'_j)$ for each $1 \leq j \leq n+1$ where $r_j, r'_j \in P$. Then $R_i^{w_n} \circ \ldots \circ R_i^{w_1}$ is a *maximal program transformation for* P iff it is the composition of program transformations $R_i^{w_1}$ for P, $\ldots$, $R_i^{w_n}$ for $R_i^{w_{n-1}} \circ \ldots \circ R_i^{w_1}(P)$ and for the program $R_i^{w_n} \circ \ldots \circ R_i^{w_1}(P)$ there is no further program transformation $R_i^{w_{n+1}}$ possible. R_B (R_H) is a well-known transformation where all rules with same body (head) are transformed to one new rule with a nested head (body), respectively. According to [15] we have that programs P, $R_B(P)$ and $R_H(P)$ have the same answer sets. It is easy to see that the same holds for R_0, that is, $AS(P) = AS(R_0(P))$.

We have the following result corresponding to Lemma 2.

Lemma 4. *Let* $R_i = R_i^{w_n} \circ \ldots \circ R_i^{w_1}$ *be some maximal program transformation for* $P \in nNLP$ *where* $w_j = (r_j, r'_j)$ *for rules* $r_j, r'_j \in P$ *(*$1 \leq j \leq n$*). Then* $R_i(P)$ *is unique for* $i \in \{0, H, B\}$.

We also have a result corresponding to Lemma 3.

Lemma 5. *The program transformation* $R_i : nNLP \rightarrow nNLP$ *is a program reduction for* $i \in \{0, H, B\}$.

Observe, that if R_i is a program reduction then Φ^{R_i} exists according to Definition 2. For program transformations we get a result similar to Theorem 3.

Theorem 6. *Let* P *be some normal nested program. Then* $(\Gamma_{R_i(P)}, \Phi^{R_i})$ *is an assigned 0-1-digraph for* P *for each* $i \in \{0, H, B\}$.

Finally, we obtain a result stating that R_i is a program reduction corresponding to T_i.

Theorem 7. *For each* $i \in \{0, H, B\}$ *we have that* T_i *corresponds to* R_i.

This theorem tells us that the program reduction R_i between nested logic programs provide a semantics for the block graph transformation T_i. Furthermore, Theorems 2, 4, 6 and 7 imply the following corollary.

Corollary 1. *Let* Γ_P *be the block graph of a normal nested logic program* P. *Then* $AC((T_i(\Gamma_P), \Phi_P^i)) \cong_{\Phi^{R_i}} AS(R_i(P))$ *for* $i \in \{0, H, B\}$.

This corollary justifies Definition 14, since the application of T_i to block graphs is equivalent to the application of R_i to normal nested programs.

	p1		p2		p3		p4	
	no	Trans	no	Trans	no	Trans	no	Trans
rules	1343	354	2783	645	7595	1242	21539	2794
ass	23657	14326	80549	45321	35494	18575	21308644	15214111
time	0.36	0.22	1.19	1.18	1.03	0.87	408	312

Table 1. Number of rules, color assignments (ass), as well as time in seconds for small planning examples.

6 Empirical Results and Discussion

We have implemented block graph transformation T_i for each $i \in \{0, H, B\}$ in the `noMoRe` system [1][5]. According to Corollary 1 this can be interpreted as answer sets computation of normal nested logic programs. Table 1 shows the cumulative influence of applying all presented transformations together (indicated by Trans). Invoking only one of the three transformations leads to similar but weaker improvements. The tested problem instances are taken from [7], where p1 and p3 are four step blocks world planning problems whereas p2 and p4 are five step blocks world planning problems. All result are given for computing all answer sets. The first line shows the number of rules, the second one gives the number of color assignments[6] and the last line contains the time in seconds used to compute all solutions. Depending on the structure of the examples we may save time and space when using graph transformations.

7 Related Work and Conclusion

In the literature we find many graph-based approaches to logic programming. Obviously, graphs are used to detect structural properties of programs, such as stratification [2], existence of answer sets [9, 3] or the actual characterization of answer sets or well-founded semantics [4, 3, 16, 17]. The usage of rule-oriented dependency graphs like block graphs is common to [4, 3, 16, 13]. In fact, the coloration of such graphs for characterizing answer sets was independently developed in [3] and [16] and further investigated in [13].

However, the aim of this paper is to detect answer set preserving subsets of rules which can be reduced to smaller subsets by means of block graph transformations. Since there is a one to one correspondence between a-colorings and answer sets [16, 13] different a-coloring preserving block graph transformations reducing the number of nodes are defined. Furthermore, we are able to show that block graphs together with transformations are more expressive than normal programs, because there exist **no** corresponding program transformations between **normal** programs. However, there are polynomial, faithful and non-modular

[5] `http://www.cs.uni-potsdam.de/~linke/nomore`

[6] By a color assignment we mean the action of coloring a node which is currently uncolored.

program transformations between **normal nested** logic programs, which are shown to be equivalent to the proposed graph transformations (see [12] for details on faithful transformations). This result has two important consequences. First, nested logic programs can be used for improving answer set computation of normal logic programs, since a-colorings and corresponding answer sets can be computed more efficiently after applying graph transformations as demonstrated in the `noMoRe` system. Basicly the computation of the deterministic consequences of `noMoRe` is improved by providing a more compact representation of block graphs. This distinguishes our work from [8] where heuristics are utilized for the same reasons. Second, as a byproduct, answer sets of normal nested logic programs can be computed by computing a-colorings of transformed block graphs corresponding to normal nested logic programs. In fact, the current version of `noMoRe` also directly computes answer sets of normal nested programs.

A related method which reduces nested programs where the bodies can be any nested expression to extended programs is introduced in [20]. However, this method transformas a nested program into an extended one which in general has more rules than the original one. Although our approach relies on similar transformations, they are applied in the other direction, that is, possibly large sets of rules in the "simpler" language of normal programs are transformed to smaller sets of rules in the more expressive language of nested programs. Usually a rather simple core language is used and then more advanced syntactical constructs are translated to the core language (see [19] for an example). Our work indicates that it is sometimes reasonable to implement a more expressive version of the language to obtain more efficient systems. Of course one has to pay a price for this: the underlying algorithms have to deal with the more expressive syntax, which often is more difficult to implement. In particular, other ASP systems, not relying on block graphs, may benefit from our results if their implementation technique can be directly generalized to normal nested logic programs. In this case, we suspect positive effects similar to those obtained for the `noMoRe` system. If this is not possible then a system cannot benefit from our results.

Acknowledgements

The author was partially supported by the German Science Foundation (DFG) under grant FOR 375/1 and SCHA 550/6, TP C and the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-37004 WASP project.

References

1. C. Anger, K. Konczak, and T. Linke. `NoMoRe`: Non-monotonic reasoning with logic programs. In G. Ianni and S. Flesca, editors, *(JELIA'02)*, volume 2424 of *LNAI*. Springer Verlag, 2002.
2. K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann Publishers, 1987.

3. G. Brignoli, S. Costantini, O. D'Antona, and A. Provetti. Characterizing and computing stable models of logic programs: the non-stratified case. In C. Baral and H. Mohanty, editors, *Proc. of Conference on Information Technology*, pages 197–201, Bhubaneswar, India, December 1999. AAAI Press.
4. Y. Dimopoulos and A. Torres. Graph theoretical structures in logic programs and default theories. *Theoretical Computer Science*, 170:209–244, 1996.
5. D. East and M. Truszczyński. dcs: An implementation of datalog with constraints. In *Proceedings of the National Conference on Artificial Intelligence*. MIT Press, 2000.
6. 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, *LPNMR*, volume 1265 of *LNAI*, pages 363–374. Springer Verlag, 1997.
7. W. Faber, N. Leone, and G. Pfeifer. Pushing goal derivation in dlp computations. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *LPNMR'99*, volume 1730 of *LNAI*, pages 177–191, El Paso, Texas, USA, 1999. Springer Verlag.
8. W. Faber, N. Leone, and G. Pfeifer. Experimenting with heuristics for answer set programming. In B. Nebel, editor, *IJCAI*, pages 635–640. Morgan Kaufmann Publishers, 2001.
9. F. Fages. Consistency of clark's completion and existence of stable models, 1992.
10. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the International Conference on Logic Programming*, pages 1070–1080. The MIT Press, 1988.
11. M. Gelfond and V. Lifschitz. Classical negation in logic programs and deductive databases. *New Generation Computing*, 9:365–385, 1991.
12. T. Janhunen. Comparing the expressive power of some syntactically restricted classes of logic programs. In Dix J, L. Feriñas del Cerro, and U. Furbach, editors, *Proceedings of the 1st International Conference on Computational Logic*, number 1861 in *LNAI*, pages 852–866. Springer Verlag, 2000.
13. K. Konczak, T. Linke, and T. Schaub. Graphs and colorings for answer set programming: Abridged report. 2003. Submitted to LPNMR.
14. V. Lifschitz. Answer set planning. In *Proceedings of the 1999 International Conference on Logic Programming*, pages 23–37. MIT Press, 1999.
15. V. Lifschitz, L. Tang, and H. Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389, 1999.
16. T. Linke. Graph theoretical characterization and computation of answer sets. In B. Nebel, editor, *IJCAI*, pages 641–645. Morgan Kaufmann Publishers, 2001.
17. T. Linke, C. Anger, and K. Konczak. More on nomore. In G. Ianni and S. Flesca, editors, *JELIA'02*, volume 2424 of *LNAI*. Springer Verlag, 2002.
18. 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, *LPNMR*, pages 420–429. Springer, 1997.
19. I. Niemelä and P. Simons. Extending the smodels system with cardinality and weight constraints. *Logic-Based Artificial Intelligence*, pages 491–521, 2000.
20. J. You, L. Yuan, and M. Zhange. On the equivalence between answer sets and models of completion for nested logic programs. In *Proc. IJCAI03*, page to appear, 2003.