Suitable Graphs for Answer Set Programming

Thomas Linke

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

Abstract. Often graphs are used to investigate properties of logic programs. In general, different graphs represent different kinds of information of the corresponding programs. Sometimes this information is not sufficient for solving a certain problem. In this paper we define graphs which are *suitable* for computing answer sets of different classes of logic programs. Intuitively, a graph associated to a logic program is suitable for answer set semantics if its structure is sufficient to compute the answer sets of the corresponding program. That is, algorithms that use suitable graphs do not have to consider the original logic program any longer. We investigate different classes of graphs which are suitable for answer set computation of normal nested logic programs, normal logic programs and normal programs with at most on positive body atom.

1 Introduction

In the literature many different types of graphs are associated with a given logic program in order to investigate its properties or to compute its answer sets. Among them we find dependency graphs (DGs) [2], (defined on literals), rule graphs (RGs) [7] (defined on rules for reduced negative programs) and more recently extended dependency graphs (EDG) [4, 6] as well as block graphs [14] (defined on rules of normal programs). Block graphs are also called rule dependency graphs (RDGs) [11] since they reflex positive and negative dependency between the rules of a logic program. In general, different graphs represent different dependencies among rules or atoms of the corresponding programs. Sometimes this information is not sufficient for solving a certain problem. In this paper we are interested in the question which graphs are suitable for computing answer sets of logic programs[1].

Intuitively, a class of graphs is suitable for answer set semantics of a class of programs if the structure of a graph is sufficient to compute the answer sets of the corresponding program. For an example, let us look at the rule dependency graphs [11] (block graphs [14]) of the following two programs:

$P_1 = \{r_1 : a \leftarrow not\ b.\ \ r_2 : b \leftarrow not\ a.\ r_3 : c \leftarrow a.\ \ r_4 : d \leftarrow b.\ \ r_5 : x \leftarrow c, d.\ \}$
$P_2 = \{r_1 : a \leftarrow not\ b.\ \ r_2 : b \leftarrow not\ a.\ r_3 : c \leftarrow a.\ \ r_4 : c \leftarrow b.\ \ r_5 : x \leftarrow c.\ \}.$

[1] The language of logic programs is not the only one used for ASP. Others are propositional logic or DATALOG with constraints [8].

Program P_1 has answer sets $\{a, c\}$ and $\{b, d\}$, whereas program P_2 has answer sets $\{a, c, x\}$ and $\{b, d, x\}$. With the above rule naming both programs have the same rule dependency graphs (block graph) which is shown in Figure 1, where 0-arcs (1-arcs) correspond dependencies between rules via positive (negative) body literals, respectively. Since the semantic difference between the two

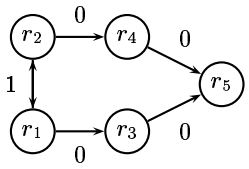

Fig. 1. The block graph of programs P_1 and P_2.

programs cannot be detected from the structure of the corresponding rule dependency graph, those graphs are not suitable for answer set computation of normal programs. For dealing with this problem there different solutions. For example, the approaches in [7, 4] rely on a translation of normal programs into negative ones before using rule graphs and extended dependency graphs, respectively, for characterizing answer sets. Notice that the above problem disappears for negative programs. On the other hand one may use some kind of additional meta-information not present in the graph structure for rules like r_5 in program P_1. In fact, this is done in the `noMoRe` system [1] and its underlying theory (see [16, 11, 15] for details).

In this paper, we make a first step into another research direction by investigating suitable graphs for different classes of programs. More precisely, we deal with the classes of normal nested logic programs (nNLP) a syntactically restricted subclass of nested logic programs [13], normal logic programs (nLP) and normal logic programs with at most one positive body atom (nLP_1). The central contribution of our approach is to allow different kinds of nodes in one graph, in order to incorporate more information of the underlying logic program. For example, the so-called *rule-head-graph* of a given program P has the rules **and** the heads of P as its nodes (see Figure 2 for the rule-head-graphs of P_1 and P_2). This ensures that the different semantics of programs P_1 and P_2 are reflected in the corresponding rule-head-graph. Furthermore, we introduce *body-head-graphs*, where the bodies **and** heads of a corresponding program form the set nodes. We prove that body-head-graphs are suitable for answer set computation of normal nested logic programs. To the best of our knowledge, this is the first time that graphs corresponding to normal nested logic programs are introduced and utilized to characterize and compute answer sets for those programs. Additionally, we obtain suitability results for all other of the above mentioned classes of programs.

2 Background

In this paper we consider a proper subclass of propositional nested logic programs [12]. 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* then it is called 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. Furthermore, with nLP_1 we denote the class of normal programs with at most one positive body literal. Notice that $\mathrm{nLP}_1 \subset \mathrm{nLP} \subset \mathrm{nNLP}$. 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 $Head(P) = \cup_{r \in P} Head(r)$ and $Body(P) = \cup_{r \in P} Body(r)$. 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 $r = Head(r) \leftarrow B$.

Answer sets for general nested programs were first defined in [13]. Here we adapt the definition of stable models [9] (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 $r \in P$, $Head(r) \subseteq X$ whenever there is a $B \in Body(r)$ 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 $B \in Body(P)$ 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 $P^X = \{r^X \mid r \in P \text{ and } Body(r^X) \neq \{\bot\}\}$. 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 [12]. The set of all answer sets of program

P is denoted by $AS(P)$. Let P be a program and let X be a set of atoms. We define the set of generating bodies and the set of generating rules of P wrt X as

$B_P(X) = \{B \in Body(P) \mid B^+ \subseteq X \text{ and } B^- \cap X = \emptyset\}$ and
$R_P(X) = \{r \in P \mid \text{ there is some } B \in Body(r) \text{ s.t. } B \in B_P(X)\}$, respectively.

A 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 $B \in Body(r_i)$ with $B^+ \subseteq Head(\{r_1, \ldots, r_{i-1}\})$. Then X is an answer set of P iff we have $X = \mathrm{Cn}(R_P(X)^+)$. This characterizes answer sets in terms of generating rules.

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 $\mathrm{Succ}(M) = \cup_{v \in M} \mathrm{Succ}(v)$. We need a special kind of arc labeling for digraphs. $G = (V, E_0, E_1, E)$ is a directed graph whose arcs $E_0 \cup E_1 \cup E$ are labeled zero (E_0) or one (E_1) and additionally there are arcs without labels (E). Those graphs are called *0-1-digraphs* and $G_{0,1}$ is the class of all 0-1-digraphs. 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 $\mathrm{Succ}(v) = \mathrm{Succ0}(v) \cup \mathrm{Succ1}(v)$. 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 sometimes 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 Graphs for Logic Programs

In this section we define several directed graphs, which are shown to be suitable for different classes of logic programs in the next section.

Definition 1. *Let P be a logic program.*

The BH-graph (body-head-graph) $BH_P = (V, E_0, E_1, E)$ *of P is a directed graph with nodes $V = Body(P) \cup Head(P)$ and labeled arcs*

$$\begin{aligned} E &= \{(B, h) \mid B \in Body(P), h \in Head(P) \text{ and } (h \leftarrow B) \in P\} \\ E_0 &= \{(h, B) \mid B \in Body(P), h \in Head(P) \text{ and } h \in B^+\} \\ E_1 &= \{(h, B) \mid B \in Body(P), h \in Head(P) \text{ and } h \in B^-\}. \end{aligned}$$

The RH-graph (rule-head-graph) $RH_P = (V, E_0, E_1, E)$ *of P is a directed graph with nodes $V = P \cup Head(P)$ and labeled arcs*

$$\begin{aligned} E &= \{(r, h) \mid r \in P, h \in Head(P) \text{ and } Head(r) = \{h\}\} \\ E_0 &= \{(h, r) \mid r \in P, h \in Head(P) \text{ and } h \in B_r^+\} \\ E_1 &= \{(h, r) \mid r \in P, h \in Head(P) \text{ and } h \in B_r^-\}. \end{aligned}$$

The BR-graph (body-rule-graph) $BR_P = (V, E_0, E_1, E)$ *of* P *is a directed graph with nodes* $V = P \cup Body(P)$ *and labeled arcs*

$$\begin{aligned} E &= \{(B, r) \mid r \in P, B \in Body(P) \text{ and } Body(r) = \{B\}\} \\ E_0 &= \{(r, B) \mid r \in P, B \in Body(P) \text{ and } Head(r) \subseteq B^+\} \\ E_1 &= \{(r, B) \mid r \in P, B \in Body(P) \text{ and } Head(r) \subseteq B^-\}. \end{aligned}$$

Observe that all graphs in Definition 1 have two different kinds of nodes such as rules and head for rule-head-graphs, bodies and rules for body-rule-graphs and bodies and heads for body-head-graphs. Oppositely many graphs found in the literature have a single kind of nodes such as the literals or the rules of a given program. The introduction of two different kinds of nodes enables us to put more information into graphs corresponding to logic programs.

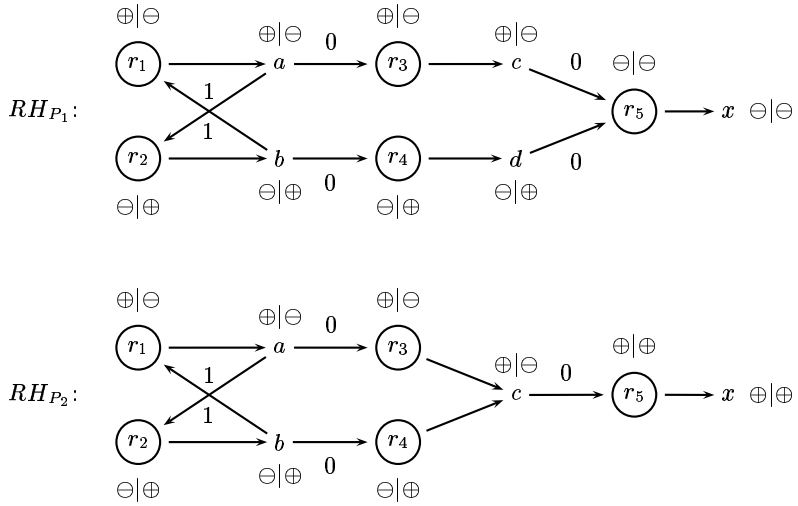

Fig. 2. The rule-head-graphs RH_{P_1} and RH_{P_2} of programs P_1 and P_2 together with its a-colorings (see Definition 4 in Section 4).

For completeness we also investigate three types of graphs where the nodes are the bodies, rules and heads, respectively.

Definition 2. *Let* P *be a logic program.*

The B-graph (body-graph) $B_P = (V, E_0, E_1, \emptyset)$ *of* P *is a directed graph with nodes* $V = Body(P)$ *and labeled arcs*

$$\begin{aligned} E_0 &= \{(B', B) \mid \exists r' \in P \text{ s.t. } B' \in Body(r') \text{ and } Head(r') \subseteq B^+\} \\ E_1 &= \{(B', B) \mid \exists r' \in P \text{ s.t. } B' \in Body(r') \text{ and } Head(r') \subseteq B^-\}. \end{aligned}$$

The R-graph (rule-graph) $R_P = (V, E_0, E_1, \emptyset)$ *of* P *is a directed graph with nodes* $V = P$ *and labeled arcs*

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

The H-graph (head-graph) $BH_P = (V, E_0, E_1, \emptyset)$ *of* P *is a directed graph with nodes* $V = Head(P)$ *and labeled arcs*

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

In order to obtain a uniform graph representation for this paper, we have not deleted the unnecessary empty set of unlabeled arcs in Definition 2. Observe, that R_P is the rule dependency graph (block graph) as defined in [11, 14] provided that P is a normal logic program.

4 Graph Colorings Characterizing Answer Sets

As done in [4, 14, 11] for normal programs, we now characterize answer sets of normal nested programs as special non-standard graph colorings. We need the following definitions.

Definition 3. *Let* P *be a normal nested program and let* B *be a conjunction of literals. Then*

1. B *is* grounded wrt P *iff there exists* $S_B \subseteq P$ *s.t.* S_B *is grounded and* $B^+ \subseteq Head(S_B)$
2. B *is* blocked wrt P *iff* $B^- \cap Head(P) \neq \emptyset$.

Now let P be a normal nested program and let $P' \subseteq P$ be a subset of rules. A body $B \in Body(P)$ is *applicable wrt* P' iff B is grounded and not blocked wrt P'. Next we define applicable rules and heads wrt applicable bodies. A rule $r \in P$ (possibly not in P') is *applicable wrt* P' iff there exists some $B \in Body(r)$ s.t. B is applicable wrt P'. Similarly a head $h \in Head(P)$ (possibly not in $Head(P')$) is *applicable wrt* P' iff there exists some rule $r \in P$ s.t. r is applicable wrt P' and $h \in Head(r)$.

Definition 4. *Let* P *be a normal program, let* $\Gamma \in \{RH, BR, BH, B, B, H\}$*, let* $\Gamma_P = (V, E_0, E_1, E)$ *be the* Γ*-graph of* P *and let* $\mathcal{C}$ *be a total coloring of* Γ_P*. Then* $\mathcal{C}$ *is an* a-coloring *of* Γ_P *iff for each* $v \in V$ *we have*

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

For each $\Gamma \in \{BH, RH, BR, B, R, H\}$ let $\mathrm{AC}(\Gamma_P)$ denote the set of all a-colorings of Γ_P. For example, the two a-colorings of the rule-head-graphs of programs P_1 and P_2 are given in Figure 2 on the left and right side of |,respectively.

It is easy to see, that for both programs the a-colorings are in one to one correspondence with the respective answer sets, by just looking at the $\oplus$-colored head-nodes.

Let $S_{RH} \subseteq P \cup Head(P)$, $S_{BR} \subseteq P \cup Body(P)$, $S_{BH} \subseteq Body(P) \cup Head(P)$, $S_B \subseteq Body(P)$, $S_R \subseteq P$ and $S_H \subseteq Head(P)$ be subsets of $P \cup Body(P) \cup Head(P)$. Then for $x \in \{BH, RH, BR, B, R, H\}$ the *set of rules* $Rs(S_x)$ is defined as follows:

$$\begin{array}{ll} Rs(S_x) = P \cap S_x & \text{if } x \in \{RH, BR, R\} \\ Rs(S_x) = \{r \in P \mid Body(r) \cap S_x \neq \emptyset\} & \text{if } x \in \{BH, B\} \\ Rs(S_x) = \{r \in P \mid \exists B \in Body(r) \text{ s.t. } B_r^+ \subseteq S_x\} & \text{if } x \in \{H\}. \end{array}$$

Furthermore, for a subset $S \subseteq P \cup Body(P) \cup Head(P)$ we define the *set of bodies* $Bs(S)$ and the *set of heads* $Hs(S)$ *of* S as

$$\begin{array}{l} Bs(S) = S \cap Body(P) \text{ and} \\ Hs(S) = S \cap Head(P), \text{ respectively.} \end{array}$$

With this notations we are able to formulate the following theorem which gives equivalence results between answer sets and a-colorings for the different graphs types.

Theorem 1. *Let P be a logic program, let $\Gamma \in \{BH, RH, BR, B, R, H\}$, let $\Gamma_P = (V, E_0, E_1, E)$ be the Γ-graph of P and let $\mathcal{C}$ be a total coloring of Γ_P. Then we have the following equivalences:*

1. *$\mathcal{C} \in AC(RH_P)$ s.t. $X = Hs(\mathcal{C}_\oplus)$ iff $X \in AS(P)$ s.t. $\mathcal{C}_\oplus = R_P(X) \cup Head(R_P(X))$*
2. *$\mathcal{C} \in AC(BR_P)$ s.t. $X = Head(Rs(\mathcal{C}_\oplus))$ iff $X \in AS(P)$ s.t. $\mathcal{C}_\oplus = B_P(X) \cup R_P(X)$*
3. *$\mathcal{C} \in AC(BH_P)$ s.t. $X = Hs(\mathcal{C}_\oplus)$ iff $X \in AS(P)$ s.t. $\mathcal{C}_\oplus = B_P(X) \cup Head(R_P(X))$.*
4. *$\mathcal{C} \in AC(B_P)$ s.t. $X = Head(\{r \in P \mid B \in Body(r)$ and $B \in \mathcal{C}_\oplus\})$ iff $X \in AS(P)$ s.t. $\mathcal{C}_\oplus = B_P(X)$*
5. *$\mathcal{C} \in AC(R_P)$ s.t. $X = Head(\mathcal{C}_\oplus)$ iff $X \in AS(P)$ s.t. $\mathcal{C}_\oplus = R_P(X)$*
6. *$\mathcal{C} \in AC(H_P)$ iff $\mathcal{C}_\oplus \in AS(P)$.*

This theorem gives not only the mentioned equivalence between the a-colorings of a graph (belonging to different classes) and the answer sets of the associated logic program, but it also tells us how to extract an answer set from a given a-coloring. Principally, this can be done by collecting the "corresponding" heads of nodes in $\mathcal{C}_\oplus$. Next we characterize applicability purely by the structure of Γ-graphs as given in definitions 1 and 2.

Definition 5. *Let P be a program, let $\Gamma \in \{BH, RH, BR, B, R, H\}$, let $\Gamma_P = (V, E_0, E_1, E)$ be the Γ-graph of P, and for $v \in V$ let $G_v = (V^v, E_0^v, E^v)$ be a graph s.t. $V^v \subseteq V$. Then G_v is a* proof graph for v wrt Γ_P *iff the following conditions hold:*

1. *G_v is the 0-subgraph of Γ_P induced by V^v*
2. *G_v is acyclic*
3. *for each $v' \in V^v$ we have $Pred0(v') \subseteq V^v$ and $Pred(v') \cap V^v \neq \emptyset$.*

This definition gives a graph theoretical characterization of groundedness. Whereas groundedness has to be check globally by finding a proof graph by considering 0-arcs, that is positive dependencies between nodes, blockage is a rather local property and can be verified locally by looking at the 1-predecessors of a node.

Theorem 2. *Let $P \in nNLP$ be a normal nested logic program, let $\Gamma_P = (V, E_0, E_1, E)$ be the Γ-graph of P where $\Gamma = BH$, and and let $\mathcal{C}$ be a total coloring of Γ_P. Furthermore, let $v \in V$ be a node of Γ_P. Then v is applicable wrt $Rs(\mathcal{C}_\oplus)$ iff both of the following conditions hold:*

1. *there exists a proof graph (V^v, E_0^v, E^v) for v wrt Γ_P s.t. $V^v \subseteq \mathcal{C}_\oplus$*
2. *for each $v' \in Pred1(v)$ we have $v' \in \mathcal{C}_\ominus$.*

This theorem characterizes applicability of bodies and heads for normal nested logic programs by exclusively appealing to the structure of the body-head-graph BH_P of P. The next two theorems give similar results for classes nLP and nLP_1.

Theorem 3. *Let $P \in nLP$ be a normal logic program, let $\Gamma_P = (V, E_0, E_1, E)$ be the Γ-graph of P where $\Gamma \in \{BH, RH\}$ and and let $\mathcal{C}$ be a total coloring of Γ_P. Furthermore, let $v \in V$ be a node of Γ_P. Then v is applicable wrt $Rs(\mathcal{C}_\oplus)$ iff conditions* 1. *and* 2. *of Theorem 2 hold.*

Theorem 4. *Let $P \in nLP_1$ be a normal logic program with at most one positive body literal, let $\Gamma_P = (V, E_0, E_1, E)$ be the Γ-graph of P where $\Gamma \in \{BH, RH, BR, B, R\}$ and let $\mathcal{C}$ be a total coloring of Γ_P. Furthermore, let $v \in V$ be a node of Γ_P. Then v is applicable wrt $Rs(\mathcal{C}_\oplus)$ iff conditions* 1. *and* 2. *of Theorem 2 hold.*

To sum up, for the program classes nNLP, nLP and nLP_1 the above theorems demonstrate how to characterize applicability of bodies, rules and heads by exclusively referring to the structure of the corresponding graphs. This results form the basis for the suitability results in the next section.

5 Suitability Results

Now we give several suitability results for the different graphs introduced in Section 3. We start by clarifying our intuition about graphs to be suitable for answer set semantics.

Definition 6. *Let $\mathcal{P}$ be a class of logic programs, let $\Gamma : \mathcal{P} \to G_{0,1}$ be a mapping and let $\mathcal{A} : G_{0,1} \to 2^{Node}$ be an operator*[2]*. Then Γ is* suitable (for ASP) for $\mathcal{P}$ wrt $\mathcal{A}$ *iff for each program $P \in \mathcal{P}$ we have $\mathcal{A}(\Gamma(P)) = AC(\Gamma_P)$.*

[2] With 2^{Node} we denote the power set of all nodes of graphs in $G_{0,1}$.

Observe, that operator $\mathcal{A}$ must no use the original program, in order to compute a-colorings. That is, $\mathcal{A}$ uses only the graph assigned to a program. Actually in Definition 6 one could suspect an operator $\mathcal{A}$ which gives the answer sets instead of the a-colorings, because we are mainly interested in them. However, in our setting there are graphs like the rule-graph or the body-rule-graph that do not contain any information on the heads of the corresponding program. For those we need an additional interpretation step which gives the answer set corresponding to some a-coloring. Observe that this step is trivial wrt computation of answer sets in the sense that it can be performed in linear time by looking once at each node of the graph.

Definition 7. *Let $\Gamma \in \{BH, RH, BR, B, R, H\}$, let $\Gamma_{(\cdot)} : nNLP \to G_{0,1}$ be the mapping that assigns the Γ-graph Γ_P to each program P, let $C \in AC(\Gamma_P)$ be an a-coloring of Γ_P and let X_C be a set of atoms. Then X_C is the* answer set corresponding to *C iff one of the following conditions hold:*

1. *if $\Gamma \in \{BH, RH\}$ then $X = Hs(\mathcal{C}_\oplus)$*
2. *if $\Gamma = BR$ then $X = Head(Rs(\mathcal{C}_\oplus))$*
3. *if $\Gamma = B$ then $X = Head(\{r \in P \mid B \in Body(r) \text{ and } B \in \mathcal{C}_\oplus\})$*
4. *if $\Gamma = R$ then $X = Head(\mathcal{C}_\oplus)$*
5. *if $\Gamma = H$ then $X = \mathcal{C}_\oplus$.*

In this definition we clarify how to interpret a-colorings for the different graphs as answer sets of corresponding programs. We use exactly the relations between answer sets and a-colorings as given in Theorem 1.

We have the following results.

Theorem 5. *The mapping $BH_{(\cdot)} : nNLP \to G_{0,1}$ is suitable for nNLP, nLP and nLP_1 wrt $\mathcal{A}_{BH}$.*

For illustration consider the logic programs

$$\begin{array}{l} P_3 = \{r_1 \ : \ x \leftarrow (not\ a); (not\ b).\ \ r_2 \ : \ a \leftarrow not\ b.\ \ r_3 \ : \ b \leftarrow not\ a.\ \} \\ P_4 = \{r_1 \ : \ \ x \leftarrow not\ a, not\ b.\ \ \ r_2 \ : \ a \leftarrow not\ b.\ \ r_3 \ : \ b \leftarrow not\ a.\ \}. \end{array} \tag{2}$$

Observe that program P_3 has answer sets $\{a, x\}$ and $\{b, x\}$ and program P_4 has answer sets $\{a\}$ and $\{b\}$. According to Theorem 5 both programs have different BH-graphs; BH_{P_3} is the graph shown in the middle and BH_{P_3} is the graph on the right side of Figure 3, respectively. In Figure 3, $B_{\overline{a}}$ denotes body $(not\ a)$, $B_{\overline{b}}$ denotes body $(not\ b)$ and $B_{\overline{a,b}}$ denotes body $(not\ a, not\ b)$, repsectively.

This also is reflected by the two different a-colorings of both BH-graphs:

$$\begin{array}{l} \mathrm{AC}(BH_{P_3}) = \{(\{B_{\overline{a}}, a, x\}, \{B_{\overline{b}}, b\}), (\{B_{\overline{b}}, b, x\}, \{B_{\overline{a}}, a\})\} \\ \mathrm{AC}(BH_{P_4}) = \{(\{B_{\overline{a}}, a\}, \{B_{\overline{b}}, b, B_{\overline{a,b}}, x\}), (\{B_{\overline{b}}, b\}, \{B_{\overline{a}}, a, B_{\overline{a,b}}, x\})\}. \end{array}$$

Although P_3 and P_4 have different answer sets both programs have the same RH-graph, which is depicted on the left hand side of Figure 3. Obviously, we cannot give any a-coloring of RH_{P_i} $(i \in \{3, 4\})$ corresponding to some answer set of P_i without referring to the original program. Therefore, we have the following result.

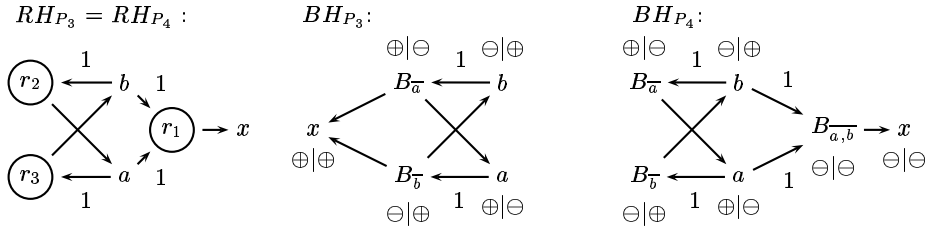

Fig. 3. The RH-graph of programs P_3 and P_4 (left) as well as the BH-graph of programs P_3 (middle) and P_4 (right).

Theorem 6. *The mapping $\Gamma_{(.)} : nNLP \to G_{0,1}$ is* **not** *suitable for nNLP wrt $\mathcal{A}_\Gamma$ for each $\Gamma \in \{B, R, H, BR, RH\}$.*

However, RH-graphs are suitable for normal logic programs.

Theorem 7. *The mapping $RH_{(.)} : nLP \to G_{0,1}$ is suitable for nLP and nLP_1 wrt $\mathcal{A}_{RH}$.*

Next we demonstrate the difference between RH- and BR-graphs by programs P_1 and P_2 from Section 1. Clearly, the RH-graphs of P_1 and P_2 are different, because $Head(P_1)$ and $Head(P_2)$ are different. That is, d is a node of RH_{P_1} but it is no node of RH_{P_2}. For this reason the RH-graph is able to distinguish the semantic differences between P_1 and P_2 (see Figure 2 in Section 3). For the BR-graph we obtain a different situation, because both of the above programs possess the same BR-graph. It is given in Figure 4, where $B_{\overline{b}}$ denotes body

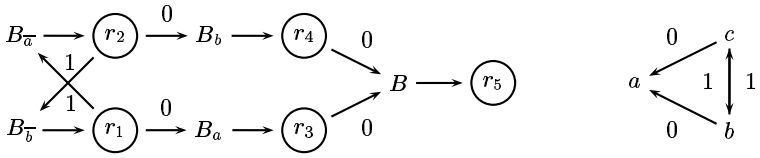

Fig. 4. The body-rule-graph (left) and the head-graph (right) of programs P_1 and P_2.

(*not* b) of rule r_1, $B_{\overline{a}}$ denotes body (*not* a) of rule r_2, B_a denotes body (a) of rule r_3, B_b denotes body (b) of rule r_4 and B denotes one of the bodies (d, c) or (c) of rule r_5 depending on which program we are interested in P_1 or P_2. We have the following theorem.

Theorem 8. *The mapping $\Gamma_{(.)} : nLP \to G_{0,1}$ is* **not** *suitable for nLP wrt $\mathcal{A}_\Gamma$ for each $\Gamma \in \{BR, B, R, H\}$.*

If we focus on normal programs with at most one positive body literal, we obtain suitability for Γ-graphs for $\Gamma \in \{BR, B, R\}$.

Theorem 9. *The mapping $\Gamma_{(\cdot)} : nLP_1 \to G_{0,1}$ is suitable for nLP_1 wrt $\mathcal{A}_\Gamma$ for each $\Gamma \in \{BR, B, R\}$.*

Surprisingly, head-graphs are not even suitable for nLP_1. Consider the following programs

$$\begin{array}{l} \{\ a \leftarrow b, c. \qquad\quad b \leftarrow not\ c.\ \ c \leftarrow not\ b.\ \ \} \\ \{\ \ a \leftarrow b.\ \ \ a \leftarrow c.\ \ b \leftarrow not\ c.\ \ c \leftarrow not\ b.\ \ \}. \end{array} \tag{3}$$

They both have the same head-graph, which is depicted on the right side of Figure 4. Hence we have the final result.

Theorem 10. *The mapping $H_{(\cdot)} : nLP \to G_{0,1}$ is* **not** *suitable for nLP wrt $\mathcal{A}_H$.*

All suitability results are summarized in Table 1.

6 Related Work and Discussion

In the literature many different types of graphs are associated with a given logic program. Obviously, those graphs are used to detect structural properties of programs, such as stratification [3], existence of answer sets [10, 4], or the actual characterization of answer set semantics or well-founded semantics [7, 4, 14, 16]. The usage of rule-oriented dependency graphs is common to [7, 4, 14, 11]. In fact, the coloration of such graphs for characterizing answer sets was independently developed in [4] and [14] and further investigated in [11]. However, as the two normal logic progrmas P_1 and P_2 demonstrate, rule dependency graphs (block graphs) are not suitable for computing answer sets of normal logic programs. Therefore, the approaches in [7, 4] rely on translations of normal programs into negative ones before using their respective dependency graphs for characterizing answer sets. Clearly, the above problem disappears for negative programs. In the `noMoRe` system [1] and its underlying theory [16, 11, 15] some kind of additional meta-information not present in the graph structure is neccessary to overcome the mentioned problem (see Section 1). Obviously, non of the graphs discussed so far is suitable for normal nested logic programs[3].

The main contribution of this work is the introduction of body-head-graphs which are shown to be suitable for normal nested logic programs. Hence body-head-graphs handle multiple positive body atoms in an elegant and correct way for normal programs. Furthermore, they also handle disjunctions of "normal" bodies correctly. To the best of our knowledge, this is the first time that graphs corresponding to normal nested programs are introduced and used for characterizing answer sets. As a byproduct the application of transformations utilized in [15] in order to replace all rules with same head and all rules with same body by just one nested normal rule, respectively, comes for free when using body-head-graphs. That is, body-head-graphs give a much more compact representation than all other mentioned graphs. Hence they should be preferred for

[3] Observe, that normal nested logic programs are also utilized in [17] for translating nested programs into extended ones.

representing logic programs. In fact, we have generalized a-colorings [14, 16] to all graphs introduced in Section 3, in order to define suitability of graphs for answer set semantics. This can be done because of two reasons. First, answer sets and a-colorings are equivalent (see Theorem 1) and second, by using the "correct" graphs (which are the suitable ones later, see Theorems 2,3 and 4), the global concept of groundedness can be characterized by exclusively referring to those graphs. Consequently, applicability can be checked by exclusively investigating the structure of corresponding graphs.

Observe that, the other graph formalisms for logic programs mentioned above are only defined for normal [2, 14, 11] or normal negative programs [7, 4]. As show in [5], extended dependency graphs [4] are suitable for kernel programs, where kernel programs form a syntactically restricted but equivalent (wrt answer set semantics) subclass of nLP which do not have any positive body atoms. Each head atom of a kernel program must also appear as a body atom of the program and all atoms are undefined in the well-founded model. As stated in [4], every normal program is equivalent to some kernel program. Observe, that EDG and the rule dependency graph (block graph) of a kernel program are isomorphic. Here we have generalized [5] in two important aspects. First, we have developed our concepts directly for each class of programs $\Gamma \in \{\mathrm{nNLP}, \mathrm{nLP}, \mathrm{nLP}_1\}$. That is, we do not use any program transformations to equivalent but syntactically restricted programs. In this way we also generalize [7] where rule graphs are defined for reduced negative programs. Another difference between [4] and our approach is that we abstract from actual atom names in the definition of suitable graphs (see Definition 6) whereas in [4] atom names (as label of nodes) are used to obtain different graphs for semantically different programs.

Finally, one may ask whether we do not have investigated rule-body-head-graphs in this context. The reason is that distinguishing rules and bodies does not give much more information on the underlying logic program. This is reflected in Theorem 5 where it is shown that body-head graphs are sufficient to deal with normal nested programs; and also Theorem 8 gives an argument for not introducing bodies and rules in the same graph, since it shows that body-rule-graphs are even not suitable for normal logic programs. In fact body-rule-graphs give not more information than rule-graphs or body-graphs (see Theorem 9) and are only useful for characterizing answer sets for normal programs with at most one positive body atom. All suitability results are summarized in Table 1, where each line gives the suitability of some graph wrt program classes nNLP, nLP and nLP_1. Currently we are implementing body-head-graphs in the `noMoRe` system.

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.

	nNLP	nLP	nLP_1
BH	Yes	Yes	Yes
RH	No	Yes	Yes
BR	No	No	Yes
R	No	No	Yes
B	No	No	Yes
H	No	No	No

Table 1. Suitability of $\Gamma \in \{BH, RH, BR, B, R, H\}$ for program classes nNLP, nLP and nLP_1.

References

1. C. Anger, K. Konczak, and T. Linke. `NoMoRe`: A system for non-monotonic reasoning under answer set semantics. In W. Faber T. Eiter and M. Truszczyński, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, pages 406–410. Springer, 2001.
2. K. R. Apt and R. N. Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19/20:9–71, 1994.
3. 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.
4. 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.
5. S. Costantini. Comparing different graph representations of logic programs under the answer set semantics. In T. C. Son and A. Provetti, editors, *Proc. of the AAAI Spring 2001 Symposium on: Answer Set Programming, Towards Efficient and Scalable Knowledge Representation and Reasoning*, Stanford, CA, USA, 2001. AAAI Press.
6. S. Costantini, O. D'Antona, and A. Provetti. On the equivalence and range of applicability of graph-based representations of logic programs. *Information Processing Letters*, 84(5):241–249, 2002.
7. Y. Dimopoulos and A. Torres. Graph theoretical structures in logic programs and default theories. *Theoretical Computer Science*, 170:209–244, 1996.
8. D. East and M. Truszczyński. dcs: An implementation of datalog with constraints. In *Proceedings of the National Conference on Artificial Intelligence*. MIT Press, 2000.
9. 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.
10. F. Fages. Consistency of clark's completion and existence of stable models, 1992. Volume 1 of *Journal of Methods of Logic in Computer Science*, pages 51–60. The MIT Press, 1994.
11. K. Konczak, T. Linke, and T. Schaub. Graphs and colorings for answer set programming. 2003. Submitted to LPNMR.

12. V. Lifschitz. Answer set planning. In *Proceedings of the 1999 International Conference on Logic Programming*, pages 23–37. MIT Press, 1999.
13. V. Lifschitz, L. Tang, and H. Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389, 1999.
14. T. Linke. Graph theoretical characterization and computation of answer sets. In B. Nebel, editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 641–645. Morgan Kaufmann Publishers, 2001.
15. T. Linke. Using Nested Logic Programs for Answer Set Programming. 2003. Submitted to LPNMR.
16. T. Linke, C. Anger, and K. Konczak. More on nomore. In G. Ianni and S. Flesca, editors, *Eighth European Workshop on Logics in Artificial Intelligence (JELIA '02)*, volume 2424 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2002.
17. 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.