Construction of Explanation Graphs from Extended Dependency Graphs for Answer Set Programs

Ella Albrecht, Patrick Krümpelmann(✉), and Gabriele Kern-Isberner

Technische Universität Dortmund, Dortmund, Germany
patrick.kruempelmann@cs.tu-dortmund.de

Abstract. Extended dependency graphs are an isomorphic representation form for Answer Set Programs, while explanation graphs give an explanation for the truth value of a literal contained in an answer set. We present a method and an algorithm to construct explanation graphs from a validly colored extended dependency graph. This method exploits the graph structure of the extended dependency graph to gradually build up explanation graphs. Moreover, we show interesting properties and relations of the graph structures, such as loops, and we consider both answer set and well-founded semantics. We also present two different approaches for the determination of assumptions in an extended dependency graph, an optimal but exponential and a sub-optimal but linear one.

1 Introduction

Graphs are an excellent tool for the illustration and understanding of non-monotonic reasoning formalisms, and for the determination and explanation of models. For answer set programs two graph based representations have recently been proposed: Extended dependency graphs (EDG) [2] and explanation graphs (EG) [1]. EDGs are an isomorphic representation of extended logic programs and use a coloring of the nodes to determine answer sets. Explanation graphs, on the other hand, provide an explanation for the appearance of a single literal in an answer set. In [1] it was conjectured that there is a strong relation between a validly colored extended dependency graph and an explanation graph. In this work we present a method to construct explanation graphs from a successfully colored extended dependency graph and prove its correctness. The way of proceeding exploits the structure of the EDG and the fact that explanation graphs can be built up gradually from smaller sub-explanation graphs.

In [1] assumptions are introduced, which describe literals whose truth value has to be guessed during the determination process of answer sets, but there is actually no appropriate method given to find proper assumptions. We present two systematic approaches which extract assumptions from an EDG. This is the most difficult part of the construction of EGs, since intra-cyclic as well as inter-cyclic dependencies between nodes have to be considered. The first approach makes use of basic properties of assumptions and the graph to reduce the size

M. Hanus and R. Rocha (Eds.): KDPD 2013, LNAI 8439, pp. 1–16, 2014.
DOI: 10.1007/978-3-319-08909-6_1,

of assumptions in linear runtime. The second approach exploits cycle structures and their interdependencies to determine the minimal assumptions, which comes with the cost of exponential runtime.

In Sect. 2 we give an introduction to answer set programming and in Sect. 3 we present extended dependency graphs and explanation graphs. Section 4 deals with the construction process of the EGs from a validly colored EDG. The fifth section deals with the different approaches of finding proper assumptions in an EDG.

2 Answer Set Programming

We consider extended logic programs under the answer set semantics [3]. An *extended logic program* P is a set of rules r of the form $r: \ h \leftarrow a_1, ..., a_n, not\ b_1, ..., not\ b_n.$ where $h, a_1, ...a_m, b_1, ..., b_n$ are literals. A *literal* may be of the form x or $\neg x$ where x is a propositional symbol called *atom* and $\neg$ is the classical negation. $head(r) = \{h\}$ denotes the head, $pos(r) = \{a_1, ..., a_m\}$ denotes the positive, and $neg(r) = \{b_1, ..., b_n\}$ the negative body literals of a rule. The *Herbrand base* $\mathcal{H}(P)$ of a logic program P is the set of all grounded literals of P. A literal is *grounded*, if it does not contain a variable. In this work, we assume that the logic programs are grounded, i.e., every literal appearing in the program is grounded.

Let $M \subseteq \mathcal{H}(P)$ be a consistent set of literals, i.e., M does not contain any complementary literals. The Gelfond-Lifschitz reduct of a program P is the program $P^M = \{h \leftarrow a_1, ..., a_n. \mid h \leftarrow a_1, ..., a_n, not\ b_1, ..., not\ b_m. \in P, \{b_1, ..., b_n\} \cap M = \emptyset\}$ that is obtained by removing all rules where a literal $b_i \in M$ appears as a negative body literal, and removing all negative body literals from the remaining rules. M is *closed* under P^M if $head(r) \in M$ whenever $pos(r) \subseteq M$ for every rule $r \in P^M$. M is an *answer set* for P if M is closed under P^M and M is minimal w.r.t. set inclusion.

Answer set semantics may yield multiple models resp. answer sets. Another semantics for logic programs is the well-founded semantics [6]. Its basic idea is that there exist literals which have to be true with certainty and literals which have to be false with certainty. Under the answer set semantics such information gets lost if no answer set exists.

For a logic program P and a logic program P^+ that we get if we remove all rules with negative body literals, the sequence $(K_i, U_i)_{i \geq 0}$ is defined as $K_0 = lfp(T_{P^+,\emptyset})U_0 = lfp(T_{P,K_0})K_i = lfp(T_{P,U_{i-1}})U_i = lfp(T_{P,K_i})$ where $T_{P,V}(S) = \{a \mid \exists r \in P: \ head(r) = a, \ pos(r) \subseteq S, \ neg(r) \cap V = \emptyset\}$. The well-founded model is $WF_P = \langle W^+, W^- \rangle$ where $W^+ = K_j$ is the *well founded* set and $W^- = \mathcal{H}(P) \setminus U_j$ is the *unfounded* set, with j being the first index with $\langle K_j, U_j \rangle = \langle K_{j+1}, U_{j+1} \rangle$. Literals that are neither contained in W^+ nor in W^- are called *undefined*.

3 Graphs for Answer Set Programs

We introduce two types of graphs, the first graph type is the extended dependency graph [2].

Definition 1 (Extended dependency graph). *The* extended dependency graph *(EDG) to a logic program P with its Herbrand base $\mathcal{H}(P)$ is a directed graph $EDG(P) = (V, E)$ with node set $V \subseteq \mathcal{H}(P)$ and edge set $E \subseteq V \times V \times \{+, -\}$, according to the following rules:*

V1 *There is a node a_i^k for every rule $r_k \in P$ where $head(r_k) = a_i$.*
V2 *There is a node a_i^0 to every atom $a_i \in \mathcal{H}(P)$ which does not appear as the head of a rule.*
E1 *There is an edge $(c_j^l, a_i^k, +)$ for every node $c_j^l \in V$ iff there is a rule $r_k \in P$ where $c_j \in pos(r_k)$ and $head(r_k) = a_i$.*
E2 *There is an edge $(c_j^l, a_i^k, -)$ for every node $c_j^l \in V$ iff there is a rule $r_k \in P$ where $c_j \in neg(r_k)$ and $head(r_k) = a_i$.*

To every logic program a unique EDG can be constructed, this means a logical program is isomorphic to its representation as an EDG in the sense that the structure of the program is reflected one-to-one by the structure of the EDG. Properties of a logic program can be obtained from properties of the corresponding EDG and vice versa. Colorings of the graph that comply with the semantics of the edges are called valid colorings and correspond to answer sets of a logic program. A green colored node represents a successfully deduced head of a rule, and a red colored node represents a head of a rule that can not be deduced. A literal is contained in the answer set if there exists a node that represents the literal which is colored green. A literal is not contained in the answer set if all nodes corresponding to the literal are colored red.

Definition 2 (Valid coloring of an EDG). *Let a program P be given. A coloring $\nu : V \to \{green, red\}$ of the graph $EDG(P) = (V, E)$ is* valid*, if the following conditions holds:*

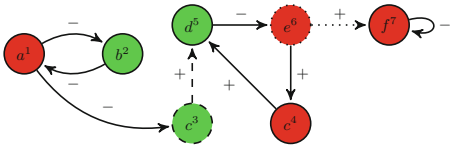
1. *$\forall i, k$ where $k \geq 1$, $\nu(a_i^k) = green$ if a_i^k has no incoming edge.*
2. *$\forall i, k$, $\nu(a_i^k) = green$ if the following two conditions are met:*
   (a) $\forall j, m$ where $(a_j^m, a_i^k, +) \in E$, $\exists a_j^h \in V$ such that $\nu(a_j^h) = green$
   (b) $\forall j, m$ where $(a_j^m, a_i^k, -) \in E$, $\nu(a_j^m) = red$
3. *$\forall i, k$ $\nu(a_i^k) = red$, if at least one of the two following conditions is met:*
   (a) $\exists j, m$ where $(a_j^m, a_i^k, +) \in E$ and $\forall a_j^h \in V$, $\nu(a_j^h) = red$
   (b) $\exists j, m$ where $(a_j^m, a_i^k, -) \in E$ and $\nu(a_j^m) = green$
4. *For every positive cycle C where $\nu(a_i^k) = green$ for all $a_i^k \in C$ the following condition holds: There is an i and $l \neq k$, such that $\nu(a_i^l) = green$.*

A main feature of EDGs is their representation of cycles and handles for those. A cycle consists of several literals that are dependent in a cyclic way, e.g., given the two rules $r_1 : a \leftarrow b.$ and $r_2 : b \leftarrow a.$, both literals a and b are interdependent in a cyclic way. Generally, cycles can be connected in two different ways to the rest of the program:

– *OR-handle*: Let the rule $r_1 : \ a \leftarrow \beta.$ be part of a cycle where β may be of the form b or *not* b. If there exists another rule $r_2 : \ a \leftarrow \delta$ where δ may be of the form d or *not* d, then δ is an OR-handle for the cycle to which r_1 belongs. The OR-handle is called active if δ is true.

$P_1 := \{a \leftarrow not\ b.$
$b \leftarrow not\ a.$
$c \leftarrow \{not\ a\}.$
$c \leftarrow e.$
$d \leftarrow c.$
$e \leftarrow not\ d.$
$f \leftarrow [e],\ not\ f.\}$

(a) Logic program P_1



(b) Successfully colored EDG to program P_1

Fig. 1. A logic program, the corresponding EDG, and a valid coloring

– *AND-handle*: If a rule r is part of a cycle and has an additional condition γ, that means the rule is of the form $r:\ a \leftarrow \beta, \gamma$ where γ may be of the form c or *not* c, then γ is an AND-handle. The AND-handle is called active if γ is false.

An extended dependency graph extends a normal dependency graph in so far that it distinguishes between AND- and OR-handles.

Example 1. *Figure 1 shows a logic program (a) and the corresponding EDG (b). The EDG has a valid coloring which represents the answer set* $\{b, c, d\}$*. The AND-handle is marked in the program with* [ ] *and is dotted in the EDG. The OR-handle is marked in the program with* { } *and is dashed in the EDG.*

The second graph type is the explanation graph (EG). In contrast to the EDGs, which visualize the structure of a whole logic program, EGs provide an explanation for why a single literal appears or does not appear in an answer set and is always constructed with regard to an answer set and a set of assumptions. Assumptions are literals for which no explanation is needed since their value is assumed. All literals that are qualified for being used as assumptions are called *tentative assumptions* and are formally defined as follows:

Definition 3 (Tentative Assumptions). *Let* P *be a logic program,* M *an answer set of* P *and* $WF = \langle WF^+, WF^- \rangle$ *the well-founded model of* P*. Then the set of* tentative assumptions *of* P *w.r.t.* M *is*

$$\mathcal{TA}_P(M) = \{a \mid a \in NANT(P) \text{ and } a \notin M \text{ and } a \notin WF^+ \cup WF^-\}$$

where $NANT(P)$ *is the set of all literals appearing in* P *as a negative body literal:* $NANT(P) = \{a \mid \exists r \in P:\ a \in neg(r)\}$*.*

Given a logic program P and a subset $U \subseteq \mathcal{TA}_P(M)$ of tentative assumptions, one can obtain the negative reduct $NR(P, U)$ of a program P w.r.t. U by removing all rules where $head(r) \in U$.

Definition 4 (Assumption). *An* assumption *of a program* P *regarding an answer set* M *is a set* $U \subseteq \mathcal{TA}_p(M)$ *where the well-founded model of the negative reduct corresponds to the answer set* M*, i. e.* $WF_{NR(P,U)} = \langle M, \mathcal{H}(P) \setminus M \rangle$*.*

This means that setting all literals of the assumption U to *false* leads to all literals being defined in the well-founded model.

Explanation graphs are based on local consistent explanations (LCE). These are sets of literals which directly influence the truth value of a literal a. For a literal a that is contained in the answer set and a rule where a is the head and all conditions of the rule are fulfilled, i.e., all body literals are true, the LCE consists of all body literals of the rule. Since there may exist several fulfilled rules with a as their head, a can also have various LCEs. For a literal a that is not contained in the answer set, an LCE is a minimal set of literals that together falsify all rules that define a. For this purpose the LCE has to contain one falsified condition from each rule.

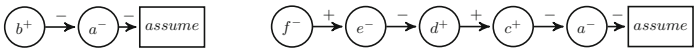
Definition 5 (Local Consistent Explanation). *Let a program P be given, let a be a literal, let M be an answer set of P, let U be an assumption and let $S \subseteq \mathcal{H}(P) \cup \{not\ a \mid a \in \mathcal{H}(P)\} \cup \{assume, \top, \bot\}$ be a set of (default-negated) literals and justifying symbols.*

1. *S is an* LCE *for a^+ w.r.t. (M, U), if $a \in M$ and*
   - *$S = \{assume\}$ or*
   - *$S \cap \mathcal{H}(P) \subseteq M$, $\{c \mid not\ c \in S\} \subseteq (\mathcal{H}(P) \setminus M) \cup U$ and there exists a rule $r \in P$ where $head(r) = a$ and $S = body(r)$. For the case that $body(r) = \emptyset$ one writes $S = \{\top\}$ instead of $S = \emptyset$.*
2. *S is an* LCE *for a^- w.r.t. (M, U), if $a \in (\mathcal{H}(P) \setminus M) \cup U$ and*
   - *$S = \{assume\}$ or*
   - *$S \cap \mathcal{H}(P) \subseteq (\mathcal{H}(P) \setminus M) \cup U$, $\{c \mid not\ c \in S\} \subseteq M$ and S is a minimal set of literals, such that for every $r \in P$ the following holds: if $head(r) = a$ then $pos(r) \cap S \neq \emptyset$ or $neg(r) \cap \{c \mid not\ c \in S\} \neq \emptyset$. For the case S being the empty set one writes $S = \{\bot\}$.*

In an EDG an edge (a_i^k, a_j^l, s) with $s \in \{+, -\}$ means that the truth value of literal a_j depends on the truth value of literal a_i. In an explanation graph the edges are defined the other way round, so that an edge (a_i, a_j, s) with $s \in \{+, -\}$ means that a_j explains or supports the truth value of a_i. A node in an explanation graph is either annotated with $+$ or $-$, depending on whether the literal is contained in the answer set or not. We define two sets of annotated literals $\mathcal{H}^p = \{a^+ \mid a \in \mathcal{H}(P)\}$ and $\mathcal{H}^n = \{a^- \mid a \in \mathcal{H}(P)\}$. Furthermore we define $literal(a^+) = a$ and $literal(a^-) = a$. The *support* of a node a_i in an EG is the set of all direct successors of a_i in the EG and is formally defined as follows:

Definition 6 (Support). *Let $G = (V, E)$ be a graph with node set $V \subseteq \mathcal{H}^p \cup \mathcal{H}^n \cup \{assume, \top, \bot\}$ and edge set $E \subseteq V \times V \times \{+, -\}$. Then the* support *of a node $a \in V$ w.r.t. G is defined as:*

- *$support(a, G) = \{literal(c) \mid (a, c, +) \in E\} \cup$*
  $\{not\ literal(c) \mid (a, c, -) \in E\}$,
- *$support(a, G) = \{\top\}$ if $(a, \top, +) \in E$,*
- *$support(a, G) = \{\bot\}$ if $(a, \bot, -) \in E$ or*
- *$support(a, G) = \{assume\}$ if $(a, assume, s) \in E$ where $s \in \{+, -\}$.*



(a) EG for b (b) EG for f

Fig. 2. Explanation graphs for literals b and f in program P_1 w.r.t. answer set $M = \{b, d, c\}$ and assumption $U = \{a\}$

Definition 7 (Explanation graph). *An* explanation graph *for a literal $a \in \mathcal{H}^p \cup \mathcal{H}^n$ in a program P w.r.t. an answer set M and an assumption $U \in Assumptions(P, M)$ is a directed graph $G = (V, E)$ with node set $V \subseteq \mathcal{H}^p \cup \mathcal{H}^n \cup \{assume, \top, \bot\}$ and edge set $E \subseteq V \times V \times \{+, -\}$, such that the following holds:*

1. *The only sinks in the graph are $assume$, $\top$ and $\bot$, where $\top$ is used to explain facts of the program P, $\bot$ is used to explain literals which do not appear as a head of any rule and $assume$ is used to explain literals for which no explanations are needed since their value is assumed to be false.*
2. *If $(c, l, s) \in E$ where $l \in \{assume, \top, \bot\}$ and $s \in \{+, -\}$, then (c, l, s) is the only outgoing edge for every $c \in V$.*
3. *Every node $c \in V$ is reachable from a.*
4. *For every node $c \in V \setminus \{assume, \top, \bot\}$ the support $support(c, G)$ is an LCE for c regarding M and U.*
5. *There exists no $c^+ \in V$, such that $(c^+, assume, s) \in E$ where $s \in \{+, -\}$.*
6. *There exists no $c^- \in V$, such that $(c^-, assume, +) \in E$.*
7. *$(c^-, assume, -) \in E$ iff $c \in U$.*

Example 2. *Figure 2 shows the explanation graphs for literals b and f from the program in Fig. 1a w.r.t. the answer set $M = \{b, d, c\}$ and assumption$= \{a\}$.*

4 Construction of Explanation Graphs

In this section we introduce an approach for the construction of explanation graphs by extracting the required information from a validly colored extended dependency graph. Suppose we are given an extended dependency graph $G = (V, E)$ with a valid coloring $\nu : V \to \{green, red\}$. In the first step, we clean up the EDG by removing irrelevant edges and nodes. Irrelevant edges and nodes are those edges and nodes that do not have influence on the appearance or non-appearance of a literal in the answer set. This means they do not provide an explanation for a literal and hence are not needed for any explanation graph.

Definition 8 (Irrelevant edge, irrelevant node). *An edge (a_i^k, a_j^l, s) is* irrelevant *if*

- $\nu(a_i^k) = green$, $\nu(a_j^l) = green$ *and* $s = -$,
- $\nu(a_i^k) = green$, $\nu(a_j^l) = red$ *and* $s = +$,

- $\nu(a_i^k) = red$, $\nu(a_j^l) = green$ *and* $s = +$ *or*
- $\nu(a_i^k) = red$, $\nu(a_j^l) = red$ *and* $s = -$.

A node a_i^k *is* irrelevant *if* $\nu(a_i^k) = red$ *and there exists* $l > 0$ *where* $\nu(a_i^l) = green$.

If an irrelevant node is removed, all its incoming and outgoing edges are also removed. After removing irrelevant edges and nodes we get an EDG $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$. In the second step, nodes are gradually marked in the EDG. The marking process starts at nodes which have no incoming edges, because the explanation graphs for these nodes do not depend on other nodes. Every time a node is marked, the explanation graphs for the marked node are built. For this purpose five types of transformations are defined. The two first transformations describe the construction of explanation graphs for simple nodes, i.e., nodes which have no incoming edges in the EDG. The third and fourth transformations describe the construction of nodes which are dependent on other nodes, i.e., have incoming edges, distinguished by the color of the nodes. The last transformation is used for the construction of EGs for literals that are used as assumptions.

Transformation 1 (Transformation of fact nodes). *The EG for a node* a_i^k *which has no incoming edges and satisfies* $\nu(a_i^k) = green$ *consists of a node* a_i^+, *a node* $\top$ *and an edge* $(a_i^+, \top, +)$ *(Fig. 3a), because such a node corresponds to a fact of the logic program.*

Transformation 2 (Transformation of unfounded nodes). *The EG to a node* a_i^k *which has no incoming edges and satisfies* $\nu(a_i^k) = red$ *consists of a node* a_i^-, *a node* $\bot$ *and an edge* $(a_i^-, \bot, -)$ *(Fig. 3b).*

After marking nodes without incoming edges, we can mark nodes in positive cycles (cycles that contain only positive edges) that do not have an active handle, since the corresponding literals do not have a supportive justification and are unfounded in the well-founded model. Since there exists no active handle for the cycle, there is no other explanation for the nodes of the cycle than the one consisting of the cycle itself (with reversed edges). Now we continue marking nodes using the following rules until no more nodes can be marked:

A green node a_i^k can be marked if

- for all a_j^l where $(a_j^l, a_i^k, +) \in E'$, there exists $n \geq 1$ with $a_j^n \in V'$, such that a_j^n is marked, and
- for all nodes a_j^l where $(a_j^l, a_i^k, -) \in E'$, a_j^l is marked.

That means that a green node can be marked, if all its predecessor nodes are marked. For literals which are represented by multiple green nodes, it is sufficient if one of these nodes is marked.

A red node a_i^k can be marked if

- $\exists (a_j^l, a_i^k, -) \in E'$ where a_j^l is marked, or
- $\exists (a_j^l, a_i^k, +) \in E'$ where for all $n \geq 0$, $a_j^n \in V'$ is marked.

That means that a red node can be marked, if at least one of its predecessor nodes is marked. In case that a predecessor literal is represented by multiple red nodes, all these nodes have to be marked.

Lemma 1. *The well-founded set W^+ corresponds to the set of all marked green nodes and the unfounded set W^- corresponds to the set of all marked red nodes.*

Proof sketch. It has to be shown that K_i always contains marked green nodes and $X_i = \mathcal{H}(P) \setminus U_i$ always contains marked red nodes where K_i and U_i are the sets that are generated during the calculation of the well-founded model (see Page 2). For this purpose the fixpoint operator $T_{P,V}$ for the generation of K_i and X_i has to be adjusted to X_i instead of U_i, especially in the adjustment of the operator for X_i positive cycles have to be considered. Then it can be seen, that the resulting operators exactly describe the process of marking green resp. red nodes.

From Lemma 1 we get the following proposition:

Proposition 1. *All unmarked nodes are undefined in the well-founded model.*

Green nodes represent literals that are contained in the answer set. So the local consistent explanation for such a node consists of all direct predecessor nodes (resp. the literals they represent).

Transformation 3 (Transformation of dependent green nodes). *Let $ie(a_i^k)$ be the set of incoming edges of node a_i^k. An EG to a node a_i^k where $\nu(a_i^k) = green$ and $ie(a_i^k) \neq \emptyset$ consists of a node a_i^+ and edges $E_{EG} = \{(a_i^+, EG(a_j), s) \mid (a_j^l, a_i^k, s) \in E'\}$, where $EG(a_j)$ is an explanation graph for a_j (Fig. 3d).*

Red nodes represent literals that are not contained in the answer set. In most cases red nodes have only active edges. The only exception is if a predecessor literal a_j of a red node a_i^k is represented by multiple red nodes, formally $|\{a_j^l \mid (a_j^l, a_i^k, -) \in E'\}| \geq 2$. To get the LCEs for literals represented by a red node, all nodes representing this literal have to be considered. Each node represents a rule where the incoming edges represent the conditions of the rule. An LCE has to contain exactly one violated condition from each rule. Since we have removed all irrelevant edges, every edge represents a violated condition. That means that an LCE contains exactly one incoming edge for every node representing the literal.

Definition 9 (Local consistent explanation in an EDG). *Let $pd(a_i^k) = \{a_j \mid (a_j^l, a_i^k, s) \in E', s \in \{+, -\}\}$ be the set of the predecessor literals of node a_i^k and a_i^1 to a_i^n the nodes representing a literal a_i. We set $L(a_i) = \{\{b_1, ..., b_n\} \mid b_1 \in pd(a_i^1), ..., b_n \in pd(a_i^n)\}$. $L(a_i)$ is an LCE for a_i if $L(a_i)$ is minimal w.r.t. set inclusion.*

Transformation 4 (Transformation of dependent red nodes). *The explanation graph for a node a_i^k where $\nu(a_i^k) = red$ w.r.t. an LCE $L(a_i)$ consists of a node a_i^- and edges*

$$E_{EG} = \{(a_i^-, EG(a_j), s \mid a_j \in L, (a_j^l, a_i^k, s) \in E' \text{ for any } l, k\} \text{ (Fig. 3e).}$$

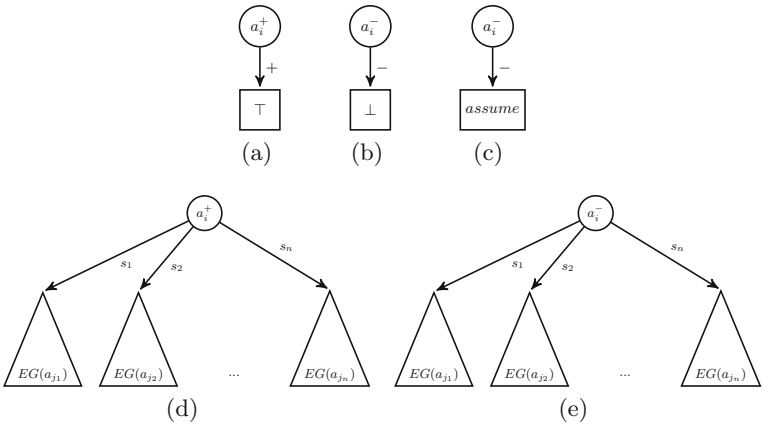


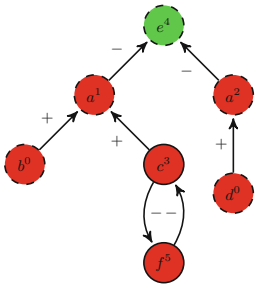
Fig. 3. Templates for constructing an explanation graph for a_i

As mentioned before, every time a node is marked explanation graphs are constructed. It should be remarked that not all explanation graphs for a literal can be created when a node is reached for the first time and marked. This follows from the fact that there can exist multiple nodes for one literal and that a red node can be already marked if one of its predecessors is marked.

Example 3. *In the graph from Fig. 4b for the logic program P_2 (Fig. 4a) we can see that although both nodes are marked, we cannot construct all explanation graphs to the literal a and also e, since e depends on a. The explanation graph for the LCE $\{c, d\}$ of a is missing, because c depends on a cycle where an assumption has to be determined. So, if the node a^1 is reached again by the other edge $(c^3, a^1, +)$ during the marking process, the set of its explanation graphs has to be updated and the information must be propagated to all successor nodes.*

$P_2 := \{a \leftarrow b, c.$
$a \leftarrow d.$
$c \leftarrow not\ f.$
$e \leftarrow not\ a.$
$f \leftarrow not\ c.\}$

(a) Logic program P_3



(b) Successfully colored EDG for program P_2

Fig. 4. A logic program and the corresponding EDG

The next step is to determine assumptions. Different approaches for choosing assumptions are proposed in Sect. 4. After choosing the assumption U, the incoming edges of all nodes representing a literal from U are removed, because no other explanations for these literals are allowed. So we get an extended dependency graph $G'' = (V', E'')$ where $E'' \subseteq E'$. The nodes of these literals are marked and the explanation graphs can be constructed.

Transformation 5 (Transformation of assumption nodes). *The explanation graph to an assumption node a_i^k consists of a node a_i^-, a node "assume" and an edge $(a_i^-, assume, -)$ (Fig. 3c).*

Then we proceed as before, marking nodes in G'' and simultaneously constructing the explanation graphs.

5 Choosing Assumptions

In most cases it is desirable to choose as few literals as possible as assumption. Assumptions where no literal can be removed without the set being no assumption anymore, are called *minimal assumptions*. Finding them in an EDG can be very complex, since all dependencies between the unmarked cycles have to be considered. For this purpose two different approaches are presented in this section. The first approach does not consider dependencies between cycles, so that assumptions can be computed in $O(|V| + |E|)$. The disadvantage of this approach is that the determined assumptions are not minimal in most cases. The second approach determines all minimal assumptions of an EDG, but has an exponential complexity.

For the determination of assumptions we first have to determine all tentative assumptions. From the definition of tentative assumptions we know that a tentative assumption has to meet three conditions: (a) it must not be contained in the answer set, (b) it has to appear as negative body literal in a rule and (c) it has to be undefined by the well-founded model. Transferred to a node in an EDG G' where all irrelevant edges and nodes were removed and all nodes are marked as far as possible, the first condition is fulfilled exactly by red nodes, the second condition is fulfilled exactly by nodes which have outgoing negative edges in the original graph G and the third condition is exactly fulfilled by unmarked nodes:

$$\mathcal{TA}(G') = \{a_i \mid a_i^k \in V', \nu(a_i^k) = red, a_i^k \text{ not marked}, (a_i^k, a_j^l, -) \in E\}.$$

The set of tentative assumptions is always an assumption as shown in [1]. Since the set of tentative assumptions is often large, we are looking for an approach to reduce the set. The aim is that after defining the assumption, all other nodes can be marked.

Lemma 2. *In a graph without any cycles all nodes can be marked without making assumptions.*

Proof sketch. Since there exist no cycles, the nodes can be ordered into different levels on the graph. Every level contains all nodes from the lower level and nodes whose predecessors are contained in a lower level. Then it can be shown via induction that all nodes can be marked.

When we choose an assumption, the truth value of the literal is fixed to false. Since no further explanations than the one that the literal is an assumption are allowed, all incoming edges of the assumption nodes are removed. We treat cycles here as sets of nodes. Minimal cycles (where minimality is understood w.r.t. set inclusion) will play a crucial role. Choosing one assumption node in each minimal cycle breaks up the minimal cycles. Since all bigger cycles contain a minimal cycle they are also broken up, so there exist no more unmarked cycles. Since it is better to choose as few literals as possible as assumption, we only choose those possible assumptions that are minimal with regard to set inclusion. Then a possible assumption consists of one tentative assumption from each minimal cycle.

Approach 1. *Let $C_1, C_2, ..., C_n$ be all minimal unmarked cycles in G'.*

$$\begin{aligned} Assumptions(G') = &\{\{a_1, a_2, ..., a_n\} \mid a_1 \in C_1, a_2 \in C_2, ..., a_n \in C_n, \\ &\{a_1, a_2, ...a_n\} \textit{ is minimal w.r.t. set inclusion}\}. \end{aligned}$$

Of course there still may exist cycles which are marked. But since we know that they can be marked, we can simply replace a marked cycle $C_m = (V_m, E_m)$ by a dummy node c_m with incoming edges $ie(c_m) = \bigcup_{v \in V_m} ie(v)$ and outgoing edges $oe(c_m) = \bigcup_{v \in V_m} oe(v)$. One very simple possibility to determine minimal assumptions is to try all combinations of tentative assumptions. Such a combination is an assumption, if the whole graph can be marked after choosing the assumption. A minimal assumption is then the combination that is successful and minimal with regard to set inclusion. But with an increasing number of tentative assumptions this approach will not be very efficient. For this reason, we will introduce an approach that tries to reduce the number of combinations that have to be checked. Its basis is not to check all tentative assumptions and combinations, but only those literals that are important to determine the value of a so called *critical node*. It is obvious that this approach is only more efficient, if the number of such literals is smaller than the number of tentative assumptions. For the sake of simplicity, the approach is limited to graphs where each literal in a cycle is represented only by one node, i.e., there exist no OR-handles. When OR-handles have to be considered a similar approach can be used, just a more complex case differentiation has to be carried out.

Since marked nodes are irrelevant for the determination of assumptions, we remove all marked nodes and their outgoing and incoming edges from the graph and obtain a sub-graph $G_{unmarked}$. In the next step, we are looking for strongly connected components of $G_{unmarked}$. A strongly connected component is a maximal sub-graph where each node is reachable from each other node. This means that every node has an influence on every other node in the same strongly connected component, so that a strongly connected component behaves like a big cycle. For this reason we call the strongly connected components *linked cycles*.

If a linked cycle consists of several smaller cycles, there exist nodes belonging to multiple cycles. Such a node is *critical*, if its value depends on more than one cycle. Since we have removed all irrelevant edges and have no OR-handles, a red node has only active AND-handles. This means that the truth value of just one predecessor node is sufficient to determine the truth value of the red node. So a red node does not depend on more than one cycle, which means that only green nodes can be critical.

Definition 10 (Critical Node). *Let $LC = (V_{LC}, E_{LC})$ be a linked cycle. A node a_i^k is* critical, *if $\nu(a_i^k) = green$ and it has at least two incoming edges $(a_j^l, a_i^k, s) \in E_{LC}$ where $s \in \{+, -\}$. The set of all critical nodes of a linked cycle LC will be denoted as $\mathcal{CN}(LC)$.*

If a linked cycle has no critical nodes, a node can be deduced from any other node. Then a minimal assumptions consists of a single literal which is a tentative assumption and is represented by a node of the linked cycle. The set of all minimal assumptions of the linked cycle $LC(V_{LC}, E_{LC})$ in G' is: $\mu Assumptions(LC) = \{\{a_i\} \mid a_i^k \in V_{LC}, a_i \in \mathcal{TA}(G')\}$.

The value of every critical node depends on the value of its predecessor nodes. We will call these nodes pre-conditions.

Definition 11 (Pre-condition). *Let $LC = (V_{LC}, E_{LC})$ be a linked cycle. The* pre-conditions *for a green critical node a_i^k are:*

$$pre(a_i^k) = \{a_j^l \mid (a_j^l, a_i^k, s) \in E_{LC}, s \in \{+, -\}\}$$

$Preconditions(LC) = \bigcup_{a_i^k \in \mathcal{CN}(LC)} pre(a_i^k)$ is the set of all pre-conditions in the linked cycle.

Lemma 3. *Nodes that are not critical can be deduced from at least one critical node.*

Proof sketch. It can be shown by induction that the truth value of a node a_n on a path $c, a_1, a_2, ..., a_{n-1}, a_n$ from a critical node c can be deduced, if $a_1, ...a_n$ are not critical.

So if we can deduce all critical nodes with an assumption, we also can deduce all other literals of the linked cycle with the assumption. $lfp(Succ(\{cn\}))$ calculates the nodes that can be deduced from a critical node $cn \in \mathcal{CN}(LC)$ where $Succ(S) = \{a_j^l \mid (a_i^k, a_j^l, s) \in E_{LC}, s \in \{+, -\}, a_i^k \in S\}$. Then the set of all nodes that can be deduced from a set of nodes S can be calculated with $lfp(T(S))$ where

$$T(S) = \{a_i^k \mid pre(a_i^k) \subseteq S\} \cup \{a_i^k \mid a_i^k \in Succ(a_j^l), a_j^l \in S \cap \mathcal{CN}(LC)\}.$$

Now combinations c of pre-conditions have to be tested for success. A combination c is successful, if all critical nodes can be deduced from them, i.e., $\mathcal{CN}(LC) \subseteq lfp(T(c))$. What we know is that

1. Each combination c has to contain at least one complete pre-condition set $pre(cn)$, $cn \in \mathcal{CN}(LC)$. Otherwise the fix-point operator could not deduce any critical node.
2. Since we are looking for minimal assumption sets, we do not have to check combinations c_1 where we already have found a smaller successful combination c_2, i.e., $c_2 \subseteq c_1$ and $\mathcal{CN}(LC) \subseteq lfp(T(c_2))$.

The way of proceeding is to first test single pre-condition sets $pre(cn) \subset c$, $cn \in \mathcal{CN}(LC)$ for success (exploits fact 1). If a set is successful, we add it to the set of successful combinations C, otherwise it is put to NC. Then we test sets $n \cup \{a_i^k\}$, where $n \in NC$ and $a_i^k \in Preconditions(LC)$. This means we test different combinations of adding one more pre-condition to all sets that have not been successful in the step before (exploits fact 2). Again we add successful combinations to C and set NC to the combinations that were not successful. This is repeated till $NC = \emptyset$ or the set to be tested consists of all pre-conditions. Then C contains all combinations of pre-conditions that suffice to deduce all critical nodes and therefore to deduce also all other nodes in the linked cycle, since they are not critical. For the purpose of determining assumptions, we determine all nodes from which a pre-condition p can be deduced. These nodes lie on paths from critical nodes to the pre-condition.

Definition 12 (Pre-condition paths). *The* pre-condition path *for a pre-condition p from a linked cycle $LC = (V_{LC}, E_{LC})$ can be obtained by $path(p) = lfp(T_{path}(\{p\}))$ where*

$$T_{path}(S) = \{a_i^k \mid (a_i^k, a_j^l, s) \in E_{LC}, s \in \{+, -\}, a_i^k \notin \mathcal{CN}(LC), a_j^l \in S\}.$$

A pre-condition path contains the nodes from which a pre-condition can be deduced. For deducing all nodes of a linked cycle we have to deduce all pre-conditions of a successful combination $c = \{p_1, ..., p_n\}$. This means that we need exactly one node from the path of each pre-condition. Since we want to determine assumptions, the nodes have also to fulfill the other conditions of an assumption.

Definition 13 (Path assumptions). *The set of path assumptions for a path p in a validly colored EDG $G = (V, E)$ is defined by*

$$\mathcal{PA}(p) = \{a_i \mid a_i^k \in p \wedge a_i^k \in \mathcal{TA}(G')\}.$$

Let C be the set of all successful pre-condition combinations $c = \{p_1, ..., p_n\}$. Then the set

$$Assumptions(LC) = \bigcup_{c \in C} \{\{a_1, ..., a_n\} \mid a_1 \in \mathcal{PA}(p_1), ..., a_n \in \mathcal{PA}(p_n)\}$$

is the set of possible minimal assumptions.

Proposition 2. *Minimal assumptions $\mu Assumptions(LC)$ of a linked cycle LC are those sets of $Assumptions(LC)$ that are minimal with regard to set inclusion.*

Proof sketch. It has to be shown that each set $S \in \mu Assumptions(LC)$ is a minimal assumption for the linked cycle LC, i.e., S is an assumption and there exists no set $S' \subset S$, such that S' is an assumption for LC. To show that S is an assumption, it has to be checked, if S meets the conditions of an assumption. To show that S is minimal, one looks at the successful combinations from which S and S' are created and distinguish between different cases. For every case it can be shown by contradiction that S' can not be in $\mu Assumptions(LC)$.

Algorithm 1. Determination of assumptions using Approach 2

Require: Marked graph $G' = (V', E')$, set of linked cycles $independentLCs$
Ensure: All minimal assumptions of G

```
1: procedure findAssumptions(G,independentLCs)
2:     var graph G'' = (V'', E'') = G
3:     varset of linked cycles iLCs
4:     for all v ∈ V do                                  ▷ Remove marked nodes
5:         if v.marked = true then G''.removeNode(v) end if
6:     end for
7:     if V'' = ∅ then                  ▷ There are no more unmarked nodes
8:         var sets of assumptions μAssumptions[independentLCs]
9:         for all LC ∈ independentLCs do
10:            μAssumptions[LC] = calculateMinimalAssumptionsLC(LC)
11:        end for
12:        return calculateMinimalAssumptionsGraph(μAssumptions)
13:    else
14:        iLCs = calculateIndependentLinkedCycles(G'')
15:        independentLCs.add(iLCs)
16:                ▷ Replace independent linked cycles by a dummy node
17:        for all LC = (V_LC, E_LC) ∈ iLCs do
18:            for v ∈ V_LC do G'.removeNode(v) end for
19:            G''.addNode(v_dummy,LC)
20:            for all e = (v_s, v_e, s) ∈ E_LC do
21:                G''.addEdge(v_dummy,LC,v_e,s)
22:                G''.removeEdge(e)
23:            end for
24:        end for
25:        markNodes(G'')
26:        return findAssumptions(G'',independentLCs)
27:    end if
28: end procedure
```

Approach 2. *To calculate the minimal assumptions of a graph G Algorithm 1 is used by calling* findAssumptions$(G', \emptyset)$*, where G' is the graph obtained from G after removing irrelevant edges and nodes and after marking the nodes like described in Sect. 4.* findAssumptions *is a recursive function, which adds further independent linked cycles to the set of all independent linked cycles of G at*

each recursion. A linked cycle is independent if the truth value of its nodes does not depend on the truth value of the nodes in the rest of the graph. Since they are independent of the rest of the graph, they have to contain an assumption. To determine independent linked cycles, first all marked nodes and their incident edges are removed, so we get a graph $G'' = (V'', E'')$ *(lines 4–6). If there still exist unmarked nodes in* G''*,* G'' *contains independent linked cycles. Independent linked cycles are those linked cycles, which have no incoming edges in* G''*. A linked cycle* $LC = (V_{LC}, E_{LC})$ *has no incoming edges if for all* $a_i^k \in V_{LC}$ *there is no* $a_j^l \in V'' \setminus V_{LC}$ *and* $s \in \{+, -\}$ *such that* $(a_j^l, a_i^k, s) \in E''$*. So all linked cycles without incoming edges are determined (line 15) and added to the set of all independent linked cycles (line 15). Then each independent linked cycle is replaced by a dummy node (lines 17–24).*

In the next step, the graph is marked using the marking rules from Sect. 4 (line 25). Since we have replaced linked cycles without incoming edges by dummy nodes, these nodes have no incoming edges and are marked. With the marking process we can determine which linked cycles are dependent of the independent linked cycles, that we already have determined. The procedure is repeated until all independent linked cycles of the graph are found. This is the case, if there exist no unmarked nodes in the graph (line 7).

Then for each linked cycle the minimal assumptions are calculated. The function calculateMinimalAssumptionsLC *(line 13) contains following steps: the critical nodes of the linked cycle LC and their pre-conditions are specified and the pre-condition paths are calculated, then successful combinations of pre-conditions are looked for. The path assumptions are determined and used to calculate the minimal assumptions of the linked cycle.*

After determining the minimal assumptions of all independent linked cycles, the minimal assumptions of the graph are calculated with the function calculateMinimalAssumptionsGraph *(line 12) which exploits Proposition 3.*

Proposition 3. *We can obtain the minimal assumptions of the EDG by taking one minimal assumption of each independent linked cycle:*

$$\mu Assumptions(G') = \{\{a_1 \cup \cdots \cup a_n\} \mid a_1 \in \mu Assumptions(LC_1), \ldots \\ a_n \in \mu Assumptions(LC_n)\}$$

Proof sketch. For every assumption $a \in \mu Assumptions(G')$ two things have to be shown: (a) a is an assumption for the EDG and (b) a is minimal. (a) can be directly shown by the stopping condition of the algorithm. For the proof of (b) a literal is removed from a and it can be shown that then not all nodes in the EDG can be marked, because of the definition of the minimal assumption in an EDG and the definition of independent linked cycles.

6 Conclusion

We presented an approach to construct explanation graphs from validly colored Extended Dependency Graphs. We exploited that the logic program is already

present in graph form. In EDGs the nodes may differ in number of incoming edges and coloring, so different types of transformations were defined to transfer a node from the EDG to an EG. For the determination of assumptions it was necessary to determine the well-founded model. A strong relationship between well-founded models and the marking process of an EDG was observed, which means that an unmarked node represents a literal which is undefined in the well-founded model. We presented two different approaches for the determination of assumptions. While the first approach determines non-minimal assumptions in $O(|V| + |E|)$, the determination of minimal assumptions has an exponential complexity.

Acknowledgement. This work has been supported by the German Research Foundation DFG, Collaborative Research Center SFB876, Project A5. (http://sfb876.tu-dortmund.de)

References

1. Pontelli, E., Son, T.C., Elkhatib, O.: Justifications for logic programs under answer set semantics. Theor. Pract. Logic Program. **9**(1), 1–56 (2009)
2. Constantini, S., Provetti, A.: Graph representations of consistency and truth-dependencies in logic programs with answer set semantics. In: The 2nd International IJCAI Workshop on Graph Structures for Knowledge Representation and Reasoning (GKR 2011), vol. 1 (2011)
3. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**(3–4), 365–385 (1991)
4. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, Heidelberg (1987)
5. Brignoli, G., Costantini, S., Provetti, A.: Characterizing and computing stable models of logic programs: the non-stratified case. In: Proceedings of the Conference on Information Technology, Bhubaneswar, India. AAAI Press (1999)
6. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM **38**(3), 620–650 (1991)
7. Dimopoulos, Y., Torres, A.: Graph theoretical structures in logic programs and default theories. Theor. Comput. Sci. **170**, 209–244 (1996)