On the Operationalization of Graph Queries with Generalized Discrimination Networks

Thomas Beyhl, Dominique Blouin, Holger Giese(✉), and Leen Lambers(✉)

Hasso-Plattner Institute at the University of Potsdam,
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
{thomas.beyhl,dominique.blouin,
holger.giese,leen.lambers}@hpi.uni-potsdam.de

Abstract. Graph queries have lately gained increased interest due to application areas such as social networks, biological networks, or model queries. For the relational database case the relational algebra and generalized discrimination networks have been studied to find appropriate decompositions into subqueries and ordering of these subqueries for query evaluation or incremental updates of queries. For graph database queries however there is no formal underpinning yet that allows us to find such suitable operationalizations. Consequently, we suggest a simple operational concept for the decomposition of arbitrary complex queries into simpler subqueries and the ordering of these subqueries in form of *generalized discrimination networks* for graph queries inspired by the relational case. The approach employs graph transformation rules for the nodes of the network and thus we can employ the underlying theory. We further show that the proposed generalized discrimination networks have the same expressive power as nested graph conditions.

1 Introduction

The model of typed graphs and related graph queries to explore existing graphs and their properties has lately gained increased importance due to application areas of increasing relevance such as social networks, biological networks, and model queries [14] and technologies like graph databases [2] or model-driven development [4] where graphs rather than relations are the main characteristics of the employed models and queries.

While the definition of typed graphs by means of schemas, metamodels, or grammars is a formally well studied topic, there is yet no clear formal underpinning for graph queries concerning their specification as well as their operationalization (cf. [2,16]). For the *operationalization* of the query evaluation and incremental query updates of relational queries the *relational calculus* [1] and *generalized discrimination networks* (GDN) have been suggested (cf. [13]) as a formal framework to study which decomposition into subqueries and ordering of

This work was partially developed in the course of the project Correct Model Transformations II (GI 765/1-2), which is funded by the Deutsche Forschungsgemeinschaft.

R. Echahed and M. Minas (Eds.): ICGT 2016, LNCS 9761, pp. 170–186, 2016.
DOI: 10.1007/978-3-319-40530-8_11

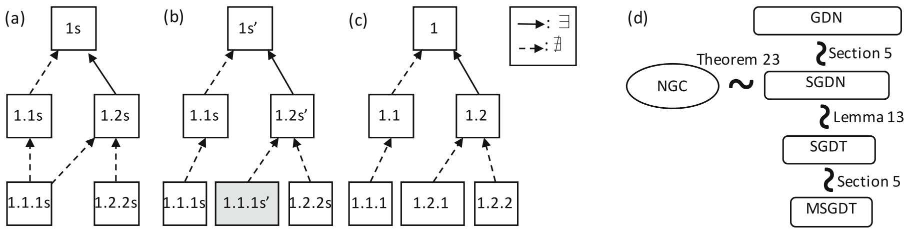

Fig. 1. GDNs in form of a SGDN (a) and SGDTs (b)(c) for a social network query

these subqueries is most appropriate. As depicted in Fig. 1(a), in such a network each node (numbered block) is responsible for evaluating a subquery and for this purpose it may compose subquery evaluations of nodes it depends on. The overall result is then the query evaluation of the terminal node. However, such a formal framework does not exist for graph queries so far.

Consequently, inspired by the relational case we suggest motivated by our practical work on view maintenance for graph databases [6] a simple operational concept for the decomposition of arbitrary complex *graph queries* into a suitable ordering of simpler subqueries in form of GDNs. Rather than considering one particular kind of GDN with particular computation nodes, we suggest employing *graph transformation* (GT) rules for these computation nodes such that we are also able to employ the well understood GT theory [9] as a basis. The basic idea to define our notion of GDN related to GT systems is to employ extra marking nodes and edges to encode the results of subqueries and specific graph transformation rules to describe the propagation behavior of the network nodes via creating and reading markings.

We study in this paper what are the core ingredients required to approach graph query evaluation based on an operational specification using the above-described GDNs while having the same expressiveness as *declarative graph queries* based on *nested graph conditions* (NGC) [12]. The latter have the expressive power of first order logic on graphs and constitute as such a natural formal foundation for pattern-based graph queries.

We assume in the following that a *graph query* is characterized by a *request graph* L delivering its answers in form of a set of matches for L into the queried graph G fulfilling some additional properties as described in the graph query.[1,2] Based on the answer set semantics we were able to establish equivalence of NGCs with GDNs including different specific subsets such as so-called simple GDNs (SGDNs), simple tree-like GDNs (SGDT), and minimal SGDTs (MSGDT). In

[1] It is to be noted that a simple record as provided by an SQL-statement is also a special form of graph where no links are included.

[2] While in practice the requested number of answers is often limited to a fixed upper bound of answers, for our more theoretical considerations in this paper, we can assume w.l.o.g. that all matches of L for G that fulfill the additional properties that must hold are building the correct set of answers.

particular as depicted in Fig. 1(d), as a main result we established the equivalence between NGCs and SGDNs and in addition showed that all GDN variants are equally expressive.

The paper is structured as follows: We first introduce our running example as well as the foundations concerning typed graphs, graph queries in their generic form, NGCs, and GT in Sect. 2. Then, in Sect. 3 operational graph queries in form of GDNs are defined and it is shown how to transform SGDNs into trees (SGDTs). That SGDNs and declarative queries based on NGCs have the same expressive power follows in Sect. 4 and we discuss the different variants of GDNs concerning their expressiveness and applicability w.r.t. optimization and incremental updates for graph queries in Sect. 5. Finally, we conclude the paper and provide an outlook on planned future work.

2 Prerequisites

After outlining our running example, we will introduce typed graphs, based on that a generic notion of graph query (language) together with the concept of equivalence, the notion of graph conditions with arbitrary nesting level (NGCs), and GT systems. Moreover, we introduce in particular the answer set of graph queries based on NGCs.

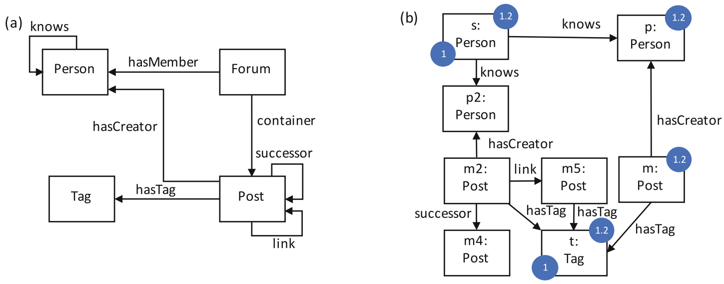

Fig. 2. Excerpt of social network type graph and an example graph G

Example 1 (social network query). As running example we use a social network model and a slightly adjusted graph query employed by the LDBC benchmark [8]. A class diagram outlining the possible graph models as well as an example graph to apply the query are depicted in Fig. 2(a) resp. (b). The considered complex graph query looks for pairs of Tags and Persons (1) such that the Tag is new in the Posts by a friend of this Person. To be a Post of a friend, the Post must be from a second Person the Person knows (1.2). In order to be new, the Tag must be linked in the latest Post of the second Person (and thus in a Post that has no successor Post) (1.2.1) and there has to be no former Post by any other or the same friend that is not her last one and where the same Tag has been already used (1.1). In both cases only Tags that are not simply inherited from a linked Post should be considered (1.1.1 and 1.2.2). Note that the employed numbering of the conditions relates to the tree-like network depicted in Fig. 1(c). Occurrences for the positive sentences (1) and (1.2) in the example graph are depicted accordingly as markers in form of blue circles with the respective number in Fig. 2(b). The circular blue markers (1) on the graph denote the occurrence of the request graph consisting of the person s and tag t. Marker (1.2) denotes the extra condition that the searched tag t must be attached (hasTag) to a post created by person p that is known by person s. Note that the markers (1) denote the only correct answer for the query. Thereby the required match for the positive subquery (1.2) depicted by the markers (1.2) is such that indeed no match exists for the negative subsubqueries (1.2.1) and (1.2.2). Furthermore, as required no match for the negative subquery (1.1) consistent with (1) exists such that no match for the negative subsubquery (1.1.1) of (1.1) can be found. Consequently, no match for (1.1) is visualized.

We briefly reintroduce the notion of typed graphs and graph morphisms [9]. A *graph* $G = (G^V, G^E, s^G, t^G)$ consists of a set G^V of nodes, a set G^E of edges, a source function $s^G : G^E \to G^V$, and a target function $t^G : G^E \to G^V$. Given the graphs $G = (G^V, G^E, s^G, t^G)$ and $H = (H^V, H^E, s^H, t^H)$, a *graph morphism* $f : G \to H$ is a pair of mappings, $f^V : G^V \to H^V, f^E : G^E \to H^E$ such that $f^V \circ s^G = s^H \circ f^E$ and $f^V \circ t^G = t^H \circ f^E$. A graph morphism $f : G \to H$ is a *monomorphism* if f^V and f^E are injective mappings. Finally, two graph morphisms $m : H \to G$ and $m' : H' \to G$ are *jointly epimorphic* if $m^V(H^V) \cup m'^V(H'^V) = G^V$ and $m^E(H^E) \cup m'^E(H'^E) = G^E$. A *type graph* is a distinguished graph $TG = (TG^V, TG^E, s^{TG}, t^{TG})$. TG^V and TG^E are called the vertex and the edge type alphabets, respectively. A tuple $(G, type)$ of a graph G together with a graph morphism $type : G \to TG$ is then called a *typed graph*. Given typed graphs $G_1^T = (G_1, type_1)$ and $G_2^T = (G_2, type_2)$, a *typed graph morphism* $f : G_1^T \to G_2^T$ is a graph morphism $f : G_1 \to G_2$ such that $type_2 \circ f = type_1$. We further denote the set of all graphs typed over some type graph TG by $\mathcal{L}(TG)$.

An example for a *typed graph* G and the type graph TG related to the social network query Example 1 are depicted in Fig. 2.

In the rest of the paper we will compare the answer sets of graph queries to analyze them for equivalence. Since we will compare queries stemming from different query languages, we introduce here a generic notion of query (language) equivalence that we will refine in the rest of the paper to particular queries and query languages. As the most generic form of a graph query language we just assume that it consists of a set of graph queries, where each graph query is characterized by a request graph L typed over some type graph TG. The query then expresses some extra properties that need to hold for the request graph L that is searched for in the queried graph G. The answer set for this query then describes all matches of L in the queried graph that fulfill these extra properties.

Definition 2 (graph query (language)). *Given a type graph TG, then a* graph query *is characterized by a so-called* request graph L*, which is a finite graph typed over* TG*. A* graph query language *is a set of graph queries.*

Definition 3 (answer set mapping, equivalence). *Given some graph query language $\mathcal{L}$, an* answer set mapping *ans* for $\mathcal{L}$ *maps each pair* (q_L, G) *with* q_L *a graph query in* $\mathcal{L}$ *with request graph* L *typed over* TG *and* G *a graph from* $\mathcal{L}(TG)$ *to a set of graph morphisms typed over* TG *with domain* L *and co-domain* G.

Given queries q_L *and* q'_L *for some request graph* L *typed over* TG *belonging to the graph query languages* $\mathcal{L}$ *and* $\mathcal{L}'$ *with answer set mappings ans and ans', resp., then* q_L *and* q'_L *are* equivalent *if for every graph* G *in* $\mathcal{L}(TG)$ *it holds that* $ans(q_L, G) = ans'(q'_L, G)$*. Two graph query languages* $\mathcal{L}$ *and* $\mathcal{L}'$ *are* equivalent *if for any query* $q_L \in \mathcal{L}$ *for some request graph* L *there exists some query* $q'_L \in \mathcal{L}'$ *for* L *such that* $q_L \sim q'_L$ *and vice versa. We denote equivalence also with* $\sim$.

We reintroduce the notion of *nested graph conditions* (NGC) from [12], since they represent the declarative kind of graph queries that we will consider in this paper. Given a finite graph L, a *nested graph condition* (NGC) over L is defined inductively as follows: (1) *true* is a NGC over L. We say that *true* has nesting level 0. (2) For every morphism $a : L \to L'$ and NGC $c_{L'}$ over a finite graph L' with nesting level n such that $n \geq 0$, $\exists(a, c_{L'})$ is a NGC over L with nesting level $n + 1$. (3) Given NGCs over L, c_L and c'_L, with nesting level n and n', respectively, $\neg c_L$ and $c_L \wedge c'_L$ are NGCs over L with nesting level n and $max(n, n')$, respectively. We restrict ourselves to finite NGCs, i.e. each conjunction of NGCs is finite. We define when a morphism $q : L \to G$ *satisfies* a NGC c_L over L inductively: (1) Every morphism q satisfies *true*. (2) A morphism q satisfies $\exists(a, c_{L'})$, denoted $q \models \exists(a, c_{L'})$, if there exists a monomorphism $q' : L' \to G$ such that $q' \circ a = q$ and $q' \models c_{L'}$. (3) A morphism q satisfies $\neg c_L$ if it does not satisfy c_L and satisfies $\wedge_{i \in I} c_{L,i}$ if it satisfies each $c_{L,i}$ $(i \in I)$. Note that $false$, $\vee$, and $\Rightarrow$ can be mapped as usual to the introduced logical connectives. Moreover we abbreviate $\exists(\emptyset \to L', c_{L'})$ with $\exists(L', c_{L'})$, $\exists(a, true)$ with $\exists a$ and $\forall(a, c_{L'})$ with $\neg\exists(a, \neg c_{L'})$. NGCs can be equipped with *typing* over a given type graph TG as usual [9] by adding typing morphisms from each graph to TG and by requiring type-compatibility with respect to TG for each graph morphism.[3]

Definition 4 ($\mathcal{L}_{NGC}$, ans_{NGC}). *The graph query language* $\mathcal{L}_{NGC}$ *is the set of all NGCs. Given some NGC* c_L *over* L*,* L *represents the so-called request graph. The* answer set mapping ans_{NGC} *for* $\mathcal{L}_{NGC}$ *is given by*

$$ans_{NGC}(c_L, G) = \{q : L \to G | q \text{ is a monomorphism and } q \models c_L\}$$

with $c_L \in \mathcal{L}_{NGC}$ *a NGC with* L *typed over some type graph* TG *and* G *in* $\mathcal{L}(TG)$.

[3] W.l.o.g. we restrict our notion of condition satisfaction to the existence of monomorphisms. In particular, in [12] it is shown how to translate conditions relying on general morphism matching/satisfaction into equivalent conditions relying on monomorphism matching/satisfaction and the other way round.

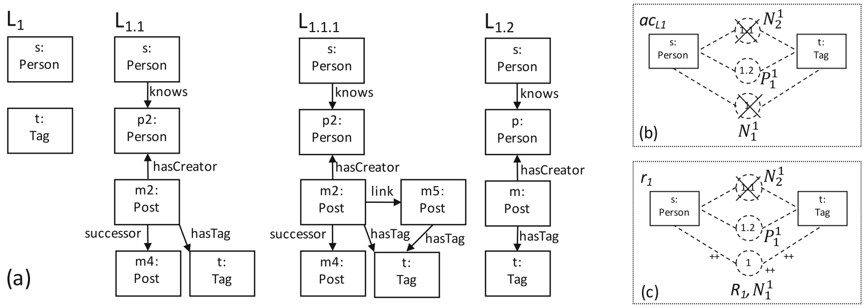

Fig. 3. Graphs for the NGC c_1 and its subconditions (a) and the application condition $ac_{L_1} = \exists(L_1 \to P_1^1) \wedge \nexists(L_1 \to N_1^1) \wedge \nexists(L_1 \to N_2^1)$ (b) and simple marking rule $r_1 = (L_1 \to R_1, ac_{L_1})$ (c)

An example NGC for the social network query of Example 1, where the subconditions refer to the introduced numbering, is the following: $c_1 = c_{1.1} \wedge c_{1.2}$ with $c_{1.1} = \neg\exists(n_{1.1} : L_1 \to L_{1.1}, c_{1.1.1})$, $c_{1.2} = \exists(p_{1.2} : L_1 \to L_{1.2}, c_{1.2.1} \wedge c_{1.2.2})$, $c_{1.1.1} = \neg\exists(n_{1.1.1} : L_{1.1} \to L_{1.1.1}, true)$, $c_{1.2.1} = \neg\exists(n_{1.2.1} : L_{1.2} \to L_{1.2.1}, true)$, and $c_{1.2.2} = \neg\exists(n_{1.2.2} : L_{1.2} \to L_{1.2.2}, true)$. The graphs L_1, $L_{1.1}$, $L_{1.1.1}$, and $L_{1.2}$ are depicted exemplarily (see [5] for the complete example) in Fig. 3(a). Morphisms are implied by equally named objects.

As foundation for an operational graph query evaluation we will employ typed GT systems with priorities. We start with reintroducing GT and thereby assume the double-pushout approach (DPO) with injective matching and non-deleting rules [9] with application conditions of arbitrary nesting level (AC) [12]. A plain GT rule $p : L \to R$ is a graph monomorphism. We say that the graphs L and R are the left-hand side (LHS) and right-hand side (RHS) of the rule, respectively. A *GT rule* $r = \langle p, \mathrm{ac}_L \rangle$ consists of a plain rule $p : L \to R$ and a so-called application condition ac_L being a graph condition over L. If the application condition $\mathrm{ac}_L = \wedge_{i \in I} \exists p_i \wedge \wedge_{j \in J} \nexists n_j$, then we say that $\exists p_i$ or $\neg\exists n_j$ is a positive application condition (PACs) or negative application condition (NAC) over L, respectively. A rule r is *applicable* to a graph G via a graph monomorphism $m : L \to G$ if $m \models \mathrm{ac}_L$. A *direct GT* via rule $r = \langle p, \mathrm{ac}_L \rangle$ consists of a pushout over p and m such that $m \models \mathrm{ac}_L$. If there exists a direct transformation from G to G' via rule r and match m, we write $G \Rightarrow_{m,r} G'$. If we are only interested in the rule r, we write $G \Rightarrow_r G'$. If a rule r in a set of rules $\mathcal{R}$ exists such that there exists a direct transformation via rule r from G to G', we write $G \Rightarrow_{\mathcal{R}} G'$. A *GT*, denoted as $G_0 \Rightarrow^* G_n$, is a sequence $G_0 \Rightarrow G_1 \Rightarrow \cdots \Rightarrow G_n$ of $n \geq 0$ direct GT. GT rules and GTs can be equipped with *typing* over a given type graph TG as usual [9] by adding typing morphisms from each graph to TG and by requiring type-compatibility with respect to TG for each graph morphism.

An example for a GT rule with AC in the context of the social network query of Example 1 is $r_1 = (L_1 \to R_1, ac_{L_1})$ as depicted in Fig. 3(c) following the compact notation where all graphs are embedded into a single one. In particular,

$ac_{L_1} = \exists(L_1 \to P_1^1) \wedge \nexists(L_1 \to N_1^1) \wedge \nexists(L_1 \to N_2^1)$ is depicted more precisely in Fig. 3(b). ++ denotes elements that are created by the rule, the additional (dashed) elements forbidden by a NAC are crossed out and the extra elements required by a PAC are dashed as well. These crosses for NAC N_1^1 are omitted from the rule visualization in Fig. 3(c) as it equals R_1^1. Note that we use in this example in addition to the node types defined in the type graph depicted in Fig. 2(a) (solid rectangles) already some additional marking node (dashed circles) and edge types (dashed lines) that will be introduced later.

A *graph transformation system* (GTS) gts $= (\mathcal{R}, \mathrm{TG})$ consists of a set of rules $\mathcal{R}$ typed over a type graph TG. If a rule r in $\mathcal{R}$ of gts exists such that a direct transformation $G \Rightarrow_r G'$ via r exists, we also write $G \Rightarrow_{\mathrm{gts}} G'$. If for some graph G it holds that r is not applicable to G, then we write $G \not\Rightarrow_r$. Moreover, if no rule in gts exists that is applicable to G, then we write $G \not\Rightarrow_{\mathrm{gts}}$. A GTS *with priorities* $\mathrm{gts}_p = ((\mathcal{R}, TG), p)$ consists of a GTS $(\mathcal{R}, TG)$ and a transitive and asymmetric relation $p \subset \mathcal{R} \times \mathcal{R}$. We write $G \Rightarrow_{\mathrm{gts}_p} G'$ if a rule r in $\mathcal{R}$ of gts_p exists with a direct transformation $G \Rightarrow_r G'$ such that $\nexists r' \in \mathcal{R} : (r, r') \in p \wedge G \Rightarrow_{r'} G''$. For a GTS with priorities gts_p and an initial graph G_0 the *set of reachable graphs* $\mathsf{REACH}(\mathrm{gts}_p, G_0)$ is defined as $\{G \mid G_0 \Rightarrow^*_{\mathrm{gts}_p} G\}$ and the *set of terminal reachable graphs* $\mathsf{TERM}(gts_p, G_0)$ is defined as $\{G | G \in \mathsf{REACH}(\mathrm{gts}_p, G_0) \colon G \not\Rightarrow_{gts_p}\}$.

3 Generalized Discrimination Networks

In the following we introduce our suggestion for the operationalization of graph queries employing generalized discrimination networks with computation nodes based on GT rules.

Example 5 (GDN (informal)). A possible GDN for the social network query Example 1 is depicted in Fig. 1(a). Node 1.1.1s and 1.2.2s produce their output independently. Then, node 1.1s and 1.2s can compute the output depending on the output of these two other nodes. Finally, the terminal node 1s can compute its output based on the output of the nodes 1.1s and 1.2s. We further distinguish in Fig. 1(a) positive and negated dependencies accordingly visualized by arrows with a single solid line when representing a PAC ($\exists$) and by arrows with a single dashed line when representing a NAC ($\nexists$).

Our queried graph G typed over TG will be marked with so-called marking nodes and edges to keep track of (sub-)query answer sets. In particular, so-called marking rules in a GDN will take care of that. A (simple) marking rule r_i is a restricted form of GT rule typed over a marking type graph TG'. The latter is equal to TG but for each marking rule r_i it is extended with a so-called marking node type t_i as well as an marking edge type t_v per node v present in r_i's LHS L_i. This allows r_i to mark each node v from L_i by adding a marking node i uniquely corresponding to r_i via its marking node type t_i, called the defined type, and by adding a marking edge e_v from this special marking node i to each node v in L_i. These marking edges encode again via their type t_v which node v in L_i they mark. Finally the application conditions in each marking rule allow for referring to the marking elements (and therefore indirectly to already matched elements) created by other rules.

The required extension for the type graph TG for the social network query Example 5 for rule r_1, which captures that a s:Person and t:Tag exist for which additional conditions must hold, are depicted in Fig. 3(c). Additional nodes visualized as circles with number 1, 1.1, and 1.2, where 1 denotes the created marking node of the rule r_1 and 1.1 and 1.2 are marking nodes of the other rules $r_{1.1}$ and $r_{1.2}$ all use types in TG' but not TG. The edges between the circles and the rectangles also belong to TG' but not TG. We do not visualize their direction, since they always point to nodes of a type from TG.

Definition 6 (marking type graph). *Given a set of graphs $(L_i)_{i \in I}$ typed over TG via $type_i : L_i \to TG$, the* marking type graph *TG' for $(L_i)_{i \in I}$ has node set $TG'^V = TG^V \uplus \{t_i | i \in I\}$ and edge set $TG'^E = TG^E \uplus \{t_v | v \in L_i^V, i \in I\}$ s.t. $s^{TG'}(e) = s^{TG}(e)$ and $t^{TG'}(e) = t^{TG}(e)$ for $e \in TG^E$ and $s^{TG'}(t_v) = t_i$ and $t^{TG'}(t_v) = type_i^V(v)$ for each $v \in L_i^V$ and $i \in I$ otherwise. We say that the nodes in $\{t_i | i \in I\}$ are* marking node types *and edges in $\{t_v | v \in L_i^V, i \in I\}$ are* marking edge types*, respectively. Given a graph G typed over TG', then we say that a node or edge in G such that its type equals a marking node or edge type in TG' is a* marking node or edge *in G, resp..*

Definition 7 ((simple) marking rule, defined type). *Given a set of graphs $(L_i)_{i \in I}$ typed over TG via $type_i : L_i \to TG$, a* marking rule *(MR) is a GT rule $r_i = \langle p_i : L_i \to R_i, \nexists p_i \wedge c_{L_i} \rangle$ typed over the* marking type graph *TG' for $(L_i)_{i \in I}$ such that (1) L_i inherits its typing from $type_{L_i}$, (2) $R_i^V = L_i^V \uplus \{i\}$ with i of type t_i the so-called* marking node *and t_i the so-called* defined type *of rule r_i, and (3) $R_i^E = L_i^E \uplus \{e_v | v \in L_i^V\}$ such that each e_v has type t_v and $s^{R_i}(e_v) = i$ and $t^{R_i}(e_v) = v$.*

A simple marking rule *(SMR) is a marking rule where the application condition $c_{L_i} = \bigwedge_{j \in J} (\exists p_j : L_i \to P_j) \wedge \bigwedge_{k \in K} (\nexists n_k : L_i \to N_k)$ such that for each $j \in J$ and $k \in K$ it holds that $P_j^V \setminus (p_j(L_i))^V$ and $N_k^V \setminus (n_k(L_i))^V$, resp., consist of exactly one marking node.*

In addition to the defined type of its created marking node each marking rule induces so-called referred types in the marking type graph. Based on these referred and defined types of MRs we define a dependency relation between MRs.

Definition 8 (referred types, dependency relation). *Given a set of graphs $(L_i)_{i \in I}$ typed over TG and a (simple) marking rule $r_i = \langle p_i : L_i \to R_i, \nexists p_i \wedge c_{L_i} \rangle$ typed over the marking type graph TG' for $(L_i)_{i \in I}$ the set of* referred types *$rt(r_i)$ is the set of all node types in TG'^V for nodes occurring in some (co-)domain graph of a morphism employed in c_{L_i}.*

Given a GTS $(\mathcal{R} = (r_i)_{i \in I}, TG')$ with each rule r_i a (simple) marking rule, a dependency relation *$\rightsquigarrow_d \subseteq \mathcal{R} \times \mathcal{R}$ consists of all rule pairs (r_i, r_j) such that the defined type t_j of rule r_j belongs to the set of referred types $rt(r_i)$.*

Note that by definition a MR r_i can only depend on itself if its defined type t_i is employed for typing elements in the application condition c_{L_i}.

The SMRs for the SGDN for the social network query of Example 5 are depicted in Fig. 4. We use here and in the following the more compact notation for SMRs where all graphs including the PACs and NACs are embedded into a single one as presented in Fig. 3(c), moreover the RHS as well as the NAC equal to p_i are omitted since they can be reconstructed from the rule's LHS uniquely.

Based on the previously introduced MRs or SMRs to encode the behavior of the computation nodes of a GDN, we can now introduce our form of GDN or SGDN, respectively.

Definition 9 (GDN, SGDN, $\mathcal{L}_{GDN}$, $\mathcal{L}_{SGDN}$). *Given a finite graph L typed over TG and a GTS $(\mathcal{R} = (r_i)_{i \in I}, TG')$ of (simple) marking rules typed over the marking type graph TG' for $(L_i)_{i \in I}$, then $gdn_L = ((\mathcal{R}, TG'), \rightsquigarrow_d^+)$ is a* (simple) generalized discrimination network *over L if the following conditions hold: (1) the transitive closure $\rightsquigarrow_d^+$ is acyclic, (2) there is a unique so-called* terminal rule *r_t with LHS $L_t = L$ for some $t \in I$, and (3) $\forall i \in I$ s.t. $i \neq t$ it holds that (r_t, r_i) is in $\rightsquigarrow_d^+$. The graph query language $\mathcal{L}_{GDN}$ ($\mathcal{L}_{SGDN}$) is the set of all GDNs (SGDNs). Given some GDN gdn_L (SGDN $sgdn_L$) over L, L represents the so-called request graph.*

Note that it follows directly from this definition that no rule of the GDN transitively depends on the terminal rule otherwise the transitive closure of the dependency relation would contain a cycle.

An example for a SGDN is depicted in Figs. 1(a) and 4, where Fig. 1(a) shows the dependencies between the nodes and Fig. 4 shows the rules for the nodes r_{1s}, $r_{1.1s}$, $r_{1.2s}$, $r_{1.1.1s}$, and $r_{1.2.2s}$.

In the following definitions we assume an operational query in the form of a GDN. In particular, each GDN represents a GTS with priorities. We consider each graph reachable via the GDN to encode an intermediate query result and the terminal graph then encodes the final query result. As shown in the subsequent lemma this terminal graph is indeed unique.

Lemma 10 (unique terminal graph). *Given a GDN $gdn_L = ((\mathcal{R}, TG'), \rightsquigarrow_d^+)$ for L typed over TG, then $\mathsf{TERM}(gdn_L, G)$ consists of exactly one graph.*

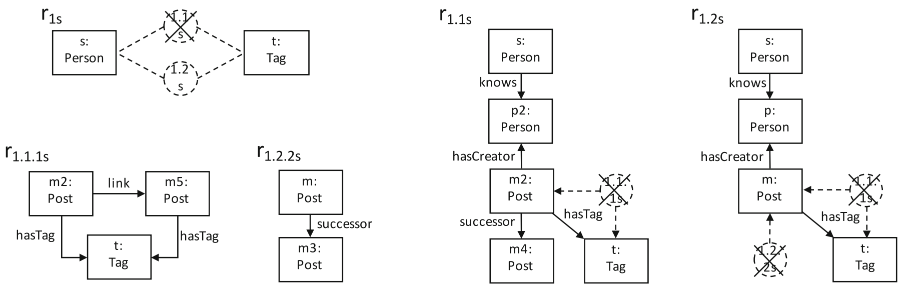

Fig. 4. SMRs for the SGDN of the social network example

Proof. (sketch; more details see [5]) As there is an upper bound on matches that can be marked and rule applications always add exactly one such marking, gdn_L *terminates. As the priorities expressed by* $\rightsquigarrow_d^+$ *exclude conflicting applications of different rules and acyclicity of* $\rightsquigarrow_d^+$ *excludes conflicting applications of a rule with itself,* gdn_L *is also confluent.*

Definition 11 (ans_{GDN}**).** *Given the graph query language* $\mathcal{L}_{GDN}$*, the* answer set mapping ans_{GDN} *for* $\mathcal{L}_{GDN}$ *is given by*

$$ans_{GDN}(gdn_L, G) := \{o: L \rightarrow G | G_i \Rightarrow_{o', r_t} G'_i \text{ is a direct GT in } t \wedge o(L) = o'(L)\}$$

with $gdn_L = ((\mathcal{R}, TG'), \rightsquigarrow_d^+)$ *some GDN such that* L *is typed over* TG*,* G *a graph in* $\mathcal{L}(TG)$*,* r_t *the terminal rule of* gdn_L *and* $t : G \Rightarrow^*_{gdn_L} G'$ *some transformation with* $\{G'\} = \mathsf{TERM}(gdn_L, G)$*.*

The above definition is well-defined, since matches are never destroyed because of dealing only with non-deleting rules and no conflicting direct transformations arise because of the priorities encoded with $\rightsquigarrow_d^+$ and acyclicity of $\rightsquigarrow_d^+$ (as mentioned also w.r.t. terminal graph uniqueness). Moreover, for $o' : L \rightarrow G_i$ it holds that $o'(L)$ is a subgraph of G.

In practice, it is important for efficiency reasons that we can reconstruct the answer set $ans_{GDN}(gdn_L, G)$ from the markings in the terminal graph G' without having to consider the transformation t leading to G'. Under the condition that we only query graphs without parallel edges of the same type this can be done uniquely (see [5]).

The following result shows that for each SGDN an equivalent tree-like SGDN exists in which no two rules exist that directly depend on the same rule and each dependency is caused by exactly one PAC/NAC. As the considerations in the following section are considerably simpler when operating on tree-like SGDNs, we will w.l.o.g (cf. Lemma 13) in the following restrict to tree-like networks.

Definition 12 (SGDT, $\mathcal{L}_{SGDT}$**).** *A* simple generalized discrimination tree (SDGT) *is a SGDN* $sgdn_L = ((\mathcal{R} = (r_i)_{i \in I}, TG'), \rightsquigarrow_d^+)$ *such that (1) for each* $(r_i, r_j) \in \rightsquigarrow_d$ *no* $k \in I$ *with* $k \neq i$ *exists s.t.* $(r_k, r_j) \in \rightsquigarrow_d$ *and (2) for each* $i \in I$ *it holds that for each PAC or NAC of* r_i *no other PAC or NAC in* r_i *exists referring to the same marking node type. The graph query language* $\mathcal{L}_{SGDT}$ *is the set of all SGDTs.*

Lemma 13 ($\mathcal{L}_{SGDN} \sim \mathcal{L}_{SGDT}$**).** *Given a SGDN* $sgdn_L$ *for a graph* L *typed over* TG*, then it holds that a SGDT* $sgdt_L$ *exists such that* $sgdn_L \sim sgdt_L$*. Moreover,* $\mathcal{L}_{SGDN} \sim \mathcal{L}_{SGDT}$*.*

Proof. (sketch, details see [5]) We can show by induction over the depth of $\rightsquigarrow_d^+$ *that we can construct an equivalent tree by employing copied rules with disjoint markings. Since each SGDT is in particular also a SGDN, it directly follows that* $\mathcal{L}_{SGDN} \sim \mathcal{L}_{SGDT}$*.*

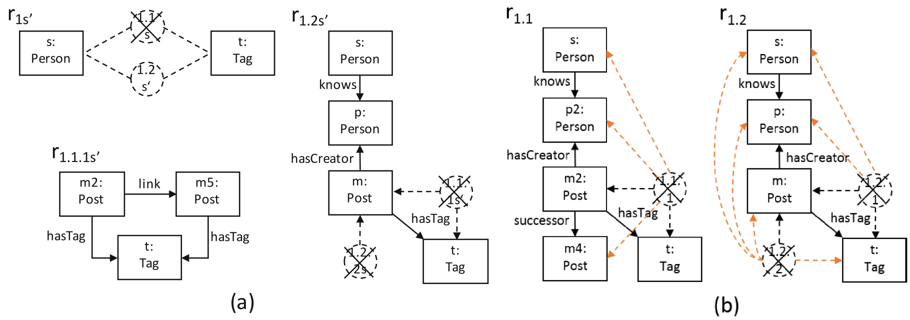

Fig. 5. SMRs for the SGDT for the social network example (a) and with maximal context (b) as denoted by the orange dashed lines.

The SMRs of the SGDT related to the SGDN of Fig. 1(a) depicted in Fig. 1(b) where multiple referenced SMRs are simply replicated are presented in Fig. 5(a). The rules $r_{1.1s}$, $r_{1.1.1s}$, and $r_{1.2.2s}$ of Fig. 4 are not shown in Fig. 5 since they remain the same. Rules $r_{1s'}$ and $r_{1.2s'}$, which differ from the rules r_{1s} and $r_{1.2s}$ of Fig. 4 only concerning the referenced other rules are shown, along with rule $r_{1.1.1s'}$, which is a replication of rule $r_{1.1.1s}$ that differs only w.r.t. created elements (omitted from the visualization).

4 Equivalence to Nested Graph Conditions

In order to prove that each NGC can be represented by some equivalent SGDT, we first show in the following Lemmas that the standard operators in NGCs (true, existential quantification, negation and binary conjunction) (Def. see Sect. 2) can be simulated by equivalent constructions in a SGDT.

Lemma 14 (*true*)**.** *Given the NGC true over L, there exists some SGDT $sgdt_L$ such that $sgdt_L \sim true$.*

Proof. Let $sgdt_L = (\{r_{L,true}\}, TG'), \rightsquigarrow_d^+)$ for L typed over TG with marking rule $r_{L,true} = \langle p : L \to R, \nexists p\rangle$, then for each graph G typed over TG, $ans_{GDN}(sgdt_L, G)$ consists of all morphisms $p : L \to G$. This means that $sgdt_L \sim true$.

Lemma 15 ($\exists(a : L \to L', c_{L'})$)**.** *Given some NGC $\exists(a : L \to L', c_{L'})$ and SGDT $sgdt'_{L'}$ such that $sgdt'_{L'} \sim c_{L'}$, there exists some SGDT $sgdt_L$ such that $sgdt_L \sim \exists(a : L \to L', c_{L'})$.*

Proof. Suppose that $sgdt'_{L'}$ has the terminal rule $r'_t = \langle p'_t : L' \to R', \nexists p'_t \wedge c'_{L'}\rangle$. We construct the SGDT $sgdt_L$ by having an additional rule $r_{L,\exists a} = \langle p : L \to R, \nexists p \wedge \exists(p'_t \circ a, true)\rangle$ w.r.t. $sgdt'_{L'}$ as terminal rule. Consider $ans_{GDN}(sgdt_L, G)$ consisting of all morphisms $o : L \to G$ s.t. $r_{L,\exists a}$ created a marking to $o(L)$. Because of the PAC $\exists(p'_t \circ a, true)$ in the terminal rule $r_{L,\exists a}$ this can only be *the case if* r'_t *created a marking for some* $o'(L')$ *with* $o' : L' \to G$ *a morphism in* $ans_{GDN}(sgdt'_{L'}, G)$. *Since* $sgdt'_{L'} \sim c_{L'}$ *we know that* r'_t *created a marking to* $o'(L')$ *iff* $o' \models c_{L'}$. *Therefore we conclude that* $o \models \exists(a : L \to L', c_{L'})$ *and thus* $sgdt_L \sim \exists(a : L \to L', c_{L'})$.

Lemma 16 **(**$\neg c_L$**).** *Given some NGC* $\neg c_L$ *and SGDT* $sgdt'_L$ *such that* $sgdt'_L \sim c_L$, *there exists some SGDT* $sgdt_L$ *such that* $sgdt_L \sim \neg c_L$.

Proof. Suppose that $sgdt'_L$ *has the terminal rule* $r = \langle p' : L \to R', \nexists p' \wedge c'_L \rangle$. *Then consider the SGDT* $sgdt_L$ *having an additional rule* $r_{L,\neg} = \langle p : L \to R, \nexists p \wedge \nexists p' \rangle$ *w.r.t.* $sgdt'_L$ *as terminal rule. Consider* $ans_{GDN}(sgdt_L, G)$ *consisting of all morphisms* $o : L \to G$ *s.t.* $r_{L,\neg}$ *created a marking to* $o(L)$. *Because of the NAC* $\nexists p'$ *in the terminal rule* $r_{L,\neg}$ *this can only be the case if* r *did not create a marking to* $o(L)$. *Since* $sgdt'_L \sim c_L$ *we know that* r *created a marking to* $o(L)$ *iff* $o \models c_L$. *Therefore we conclude that* $o \models \neg c_L$ *and thus* $sgdt_L \sim \neg c_L$.

Lemma 17 **(**$c_{1,L} \wedge c_{2,L}$**).** *Given some NGC* $c_{1,L} \wedge c_{2,L}$ *and SGDTs* $sgdt^1_L$ *and* $sgdt^2_L$ *such that* $sgdt^1_L \sim c_{1,L}$ *and* $sgdt^2_L \sim c_{2,L}$, *there exists some SGDT* $sgdt_L$ *such that* $sgdt_L \sim c_{1,L} \wedge c_{2,L}$.

Proof. Let $r_1 = \langle p_1 : L \to R_1, \nexists p_1 \wedge c_L \rangle$ *and* $r_2 = \langle p_2 : L \to R_2, \nexists p_2 \wedge c'_L \rangle$ *be the terminal rules for* $sgdt^1_L$ *and* $sgdt^2_L$, *respectively. Consider the SGDT* $sgdt_L$ *consisting of the subtrees* $sgdt^1_L$ *and* $sgdt^2_L$ *with the additional rule* $r_{L,\wedge} = \langle p : L \to R, \nexists p \wedge \exists p_1 \wedge \exists p_2 \rangle$ *as terminal rule. Consider* $ans_{GDN}(sgdt_L, G)$ *consisting of all morphisms* $o : L \to G$ *s.t.* $r_{L,\wedge}$ *created a marking to* $o(L)$. *Because of the PACs* $\exists p_1$ *and* $\exists p_2$ *in the terminal rule* $r_{L,\wedge}$ *this can only be the case if* r_1 *as well as* r_2 *created a marking to* $o(L)$. *Since* $sgdt^1_L \sim c_{1,L}$ *resp.* $sgdt^2_L \sim c_{2,L}$ *we know that* r_1 *resp.* r_2 *created a marking to* $o(L)$ *iff* $o \models c_{1,L}$ *resp.* $o \models c_{2,L}$. *Therefore we conclude that* $o \models c_{1,L} \wedge c_{2,L}$ *and thus* $sgdt_L \sim c_{1,L} \wedge c_{2,L}$.

Now we can prove that each NGC can be emulated by an equivalent SGDT.

Proposition 18. *Given a NGC* c_L, *there exists a SGDT* $sgdt_L$ *s.t.* $sgdt_L \sim c_L$.

Proof. We prove this by induction over the nesting level of NGCs and the way they are constructed.
Base case*: By Lemma 14 it follows that for* $c_L = true$ *with nesting level 0 an equivalent SGDT with a single marking rule exists. From Lemmas 16 and 17 it follows that for any combination of conditions of nesting level 0 we can still construct an equivalent SGDT.*
Induction step*: By Lemmas 15 and the induction hypothesis it follows that for any condition* $\exists(a : L \to L', c_{L'})$ *of nesting level* $n+1$ *it follows that an equivalent SGDT exists. From Lemmas 16 and 17 it follows that for any combination of conditions of nesting level n+1 we can still construct an equivalent SGDT.*

We still need to show that also each SGDT can be emulated by an equivalent NGC. An important first step thereby is the construction of a transformation of some SGDT into a SGDT with so-called maximal context. Marking rules in

GDNs are able to pass merely the context necessary for the next subquery, which is a practical property for efficiency reasons, but not for showing equivalence with NGCs based on maximal context passing. With context propagation we therefore introduce a mechanism transforming marking rules passing only partial context into rules passing maximal context. We moreover show that this context propagation does not alter the answer set semantics of the corresponding SGDT.

Definition 19 (maximal context). *Given a SGDT $sgdt_L$ for a graph L typed over TG then $sgdt_L$ has* maximal context *if for each two SMRs $r_i = \langle p_i : L_i \to R_i, \nexists p_i \wedge \bigwedge_{j \in J_i}(\exists p^i_j : L_i \to P^i_j) \wedge \bigwedge_{k \in K_i}(\nexists n^i_k : L_i \to N^i_k)\rangle$ and $r_l = \langle p_l : L_l \to R_l, \nexists p_l \wedge \bigwedge_{j \in J_l}(\exists p^l_j : L_l \to P^l_j) \wedge \bigwedge_{k \in K_l}(\nexists n^l_k : L_l \to N^l_k)\rangle$ with marking node l s.t. $(r_i, r_l) \in \rightsquigarrow_d$ because for some $j \in J_i$ (or $k \in K_i$) p^i_j (or n^i_k, resp.) uses a type equal to the type t_l of l, the sets V^i_j (or V^i_k, resp.) constructed as follows are empty:*

$$V^i_j = \{n | n \in L^V_i \text{ s.t. } \nexists e \in (P^i_j)^E \text{ with type of } s^{P^i_j}(e) = t_l \wedge t^{P^i_j}(e) = p^i_j(n)\}$$

$$V^i_k = \{n | n \in L^V_i \text{ s.t. } \nexists e \in (N^i_k)^E \text{ with type of } s^{N^i_k}(e) = t_l \wedge t^{N^i_k}(e) = n^i_k(n)\}$$

Lemma 20 (context propagation). *Given a SGDT $sgdt_L$ for a graph L typed over TG with two rules r_i and r_l such that $(r_i, r_l) \in \rightsquigarrow_d$ with non-empty V^i_j (or V^i_k) (as given in Definition 19), then there exists some $sgdt^c_L$ in which (r_i, r_l) has been replaced by a SGDT with maximal context such that $sgdt^c_L \sim sgdt_L$.*

Proof. (sketch; details see Lemma 20) We construct a $sgdt^c_L$ in which marking rules with propagated context check in contrast to r_l the presence of additional nodes and edges in the queried graph G that would otherwise have been searched for anyway by rule r_i after all matches for r_l had been found. Marking these elements earlier does not change the overall answer set.

Lemma 21 (maximal context). *For a SGDT $sgdt_L$ for a graph L typed over TG their exists a SGDT $sgdt'_L$ with maximal context such that $sgdt'_L \sim sgdt_L$.*

Proof. We proof this lemma by induction on the height of the tree.
Base case: *Suppose that we have $sgdt_L$ with height 0, then it trivially holds that $sgdt_L$ has maximal context already.*
Induction step: *Suppose that we have $sgdt_L$ with height $n+1$. Then apply subsequently for each $(r_t, r_i) \in \rightsquigarrow_d$ context propagation to $sgdt_L$ obtaining according to Lemma 20 an equivalent $sgdt^c_L$ of height $n+1$. Now consider for each r_i the subtree $sgdt^{r_i}_{L^c_i}$ in $sgdt^c_L$ of height n. Then for each $sgdt^{r_i}_{L^c_i}$ by induction hypothesis an equivalent SGDT $sgdt'_{L^c_i}$ with maximal context exists. Replacing in $sgdt^c_L$ each $sgdt^{r_i}_{L^c_i}$ with $sgdt'_{L^c_i}$ we obtain a SGDT $sgdt'_L$ with maximal context s.t. $sgdt'_L \sim sgdt_L$.*

Two of the modified SMRs of the SGDT depicted in Fig. 1(c) with maximal context related to the SGDN of Fig. 1(a) are presented in Fig. 5(b). While the rules $r_{1.1}$ and $r_{1.2}$ already have maximal context and therefore differ from the

$r_{1.1s}$ and $r_{1.2s'}$ only concerning the referenced other rules and additional links to bind the propagated context as depicted in Fig. 5(b) by the orange edges, the rules $r_{1.1.1}$, $r_{1.2.1}$, and $r_{1.2.2}$ are extended with propagated context concerning the rules $r_{1.1.1s}$, $r_{1.1.1s'}$, and $r_{1.2.2s}$ and in addition have to reference the new rules.

Now we are ready to prove that for each SGDT there exists an equivalent NGC and consequently also that the languages $\mathcal{L}_{SGDT}$ and $\mathcal{L}_{NGC}$ are equivalent.

Proposition 22. *Given, a SGDT $sgdt_L$ for a graph L typed over TG, then there exists a NGC c_L s.t. $sgdt_L \sim c_L$.*

Proof. Because of Lemma 21 we can assume w.l.o.g. that $sgdt_L$ has maximal context. We perform the proof by induction on the height of the tree.
Base case: *If $sgdt_L$ has height 0, then it consists merely of some terminal rule without any PACs or NACs. Then $ans_{gdn}(sgdt_L, G)$ consists of all matches of the terminal rule into G. If we choose c_L equal to true over L then it returns exactly the same set of morphisms s.t. $sgdt_L \sim c_L$.*
Induction step: *Suppose that $sgdt_L$ has height $n+1$ and that it has terminal rule $r = \langle p : L \to R, \nexists p \wedge \bigwedge_{j\in J}(\exists p_j : L \to P_j) \wedge \bigwedge_{k\in K}(\nexists n_k : L \to N_k)\rangle$. Then we have a subtree $sgdt_{L_j}$ and $sgdt_{L_k}$ for each p_j and each n_k, respectively. Because of induction hypothesis it holds that for each $sgdt_{L_j}$ and $sgdt_{L_k}$ there exists an equivalent NGC c_{L_j} and c_{L_k}, respectively. Since $sgdt_L$ has maximal context, we moreover know that there exist morphisms $l_j : L \to L_j$ and $l_k : L \to L_k$. Consider the NGCs $c^j_L = \exists(l_j, c_{L_j})$ and $c^k_L = \nexists(l_k, c_{L_k})$ such that $c_L = \wedge_{j\in J} c^j_L \wedge \wedge_{k\in K} c^k_L$. Now $ans_{GDN}(sgdt_L, G)$ for some G consists of all morphisms $o : L \to G$ such that the terminal rule of each $sgdt_{L_j}$ and $sgdt_{L_k}$ has been applied and not been applied, respectively. The latter is equivalent with the fact that for each $j \in J$ a morphism $o_j : L_j \to G$ exists s.t. $o_j \circ l_j = o$ with $o_j \in ans_{GDN}(sgdt_{L_j}, G) = ans_{NGC}(c_{L_j}, G)$. Analogously for each $k \in K$ there does not exist a morphism $o_k : L_k \to G$ s.t. $o_k \circ l_k = o$ and $o_k \in ans_{GDN}(sgdt_{L_k}, G) = ans_{NGC}(c_{L_k}, G)$. This is exactly what also each morphism $o : L \to G$ in $ans_{NGC}(c_L, G)$ needs to fulfill s.t. we can conclude that $sgdt_L \sim c_L$.*

Theorem 23. $\mathcal{L}_{SGDN} \sim \mathcal{L}_{SGDT} \sim \mathcal{L}_{NGC}$

Proof. From Propositions 18 and 22 we can follow directly that $\mathcal{L}_{SGDT} \sim \mathcal{L}_{NGC}$. From Lemma 13 we can conclude that $\mathcal{L}_{SGDN} \sim \mathcal{L}_{SGDT}$.

5 Discussion

In this section, we will discuss a more expressive variant, a minimal variant, as well as some observations and implications for optimization of graph queries and incremental updates concerning GDNs and the proposed SGDNs.

In particular, we can show that for *minimal* SGDT (MSGDT) – SGDT with at most two direct dependencies per SMR, where all rules adhere to one of the four rule schemes introduced in Lemmata 14, 15, 16, and 17, and where in addition all rules for existential quantification are limited to at most one additional element in form of a node or edge – holds that $\mathcal{L}_{MSGDT} \sim \mathcal{L}_{NGC}$ (see [5]) and thus the additional restrictions do not result in any loss of expressive power. As often the tree-like simplification is not wanted, we further name SGDN that are not MSGDT but fulfill all conditions besides the tree nature as MSGDN.

There are several approaches for optimization of graph queries or incremental updates of graph queries based on RETE networks (cf. [10]) such as [7] and VIATRA [4] that can be conceptually mapped to MSGDN. In these cases the RETE network structure supports only at most two direct dependencies like MSGDN and the computations of the nodes of the RETE network can be matched to the four permitted cases of MSGDN. Our results also indicate that these approaches have the same expressiveness as NGC.

In our own practical work on graph queries [6], we conceptually employ SGDN with marking rules in form of graph transformation rules for optimization of queries and incremental updates of graph queries. We were able to show that the more powerful capabilities of a single node (marking rule) and advanced dynamic pattern matching strategies [11] can lead to considerable improvements concerning the computation speed and memory consumption for SGDN compared to the restricted case of MSGDN (resp. RETE network). Similar results have been obtained also in the relational case where it has been shown that the more general GATOR networks can outperform RETE networks [13]. Consequently, it seems reasonable to study the broader class of SGDN for optimization of queries and incremental updates of graph queries and not more restricted forms such as MSGDN or MSGDT. In particular the context propagation (see Definition 19) and its inverse context elimination seem useful tools here to minimize the effort for subqueries and the propagation of their results in the network.

As outlined in [5] in more detail, we can also have *more expressive* generalized discrimination networks as given in Definition 9 for which we can show that they will not lead to an increase of expressive power such that the language equivalence $\mathcal{L}_{GDN} \sim \mathcal{L}_{NGC}$ holds. However this result only applies unless we leave the realm of pattern-based property specification concepts such as NGC and consider also path-related properties [15] or we permit cycles in the network in a controlled manner as in our own practical work on graph queries [6] to be able to support path-related properties (analogously to the controlled and repeated rule applications to support path-related properties used in [3]).

6 Conclusion and Future Work

Analog to the relational database case where the relational calculus and generalized discrimination networks have been studied to find appropriate decompositions into subqueries and ordering of these subqueries for query evaluation or incremental updates of queries, we present in this paper GDN for graph queries a simple operational concept where graph transformation describe the node behavior. We further show that the proposed GDNs in different forms all have the same expressive power as NGC.

We plan to study in our future work the complexity of evaluating and updating SGDN, their optimization, and possible extensions of SGDNs towards path-related properties to also formally cover our own practical work on graph queries [6] supporting cycles in the network.

Acknowledgments. We are grateful to Johannes Dyck for his contribution to our discussions and feedback to draft versions of the paper.

References

1. Abiteboul, S., Hull, R., Vianu, V. (eds.): Foundations of Databases: The Logical Level, 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston (1995)
2. Angles, R.: A comparison of current graph database models. In: Proceedings of the 28th International Conference on Data Engineering, pp. 171–177. IEEE (April 2012)
3. Becker, B., Lambers, L., Dyck, J., Birth, S., Giese, H.: Iterative development of consistency-preserving rule-based refactorings. In: Cabot, J., Visser, E. (eds.) ICMT 2011. LNCS, vol. 6707, pp. 123–137. Springer, Heidelberg (2011)
4. Bergmann, G., Ökrös, A., Ráth, I., Varró, D., Varró, G.: Incremental pattern matching in the viatra model transformation system. In: Proceedings of the 3rd International Workshop on Graph and Model Transformations, GRaMoT 2008, pp. 25–32. ACM (2008)
5. Beyhl, T., Blouin, D., Giese, H., Lambers, L.: On the Operationalization of Graph Queries with Generalized Discrimination Networks. Technical report 106, Hasso Plattner Institute at the University of Potsdam (2016)
6. Beyhl, T., Giese, H.: Incremental view maintenance for deductive graph databases using generalized discrimination networks. In: Electronic Proceedings in Theoretical Computer Science, Graphs as Models 2016 (2016, to appear)
7. Bunke, H., Glauser, T., Tran, T.H.: An efficient implementation of graph grammars based on the RETE matching algorithm. In: Kreowski, H.-J., Ehrig, H., Rozenberg, G. (eds.) Graph Grammars 1990. LNCS, vol. 532, pp. 174–189. Springer, Heidelberg (1991)
8. Council, L.D.B.: LDBC Social Network Benchmark (SNB) - First Public Draft Release v0.2.2 (2015). https://github.com/ldbc/ldbc_snb_docs/blob/master/LDBC_SNB_v0.2.2.pdf
9. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, Heidelberg (2006)
10. Forgy, C.L.: Rete: a fast algorithm for the many pattern/many object pattern match problem. Artif. Intell. **19**(1), 17–37 (1982)
11. Giese, H., Hildebrandt, S., Seibel, A.: Improved flexibility and scalability by interpreting story diagrams. In: Magaria, T., Padberg, J., Taentzer, G. (eds.) Proceedings of the 8th International Workshop on Graph Transformation and Visual Modeling Techniques, vol. 18. Electronic Communications of the EASST (2009)
12. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. Math. Struct. Comput. Sci. **19**, 1–52 (2009)
13. Hanson, E.N., Bodagala, S., Chadaga, U.: Trigger condition testing and view maintenance using optimized discrimination networks. Trans. Knowl. Data Eng. **14**(2), 261–280 (2002)

14. He, H., Singh, A.K.: Graphs-at-a-time: query language and access methods for graph databases. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 405–418. ACM (2008)
15. Poskitt, C.M., Plump, D.: Verifying monadic second-order properties of graph programs. In: Giese, H., König, B. (eds.) ICGT 2014. LNCS, vol. 8571, pp. 33–48. Springer, Heidelberg (2014)
16. Wood, P.T.: Query languages for graph databases. SIGMOD Rec. **41**(1), 50–60 (2012)