On the Operationalization of Graph Queries with Generalized Discrimination Networks

Thomas Beyhl, Dominique Blouin, Holger Giese,
Leen Lambers

Technische Berichte Nr. 106

des Hasso-Plattner-Instituts für
Softwaresystemtechnik
an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 106

Thomas Beyhl | Dominique Blouin | Holger Giese | Leen Lambers

On the Operationalization of Graph Queries with Generalized Discrimination Networks

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de/ abrufbar.

Universitätsverlag Potsdam 2016
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

ISBN 978-3-86956-372-5

Zugleich online veröffentlicht auf dem Publikationsserver der Universität Potsdam:
URN urn:nbn:de:kobv:517-opus4-96279
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-96279

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 query results. 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.

Contents

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 [15] 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, 18]). 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. [14]) as a formal framework to study which decomposition into subqueries and ordering of these subqueries is most appropriate. As depicted in Figure 1 (a), in such a network each network 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.

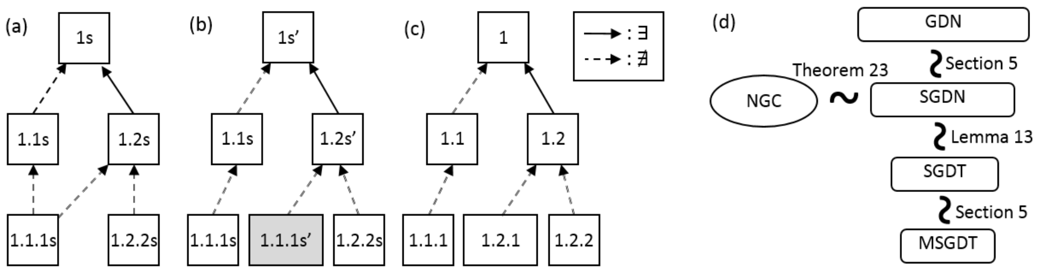

Figure 1: GDNs in form of a SGDN (a) and SGDTs (b)(c) for a social network query

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 network nodes, we suggest employing *graph transformation*

(GT) rules for these network 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) [13]. 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] 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 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 Section 2. Then, in Section 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 Section 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 Section 5. Finally, we conclude the paper and provide an outlook on planned future work.

This report is an extended version of [5]. In addition, it includes an appendix with a complete description of our running example as well as omitted results and proofs.

[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. Moreover, 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.

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.

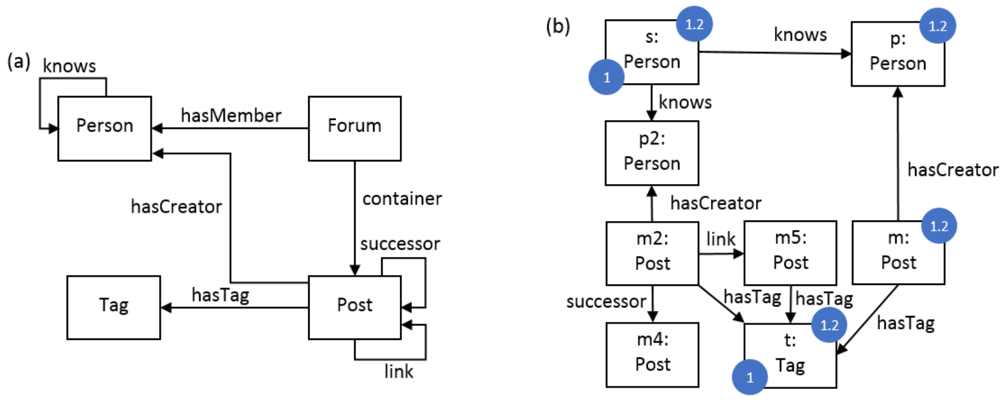

Figure 2: Excerpt of social network type graph and an example graph G

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 [16]. A class diagram outlining the possible graph models as well as an example graph to apply the query are depicted in Figure 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) (1.2.2) 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 1.2.1). Note that the employed numbering of the conditions relates to the tree-like network depicted in Figure 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 Figure 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 Figure 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 1 (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 2 (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 [13], 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.[2]

Definition 3 ($\mathcal{L}_{NGC}$, 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 $\mathcal{L}(TG)$.

An example NGC for the social network query of Example 1, where the subconditions refer to the introduced numbering, is the following: $c_1 = c_{1.1} \wedge c_{1.2}$

[2]W. l. o.g. we restrict our notion of condition satisfaction to the existence of monomorphisms. In particular, in [13] 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.

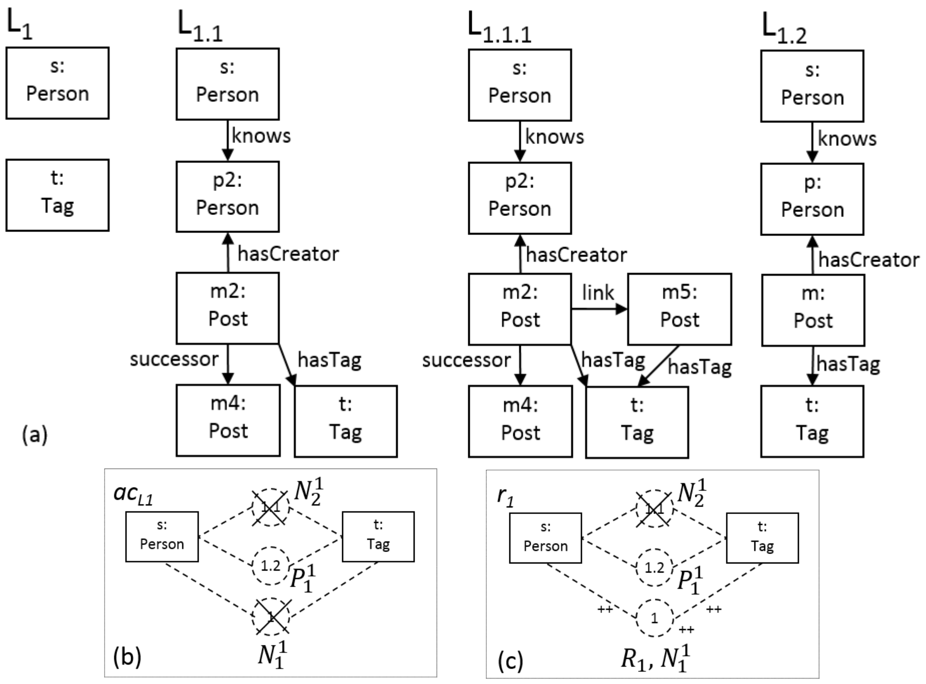

Figure 3: Graphs for the NGC c_1 and its subconditions (a) and the application condition $ac_{L_1} = \exists(L_1 \to P_1^1) \wedge \nexists(L_1 \to N_1^1) \wedge \nexists(L_1 \to N_2^1)$ (b) and simple marking rule $r_1 = (L_1 \to R_1, ac_{L_1})$ (c)

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 Section B for the complete example) in Figure 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) [13]. 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 Figure 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 Figure 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 Figure 3 (c) as it equals R_1. Note that we use in this example in addition to the node types defined in the type graph depicted in Figure 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 $G \not\Rightarrow_{\text{gts}}$. A GTS *with priorities* $\text{gts}_p = ((\mathcal{R}, TG), p)$ consists of a GTS $(\mathcal{R}, TG)$ and a transitive and asymmetric relation $p \subset \mathcal{R} \times \mathcal{R}$. We write $G \Rightarrow_{\text{gts}_p} G'$ 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* $\texttt{REACH}(\text{gts}_p, G_0)$ is defined as $\{G \mid G_0 \Rightarrow^*_{\text{gts}_p} G\}$ and the *set of terminal reachable graphs* $\texttt{TERM}(gts_p, G_0)$ is defined as $\{G | G \in \texttt{REACH}(\text{gts}_p, G_0) \colon G \not\Rightarrow_{gts_p}\}$.

3. Generalized Discrimination Networks

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

Example 2 (GDN (informal)). *A possible GDN for the social network query Example 1 is depicted in Figure 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 Figure 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 2 for rule r_1, which captures that a s:Person and t:Tag exist for which additional conditions must hold, are depicted in Figure 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 4 (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 5 ((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 6 (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 2 are depicted in Figure 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

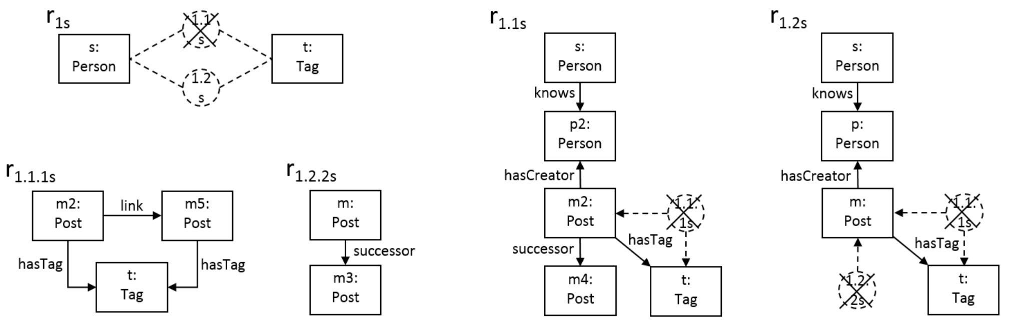

Figure 4: SMRs for the SGDN of the social network example

as presented in Figure 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 network nodes of a GDN, we can now introduce our form of GDN or SGDN.

Definition 7 (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 Figure 1 (a) and 4, where Figure 1 (a) shows the dependencies between the nodes and Figure 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 1 (unique terminal graph)**.** *Given a GDN $gdn_L = ((\mathcal{R}, TG'), \rightsquigarrow_d^+)$ for L typed over TG, then $\text{TERM}(gdn_L, G)$ consists of exactly one graph.*

Proof. (sketch; more details see Section A.1) 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 $\leadsto_d^+$ exclude conflicting applications of different rules and acyclicity of $\leadsto_d^+$ excludes conflicting applications of a rule with itself, gdn_L is also confluent. □

Definition 8 (ans_{GDN})**.** *Given the graph query language $\mathcal{L}_{GDN}$, the* answer set mapping *ans_{GDN} for $\mathcal{L}_{GDN}$ is given by*

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

with $gdn_L = ((\mathcal{R}, TG'), \leadsto_d^+)$ 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 $\{G'\} = \mathit{TERM}(gdn_L, G)$.

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 $\leadsto_d^+$ and acyclicity of $\leadsto_d^+$ (as mentioned also w. r. t. terminal graph uniqueness). Moreover, for $o' : L \to G_i$ 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 Section A.4).

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 2) in the following restrict to tree-like networks.

Definition 9 (SGDT, $\mathcal{L}_{SGDT}$)**.** *A* simple generalized discrimination tree (SDGT) *is a SGDN $sgdn_L = ((\mathcal{R} = (r_i)_{i \in I}, TG'), \leadsto_d^+)$ such that (1) for each $(r_i, r_j) \in \leadsto_d$ no $k \in I$ with $k \neq i$ exists s.t. $(r_k, r_j) \in \leadsto_d$ 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 2 ($\mathcal{L}_{SGDN} \sim \mathcal{L}_{SGDT}$)**.** *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 Section A.2) We can show by induction over the depth of $\leadsto_d^+$ 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 $\mathcal{L}_{SGDN} \sim \mathcal{L}_{SGDT}$. □

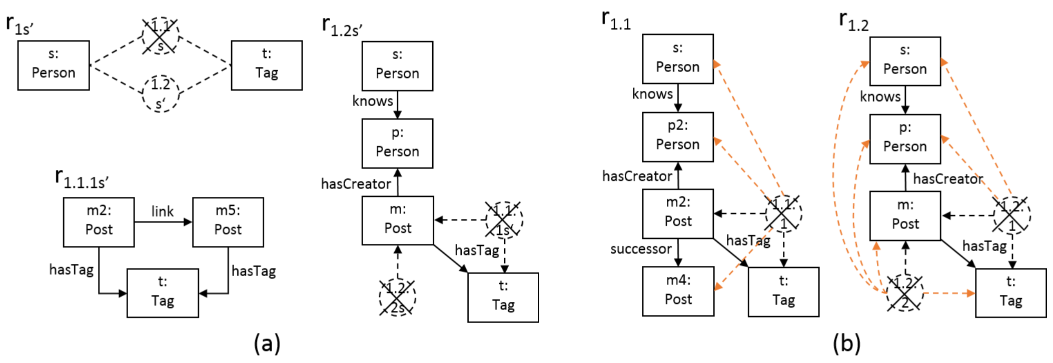

Figure 5: SMRs for the SGDT for the social network example (a) and with maximal context (b) as denoted by the orange dashed lines

The SMRs of the SGDT related to the SGDN of Figure 1 (a) depicted in Figure 1 (b) where multiple referenced SMRs are simply replicated are presented in Figure 5 (a). The rules $r_{1.1s}$, $r_{1.1.1s}$, and $r_{1.2.2s}$ of Figure 4 are not shown in Figure 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 Figure 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 3 (*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 4 ($\exists(a : L \to L', c_{L'})$). *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 $sgdt_L$ from $sgdt'_{L'}$ by adding as terminal rule the rule $r_{L,\exists a} = \langle p : L \to R, \nexists p \wedge \exists(p'_{t_{|a(L)}} \circ a', true) \rangle$ with $a' : L \to a(L)$ such that a' is identical with a. 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 $\exists(p'_{t_{|a(L)}} \circ a', true)$[3] 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 5 ($\neg c_L$)**.** *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 6 ($c_{1,L} \wedge c_{2,L}$)**.** *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.

[3]In [5] we had the PAC $\exists(p'_t \circ a', true)$ violating the restricted form of PACs allowed in a simple marking rule. This slightly corrected but in this case equivalent version of the PAC restricts the codomain of a and the domain of p'_t such that exactly one marking node together with marking edges referring to $a'(L) = a(L)$ are required.

Proposition 1 (emulate NGC by SGDT). *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 3 it follows that for $c_L = true$ with nesting level 0 an equivalent SGDT with a single marking rule exists. From Lemma 5 and 6 it follows that for any combination of conditions of nesting level 0 we can still construct an equivalent SGDT.
Induction step: By Lemma 4 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 Lemma 5 and 6 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 10 (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 7 (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 Def. 10), 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 7) 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. $\square$

Lemma 8 (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 7 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. $sgdt'_L \sim sgdt_L$. $\square$

Two of the modified SMRs of the SGDT depicted in Figure 1 (c) with maximal context related to the SGDN of Figure 1 (a) are presented in Figure 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 Figure 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 2 (emulate SGDT by NGC). *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 8 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 $c_L^j = \exists(l_j, c_{L_j})$ and $c_L^k = \nexists(l_k, c_{L_k})$ such that $c_L = \wedge_{j \in J} c_L^j \wedge \wedge_{k \in K} c_L^k$. 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 1 (language equivalence). $\mathcal{L}_{SGDN} \sim \mathcal{L}_{SGDT} \sim \mathcal{L}_{NGC}$

Proof. From Proposition 1 and 2 we can follow directly that $\mathcal{L}_{SGDT} \sim \mathcal{L}_{NGC}$. From Lemma 2 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 3, 4, 5, and 6, 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 Section A.6) 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. [11]) 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 [12] 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 [14]. 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 10) and its inverse context elimination seem useful tools here to minimize the effort for subqueries and the propagation of their results in the network.

We can also have *more expressive* generalized discrimination networks as given in Def. 7 for which we can show (see Section A.5) that they will not lead to an increase of expressive power such that the language equivalence $\mathcal{L}_{GDN} \sim \mathcal{L}_{NGC}$ holds. In particular, the use of NGCs as application conditions in the marking rules results in much more complex direct dependencies. Thus the discussed increase in expressive power of the marking rules will not increase the expressive power of the discrimination network.

The study of more powerful marking rules may still be useful when efficient matching algorithms for the supported fragment of NGCs can be employed. It is to be noted, that in case of incremental updates for graph queries the limitation to simple graph rules seems necessary to be able to propagate update effects without the need to maintain expensive additional data structures to detect the need for updates (cf., for example, instance-based scopes in [8]), while in case of optimizations of graph queries this restriction may be not always helpful.

Furthermore the language equivalence $\mathcal{L}_{GDN} \sim \mathcal{L}_{NGC}$ only applies unless we leave the realm of pattern-based property specification concepts such as NGC and consider also path-related properties [17] 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 presented in this paper GDNs for graph queries with simple operational concepts where graph transformations describe the node behavior. We further showed 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 SGDNs, 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
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.

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

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu, editors. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1995.

[2] Renzo Angles. A Comparison of Current Graph Database Models. In *Proc. of the 28th Int. Conf. on Data Engineering*, pages 171–177. IEEE, April 2012.

[3] Basil Becker, Leen Lambers, Johannes Dyck, Stefanie Birth, and Holger Giese. Iterative Development of Consistency-Preserving Rule-Based Refactorings. In Jordi Cabot and Eelco Visser, editors, *Theory and Practice of Model Transformations, Fourth International Conference, ICMT 2011, Zurich, Switzerland, June 27-28, 2011. Proceedings*, volume 6707 of *Lecture Notes in Computer Science*, pages 123–137. Springer / Heidelberg, 2011.

[4] Gábor Bergmann, András Ökrös, István Ráth, Dániel Varró, and Gergely Varró. Incremental Pattern Matching in the VIATRA Model Transformation System. In *Proceedings of the 3rd International Workshop on Graph and Model Transformations*, GRaMoT '08, pages 25–32. ACM, 2008.

[5] Thomas Beyhl, Dominique Blouin, Holger Giese, and Leen Lambers. On the Operationalization of Graph Queries with Generalized Discrimination Networks. In Rachid Echahed and Mark Minas, editors, *Proceedings of the 9th International Conference on Graph Transformations*, pages 170–186. Springer, 2016.

[6] Thomas Beyhl and Holger Giese. Incremental View Maintenance for Deductive Graph Databases using Generalized Discrimination Networks. In *Graphs as Models 2016*. Electronic Proceedings in Theoretical Computer Science, 2016. to appear.

[7] H. Bunke, T. Glauser, and T.-H. Tran. An efficient implementation of graph grammars based on the RETE matching algorithm. In Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Graph Grammars and Their Application to Computer Science*, volume 532 of *LNCS*, pages 174–189. Springer, 1991.

[8] Alexander Egyed. Instant Consistency Checking for the UML. In *ICSE '06: Proceedings of the 28th International Conference on Software Engineering*, pages 381–390, Shanghai, China, 20–28 May 2006.

[9] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.

[10] Hartmut Ehrig, Ulrike Golas, Annegret Habel, Leen Lambers, and Fernando Orejas. M-Adhesive Transformation Systems with Nested Application Conditions, Part 2: Embedding, Critical Pairs and Local Confluence. *Fundamenta Informaticae*, 118(1-2):35–63, 2012.

[11] Charles L. Forgy. Rete: A Fast Algorithm for the Many Pattern/Many object Pattern Match Problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[12] Holger Giese, Stephan Hildebrandt, and Andreas Seibel. Improved Flexibility and Scalability by Interpreting Story Diagrams. In Tiziana Magaria, Julia Padberg, and Gabriele Taentzer, editors, *Proc. of the 8th International Workshop on Graph Transformation and Visual Modeling Techniques*, volume 18. Electronic Communications of the EASST, 2009.

[13] Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19:1–52, 2009.

[14] Eric N. Hanson, Sreenath Bodagala, and Ullas Chadaga. Trigger Condition Testing and View Maintenance Using Optimized Discrimination Networks. *Transactions on Knowledge and Data Engineering*, 14(2):261–280, Mar 2002.

[15] Huahai He and Ambuj K. Singh. 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*, pages 405–418. ACM, 2008.

[16] Linked Data Benchmark Council, Coordinator: Arnau Prat (UPC). 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, accessed October 2016.

[17] Christopher M. Poskitt and Detlef Plump. Verifying monadic second-order properties of graph programs. In *Proc. International Conference on Graph Transformation (ICGT 2014)*, volume 8571 of *LNCS*, pages 33–48. Springer, 2014.

[18] Peter T. Wood. Query Languages for Graph Databases. *SIGMOD Rec.*, 41(1):50–60, April 2012.

Appendix

The following appendix presents omitted proofs and results as well as more details for the presented examples.

A. Omitted Results and Proofs

A.1. Proof of Lemma 1

Proof of Lemma 1: 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.

Proof. Since each marking rule creates a node and is non-deleting it holds that the number of nodes is strictly monotonically increasing. Moreover, there is an upper bound on the nodes that can be created since for each marking rule $r_i = \langle p_i : L_i \rightarrow R_i, \nexists p_i \wedge c_{L,i} \rangle$ the NAC $\nexists p_i$ excludes that the rule can be applied multiple times for the same match. Since we have non-deleting rules matches cannot be destroyed and having a finite number of potential matches we can follow that gdn_L terminates.

Furthermore, two non-deleting GT rules r_i and r_j are in conflict if rule r_j would be able to create elements that violate the application condition of r_i (or vice versa) [10]. Considering r_i, r_j to be marking rules this could only happen if r_j creates marking elements referred to by r_i, which is only the case if $(r_i, r_j) \in \rightsquigarrow_d$ (or vice versa). For $i \neq j$ this would mean that by the priority mechanism r_i and r_j would not be applicable to the same graph in the first place. For $i = j$ we would have that $(r_i, r_i) \in \rightsquigarrow_d$, which is in contradiction with acyclicity of $\rightsquigarrow_d^+$. Thus gdn_L is confluent and also terminates with a unique result.

□

A.2. Proof of Lemma 2

Proof of Lemma 2: 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. We show the above lemma by induction over the depth of $\rightsquigarrow_d^+$ for some given $sgdn_L = ((\mathcal{R}, TG'), \rightsquigarrow_d^+)$.
Base case: If the depth of $\rightsquigarrow_d^+$ equals 0 $sgdn_L$ is trivially already a SGDT. *Induction step*: Suppose that $\rightsquigarrow_d^+$ has depth $n+1$ and that $r = \langle p : L \rightarrow R, \nexists p \wedge \bigwedge_{j \in J}(\exists p_j : L \rightarrow P_j) \wedge \bigwedge_{k \in K}(\nexists n_k : L \rightarrow N_k) \rangle$ is the terminal rule of $sgdn_L$. Then for each r_j such that $(r, r_j) \in \rightsquigarrow_d$ consider the SGDN of depth n $sgdn_{L_j}^{r_j} = ((\mathcal{R}_{r_j}, TG'), \rightsquigarrow_{d,r_j}^+)$

with $\mathcal{R}_{r_j} = \{r_j\} \cup \{r_i | (r_j, r_i) \in \rightsquigarrow_d^+\}$ and $\rightsquigarrow_{d,r_j}^+ = \{(r_i, r_i') | (r_i, r_i') \in \rightsquigarrow_d^+ \wedge r_i, r_i' \in \mathcal{R}_{r_j}\}$. We can now replace in $sgdn_L$ by induction hypothesis each SGDN $sgdn_{L_j}^{r_j}$ by an equivalent SGDT $sgdt_{L_j}^{r_j}$ obtaining the SGDN $sgdn_L'$, which fulfills property (1), but not yet property (2) of SGDTs as given in Def. 9. We still need to ensure that no two PACs/NACs of the terminal rule r refer to identical marking node types. Thus consider for each r_j such that $(r, r_j) \in \rightsquigarrow_d$ all PACs/NACs in r referring to the marking node type t_j of rule r_j. If there are at least two, then we build for each such PAC/NAC an equivalent copy of $sgdt_{L_j}^{r_j}$ with new disjoint marking types. We replace in $sgdn_L'$ each subtree $sgdt_{L_j}^{r_j}$ by these copies and retype the PACs and NACs in the terminal rule r accordingly. In this way we obtain an equivalent SGDT $sgdt_L$ from the SGDN $sgdn_L$.

Since each SGDT is in particular also a SGDN, it directly follows that $\mathcal{L}_{SGDN} \sim \mathcal{L}_{SGDT}$. □

A.3. Proof of Lemma 7

Proof of Lemma 7: 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_j^i (or V_k^i) (as given in Def. 10), then there exists some $sgdt_L^c$ in which (r_i, r_l) has been replaced by a SGDT with maximal context such that $sgdt_L^c \sim sgdt_L$.

Proof. Given a SGDT $sgdt_L$ with marking rule set $\mathcal{R} = (r_i)_{i \in I}$ for a graph L typed over TG with two SMRs $r_i = \langle p_i : L_i \to R_i, \nexists p_i \wedge \bigwedge_{j \in J_i}(\exists p_j^i : L_i \to P_j^i) \wedge \bigwedge_{k \in K_i}(\nexists n_k^i : L_i \to N_k^i)\rangle$ and $r_l = \langle p_l : L_l \to R_l, \nexists p_l \wedge \bigwedge_{j \in J_l}(\exists p_j^l : L_l \to P_j^l) \wedge \bigwedge_{k \in K_l}(\nexists n_k^l : L_l \to N_k^l)\rangle$ 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_j^i (or n_k^i, resp.) uses a type equal to t_l, we first introduce the construction of the SGDT $sgdt_L^c$. Now $r_{i,j}^c$ ($r_{i,k}^c$) equals a rule r_i, where $\exists p_j^i : L_i \to P_j^i$ (or $\nexists n_k^i : L_i \to N_k^i$) has been replaced by the disjunction of $\exists p_j^{i,c} : L_i \to P_j^{i,c}$ (or conjunction of $\nexists n_k^{i,c} : L_i \to N_k^{i,c}$) for all possible $p_j^{i,c}$ (or $n_k^{i,c}$) with $P_j^{i,c}$ (or $N_k^{i,c}$) equal to P_j^i (or N_j^i) s.t. the unique node with node type t_l holds in addition one outgoing edge for each node in $p_j^{i,c}(V_j^i)$ ($n_k^{i,c}(V_k^i)$), respectively. Moreover, consider each $r_{l,j}^c$ constructed from r_l, p_j^i and $M(P_j^i)$ being the graph of P_j^i consisting of the marking node l and all outgoing edges with incident nodes in L_i according to the following diagram s.t. all diagrams commute, e and e' are jointly epimorphic, (1) is a pushout constructed as a pushout complement from e and p_l and (2) is a pushout constructed from p and i'':

$$\begin{tikzcd}
 & & & M(P^i_j) \arrow[r, "i"'] \arrow[dl, "i'"] & P^i_j \arrow[ddl, "e'"'] \\
P \arrow[d] \arrow[dr, phantom, "(2)"] & L_l \arrow[l, "p"'] \arrow[r, "p_l"] \arrow[d, "i''"] \arrow[dr, phantom, "(1)"] & R_l \arrow[dr, "e"] & & L_i \arrow[u, "p^i_j"'] \arrow[dl, "e' \circ p^i_j"] \\
P^c & L^c_{l,j} \arrow[l, "p^c"'] \arrow[rr, "p^c_{l,j}"] & & R^c_{l,j} &
\end{tikzcd}$$

In this diagram the plain rule morphism of $r^c_{l,j}$ equals $p^c_{l,j}$ and each PAC (or NAC) from r_l depicted by p is translated into a corresponding equivalent PAC (or NAC) p^c for $r^c_{l,j}$ by building PO (2). Analogously, each $r^c_{l,k}$ is constructed from r_l, n^i_k and $M(N^i_k)$. Note that this diagram defines a unique monomorphism $(p^c_{l,j})^{-1} \circ e' \circ p^i_j$ from L_i into $L^c_{l,j}$.

Let gdt^c_L now be the GDT obtained from $sgdt_L$ by replacing (r_i, r_l) by all pairs $(r^c_{i,j}, r^c_{l,j})$ (or $(r^c_{i,k}, r^c_{l,j})$, resp.). Now SGDT $sgdt^c_L$ is the SGDT that can be obtained from the GDT gdt^c_L by transforming according to Proposition 1 the application condition with the disjunction (not adhering to the SMR scheme) in $r^c_{i,j}$ into a SGDT and replacing it by a simple PAC referring to the terminal rule of the corresponding tree.

By construction in $sgdt^c_L$ the marking rules $r^c_{l,j}$ (or $r^c_{l,k}$) 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 later by rule r_i after all matches for r_l had been found. Marking these elements earlier does not change the overall answer set. □

A.4. From Marking Nodes to Graph Morphisms

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'. In particular, we want to reconstruct where in G' the terminal rule has been matched without having to know in which direct transformation in t this happened. In the following lemma we argue that this is possible if in the queried graph G no parallel edges with the same type occur. Having parallel edges with the same type in G can be emulated by replacing these edges by a node with two outgoing edges describing the source and target mapping of the edge.

Lemma 9 (extracting query result from markings). *Given a graph G typed over TG without parallel edges of the same type and a GDN $gdn_L = ((\mathcal{R}, TG'), \rightsquigarrow^+_d)$ for a* *graph L typed over TG such that $\{G'\} = \textsf{TERM}(gdn_L, G)$, then $ans_{GDN}(gdn_L, G)$ can be reconstructed uniquely from G'.*

Proof. We know that for a transformation $t : G \Rightarrow^*_{gdn_L} G'$ and the terminal rule r_t for gdn_L with marking node t it holds that

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

In particular, each marking edge e from a marking node n of type t in G' points uniquely to a node in $o'^V(L^V)$ because of the unique typing of e w. r. t. nodes in L^V. The corresponding edge mapping $o'^E : L^E \to G'^E_i$ can be reconstructed by looking up in G' the edges in L between the already found and uniquely marked source and target nodes in G' by comparing edge types. Altogether, this leads to a unique reconstruction of each $o : L \to G$, since no parallel edges in G exist with the same type. □

A.5. $\mathcal{L}_{GDN} \sim \mathcal{L}_{NGC}$

More expressive generalized discrimination networks can be defined by allowing the marking rules to be more expressive. A natural candidate for application conditions here are NGCs as employed for non-simple GDNs in Definition 7, which result in much more complex direct dependencies. However, as we can show in the following, the discussed increase in expressive power of the marking rules will not increase the expressive power of the discrimination network.[4]

Theorem 2 (no increase of expressive power). $\mathcal{L}_{GDN} \sim \mathcal{L}_{NGC}$

Proof. (sketch) Each GDN can be transformed into an equivalent SGDN employing Proposition 1 for the application condition of each MR. Each SGDN is obviously a GDN such that $\mathcal{L}_{GDN} \sim \mathcal{L}_{SGDN}$. Since we know also that $\mathcal{L}_{SGDN} \sim \mathcal{L}_{NGC}$ the above statement follows. □

A.6. $\mathcal{L}_{MSGDT} \sim \mathcal{L}_{NGC}$

We can investigate which additional restrictions to SGDTs can be applied without loosing any expressive power.

[4]It has, however, to be noted that in case more expressive application conditions than nested graph conditions (e. g., path-related conditions) are considered, the resulting expressive power of the generalized discrimination network is increased as well.

Definition 11 (*MSGDT*, $\mathcal{L}_{MSGDT}$)**.** *A* minimal simple generalized discrimination tree *(MSGDT) is a SGDT where each simple marking rule adheres to one of the four rule schemes $r_{L,true}$, $r_{L,\exists a}$, $r_{L,\neg}$, or $r_{L,\wedge}$ introduced in Lemmata 3, 4, 5, and 6, respectively. In addition, the rule for existential quantification $r_{L,\exists a}$ is such that the codomain of a holds at most one additional node or edge w. r. t. $a(L)$. The graph query language $\mathcal{L}_{MSGDT}$ is the set of all MSGDTs.*

Lemma 10 (emulate GDN by MSGDT)**.** *For each GDN there exists an equivalent MSGDT.*

Proof. From Theorem 2 we know that for each GDN there exists some equivalent NGC. Each NGC can be transformed into an equivalent NGC using merely binary conjunction, negation, and existential quantification limited to at most one additional node or edge. In Proposition 1 and the according Lemmas it is shown that for this NGC an equivalent SGDT exists with equivalent restrictions such that in particular it is an MSGDT. □

Lemma 11 (no further simplification)**.** *Any simplification w. r. t. the form of MSGDTs leads to a class of GDNs that is too weak to be equivalent with $\mathcal{L}_{NGC}$.*

Proof. (sketch) We systematically look for all possible simplifications and show that they are not possible without loosing expressive power: a) The basic rule $r_{L,true}$ must be supported, as no simpler form of marking rule is possible. b) The existential quantification rule $r_{L,\exists a}$ must be permitted otherwise if additional nodes/edges are not permitted, the case of a NAC is not covered. c) The negation rule $r_{L,\neg}$ must be allowed, otherwise expressiveness would equal the match finding for one large graph as merely all PACs could be combined. This case would clearly not cover e. g. NACs and it is thus too weak to cover NGCs. d) The conjunction rule $r_{L,\wedge}$ must be permitted, otherwise if only a sequence of dependencies rather than binary trees are permitted, the case of an or (resp. $\neg((\neg a) \wedge (\neg b))$) cannot be covered and thus NGCs would not be covered either. □

Theorem 3 (expressiveness of MSGDTs)**.** $\mathcal{L}_{MSGDT} \sim \mathcal{L}_{NGC}$

Proof. According to Lemma 10 each GDN can be transformed into some equivalent MSGDT. Since each MSGDT is also a GDN it follows that $\mathcal{L}_{GDN} \sim \mathcal{L}_{MSGDT}$. We know that $\mathcal{L}_{GDN} \sim \mathcal{L}_{NGC}$ such that the above statement follows. □

B. Complete Example

This section presents in detail the complete complex query example of this paper including its various representations as a Nested Graph Condition (NGC), a Simple Graph Discrimination Network (SGDN) and a Simple Graph Discrimination Tree (SGDT), for which this paper showed equivalence in terms of their answer sets.

B.1. Complex Query

The type graph for the complex query is shown in Figure 6 (a) along with an example graph in (b). The query searches for pairs of Tags t and Persons s such that the Tag is new in the Posts by a friend p of s.

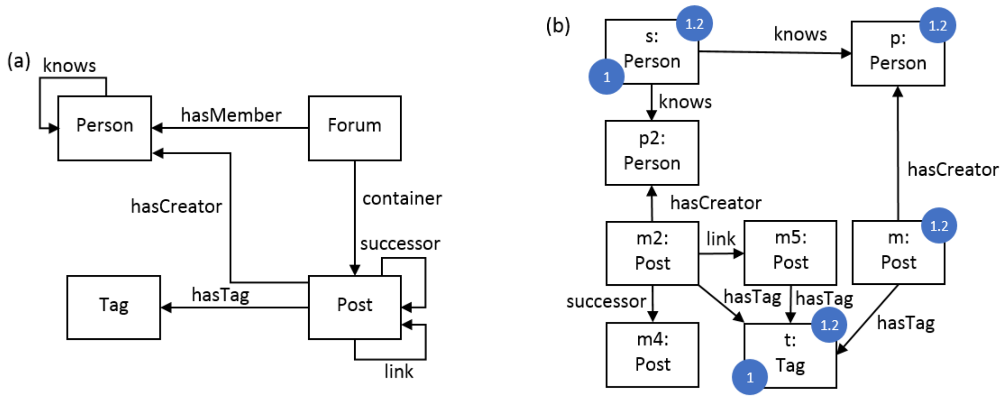

Figure 6: Excerpt of social network type graph and an example graph G

This query can be decomposed into the set of subconditions L_i illustrated by the morphisms of Figure 7, which extends Figure 3 (a). Morphisms are identified by equally named nodes.

1. In order to be a Post of a friend, the considered Post m must have been created (hasCreator) by a second Person that Person s knows, and the Tag must be linked (hasTag) to the Post ($L_{1.2}$).
   a) However, the found Tag must not be inherited from a linked Post ($m5$) ($\neg\exists L_{1.2.1}$).
   b) Furthermore, in order to be the latest Post, Post m must have no successor Post ($m3$) ($\neg\exists L_{1.2.2}$).

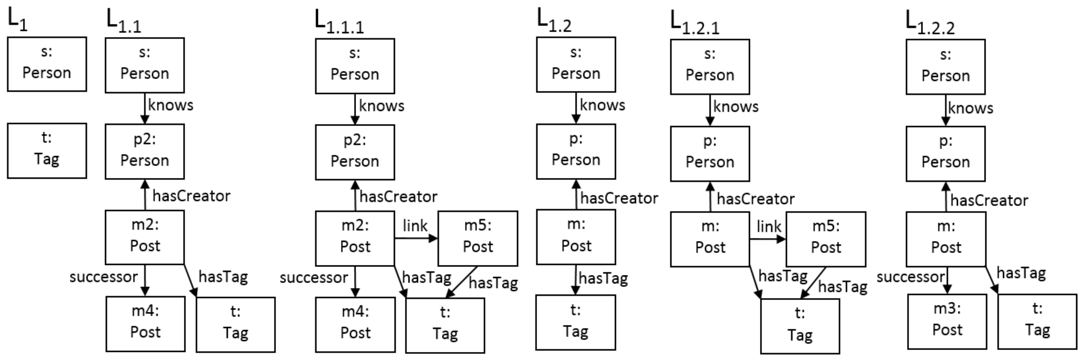

Figure 7: Complete set of graphs employed in the morphisms for the complex query and its subconditions

2. In order to be new, the Tag must not have been used by any former Post of any friend ($\neg\exists L_{1.1}$).
   a) However, similarly to 1a only Tags that are not simply inherited from a linked Post should be considered in this context ($\neg\exists L_{1.1.1}$).

Figure 6 (b) shows an example graph where occurrences for the positive sentences L_1 and $L_{1.2}$ are depicted as markers in form of blue circles with the respective number. The blue markers (1) denote the only correct answer for the query. Thereby the required match for the positive subquery $L_{1.2}$ depicted by the markers (1.2) is such that indeed no match exists for the negative subconditions $L_{1.2.1}$ and $L_{1.2.2}$. Furthermore, as required no match for the negative subcondition $L_{1.1}$ consistent with L_1 exists such that no match for the negative subcondition $L_{1.1.1}$ of $L_{1.1}$ can be found. Consequently, no match for $L_{1.1}$ is visualized.

B.2. NGC c_1

The complex query as informally stated in the previous section translates into:

NGC $c_1 = c_{1.1} \wedge c_{1.2}$

with

$c_{1.1} = \neg\exists(n_{1.1} : L_1 \rightarrow L_{1.1}, c_{1.1.1})$,

$c_{1.2} = \exists(p_{1.2} : L_1 \rightarrow L_{1.2}, c_{1.2.1} \wedge c_{1.2.2})$,

$c_{1.1.1} = \neg\exists(n_{1.1.1} : L_{1.1} \rightarrow L_{1.1.1}, true)$,

$c_{1.2.1} = \neg\exists(n_{1.2.1} : L_{1.2} \rightarrow L_{1.2.1}, true)$, and

$c_{1.2.2} = \neg\exists(n_{1.2.2} : L_{1.2} \rightarrow L_{1.2.2}, true)$

This is illustrated graphically in Figure 8 (a) where the above NGC c_1 takes an AST-like (Abstract Syntax Tree) form.

According to Proposition 1 of this paper, an equivalent Simple Graph Discrimination Tree (SGDT) can be constructed from c_1 as depicted in Figure 8 (b).

B.3. Example SGDN

The example query can also be expressed as a SGDN (Simple Graph Discrimination Network) as illustrated in Figure 9 (a), where the node $r_{1.1.1s}$ is reused as input by both $r_{1.1s}$ and $r_{1.2s}$ nodes.

For this GDN, simple Marking Rules (SMR) are defined to create markings for the found matches on the host graph when the query is executed. These marking rules are depicted in their compact notation in Figure 10.

The simple marking rule
$r_{1s} = (L_{1s} \rightarrow R_{1s}, (\exists L_{1s} \rightarrow P_1^1) \wedge (\nexists L_{1s} \rightarrow N_1^1) \wedge (\nexists L_{1s} \rightarrow N_2^1))$
and its application condition
$ac_{L_{1s}} = (\exists L_{1s} \rightarrow P_1^1) \wedge (\nexists L_{1s} \rightarrow N_1^1) \wedge (\nexists L_{1s} \rightarrow N_2^1)$
is shown graphically in Figure 11 expressed in a less compact notation.

The application condition requires with the PAC P_1^1 that rule $r_{1.2s}$ has created a marking for $L_{1.2s}$ w. r. t. L_{1s} and by the NAC N_2^1 that conversely, rule $r_{1.1s}$ did not create a marking for $L_{1.1s}$ w. r. t. L_{1s}. This is shown in the figure by dashed round circles with a crossed circle indicating a negative application as shown in the upper left corner of the figure. In addition, a marking N_1^1 must not already exist on the matched subgraph as also indicated by a cross in the upper left corner of Figure 11.

Given that the application condition holds, marking rule r_{1s} can be applied to the host graph as illustrated by the LHS and RHS of the rule in the bottom part of Figure 11.

B.4. Example SGDT

The SGDN of the previous section can be transformed into the equivalent SGDT of Figure 9 (b) according to Lemma 2. This is shown in Figure 9 (b) where the node $L_{1.1.1s}$ of (a) has been duplicated in (b) as indicated by the grayed box in order to constitute a tree.

The complete set of SMRs for this SGDT partially shown in Figure 5 (a) is presented in Figure 12.

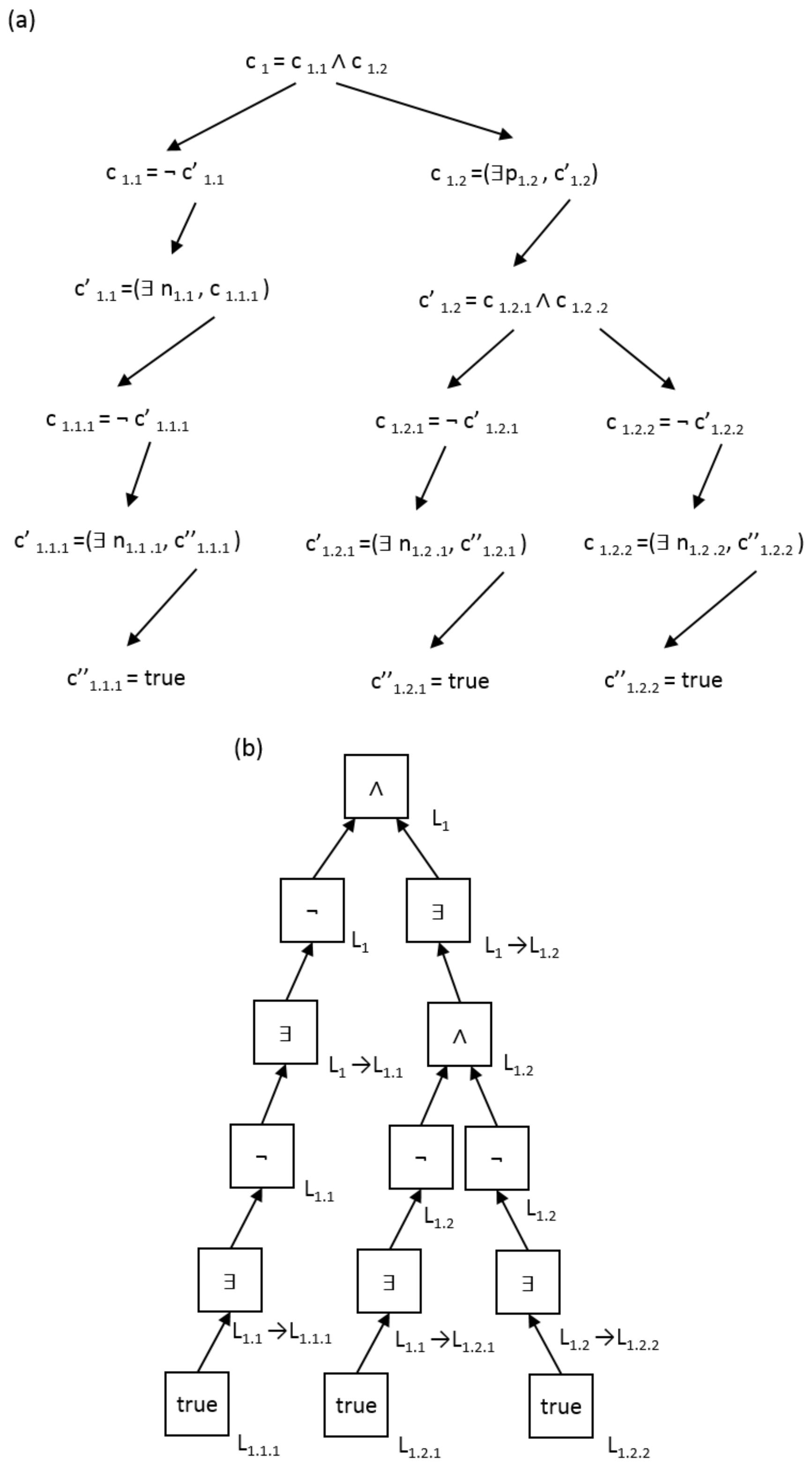

Figure 8: The NGC c_1 decomposed into its subconditions and the related SGDT constructed according to Proposition 1

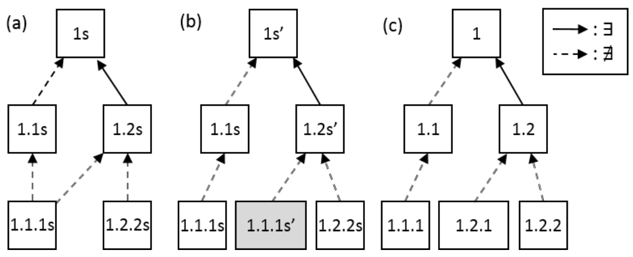

Figure 9: The SGDN decomposed into its subnodes and the related SGDT constructed according to Lemma 2

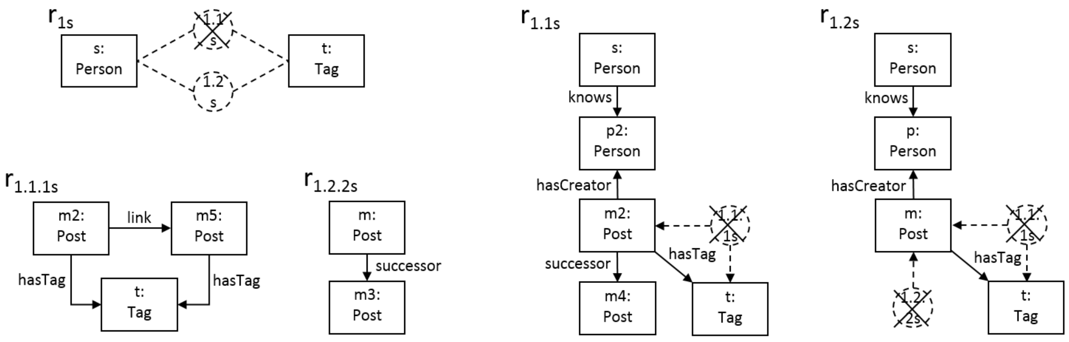

Figure 10: SMRs for the SGDN of the social network example

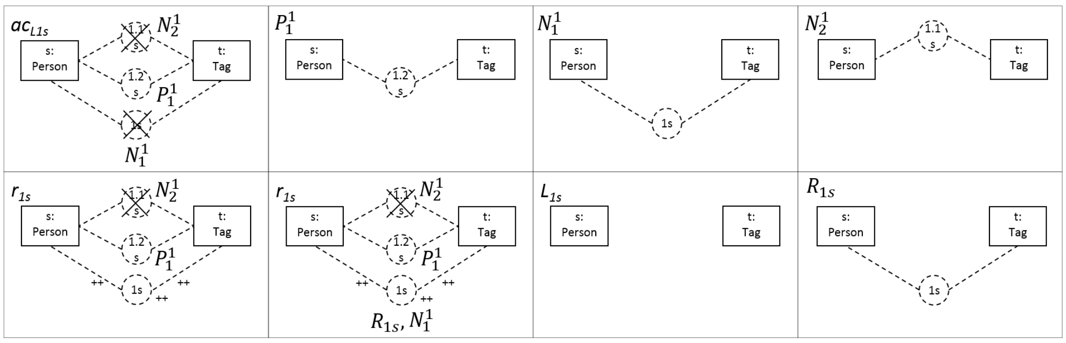

Figure 11: A diagram for the application condition $ac_{L_{1s}}$ and simple marking rule r_{1s} of the example SGDN

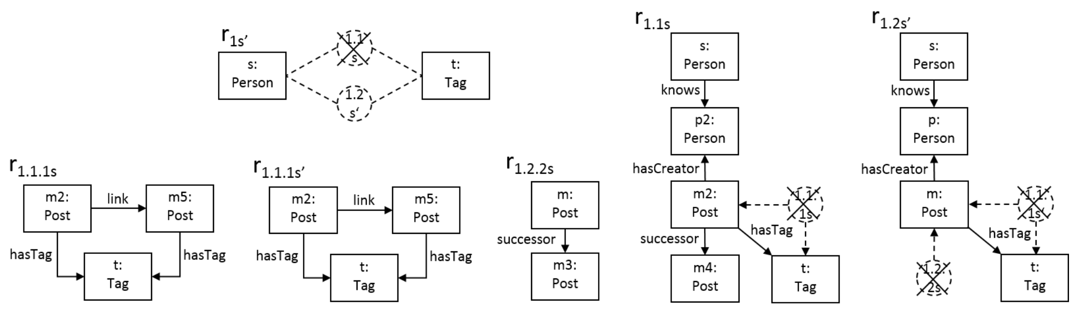
Figure 12: Complete set of SMRs for the example SGDT

B.5. SGDT with Maximal Contexts

In order to be as equally expressive as NGCs, SGDTs must be able to carry the same context provided for nested NGCs during their evaluation. For this, the SGDT needs to be extended such that SMRS also mark the elements from their parent rules (see Lemma 8 and Proposition 2). This is shown in Figure 13 where the complete set of SMRs, partially shown in Figure 5 (b) for the SGDT of Figure 9 (c) is shown. Maximal context marking links that have been added w. r. t. this maximal context generation process are displayed as orange dashed lines and corresponding available maximal context elements are identified as orange nodes and links.

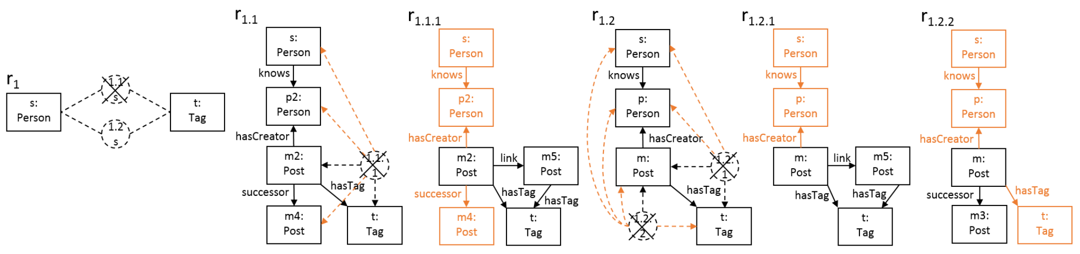
Figure 13: Complete set of SMRs for the example SGDT with maximal rule contexts

Note that for the two modified SMRs of the SGDT with maximal context, it holds that the presented rules $r_{1.1}$ and $r_{1.2}$ already have maximal context and reference directly related rules $r_{1.1.1}$, $r_{1.2.1}$, and $r_{1.2.2}$. The maximal context of these latter rules is generated according to the construction of Lemma 7, which includes an additional disjunction resp. conjunction for all alternative cases for context propagation. In particular for rule $r_{1.1.1}$ two alternative contexts were generated, but one of them can be omitted again because of the cardinality constraint that there is no succesor edge as well as a *link* edge between a Post and another Post at the same time. This constraint expresses that it is not possible to link to a successor post that can not be created yet. For rule $r_{1.2.1}$ and rule $r_{1.2.2}$ no extra alternatives arise in the first place.

B.6. Equivalence of the Example SGDT and c_1 According to Proposition 1

As depicted in Figure 8 (a) we can represent the NGC c_1 in a form consistent with the one employed in Proposition 1 where only the operators $\neg$, $\wedge$, and $\exists$ and the constant *true* for given $L_{\dots}$ are employed (see Figure 7 for the definition of the monomorphisms $n_{\dots}$ resp $p_{\dots}$). For this form of NGC, we can then also construct the equivalent SGDT according to Proposition 1 as depicted in Figure 8 (b).

ISBN 978-3-86956-372-5
ISSN 1613-5652