Operational and Abstract Semantics of the Query Language G-Log

Agostino Cortesi
Dip. di Informatica
Università Ca' Foscari
Via Torino, 155
30173 Venezia – Mestre (Italy)
cortesi@dsi.unive.it

Agostino Dovier
Dip. di Informatica
Università di Verona
Strada Le Grazie, 15
37134 Verona (Italy)
dovier@sci.univr.it

Elisa Quintarelli and Letizia Tanca
Dip. di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci, 32
20133 Milano (Italy)
quintare|tanca@elet.polimi.it

Abstract

The amount and variety of data available electronically has dramatically increased in the last decade; however, data and documents are stored in different ways and do not usually show their internal structure. In order to take full advantage of the topological structure of digital documents, and particularly web sites, their hierarchical organization should be exploited by introducing a notion of query similar to the one used in database systems. A good approach, in that respect, is the one provided by graphical query languages, originally designed to model object bases and later proposed for semi-structured data, like G-Log. The aim of this paper is to provide suitable graph-based semantics to this language, supporting both data structure variability and topological similarities between queries and document structures. A suite of operational semantics based on the notion of bisimulation is introduced both at the *concrete level* (instances) and at the *abstract level* (schemata), giving rise to a semantic framework that benefits from the cross-fertilization of tools originally designed in quite different research areas (Databases, Concurrency, Logics, Static Analysis).

1   Introduction

The amount and variety of data available electronically has dramatically increased in the last decade: such data may be structured, when coming from relational or object-oriented databases, or completely unstructured, when they consist of simple collections of text or image files. Intermediate situations arise when some kind of structure is present, as for instance in HTML files, in digital libraries or in XML documents [13]. However, a major drawback which precludes the appropriate benefits of this information richness is that the data sources stored in different forms do not usually show such internal structure.

A number of research projects are currently addressing the problem of accessing in a uniform way this plethora of *semistructured* data. Among these, we can cite LOREL [23], UnQL [5], WebSQL [24], WebOQL [3], StruQL [15].

Effectiveness and efficiency are mandatory requirements when accessing semi-structured information. Therefore, appropriate search techniques are more than necessary. Pure keyword-based search techniques proved to be ineffective, since in that setting only the document lexicon is taken into account, while the intrinsic semantics conveyed by the document structure is often lost. In practice, this leads to the retrieval of too many documents, since also the ones that do not share the required structure are often included into the result.

In order to take full advantage of the document structure, its hierarchical (or topological) organization should be somehow exploited, by introducing some notion of query like the one used in database systems, being still aware of the fact that the document structure is far from being as strict as in the usual database context.

In this paper, we refer to the graphical representation and query style of G-Log, a well-known database language for complex objects [31, 29].[1] The reason of this choice stands on observing that most of the models and languages for representing and querying semistructured information cited above share an analytical approach to data representation, lacking a synthetic notion of schema. Conversely, G-Log models semistructured information by using a concept very close to that of *database schema*, that in this context enables the user to formulate a query in an easier way. Nevertheless, the use of a schema-like facility, however desirable, should not be mandatory, since we may well imagine a situation where the user is not fully aware of the document's exact organization. In this case, assuming a strict matching between the document and the required topological structure may lead to miss some still interesting documents that do not adhere precisely to the query structure.

Our approach to attack these problems is to make the required topological similarity flexible, in order to support different similarity levels. Therefore, the aim of this paper is to illustrate effective techniques that allow the query formulator to relax and restrict topological requirements at his/her choice. Its main contribution is the design of a suite of operational and logical semantics for G-Log, based on the notion of bisimulation [27] (see also [22, 5]), given both at the instance and at the schema level. In particular, we discuss in full detail the benefits of tuning the semantics by enforcing or relaxing requirements on the bisimulation relation.

The relationship between instances and schemata is investigated using Abstract Interpretation theory [10], which provides a systematic approach to guarantee the correctness of operating on schemata with respect to the corresponding concrete computations on instances.

The revisitation of the semantics of G-Log also clarifies some subtle ambiguities in the initial semantics of G-Log queries. Since the semantics is based on the notion of bisimulation, the implementation of the language will inherit all the algorithmic properties studied in the literature [2]. In particular, Kanellakis and Smolka in [21] relate the bisimulation problem with the general (relational) coarsest partition problem, and they propose an algorithmic solution and pointed out that the partition refinement algorithms in [32] can serve, and more efficiently, to the same task. Applicability of our approach is strongly based on this efficient implementation of the bisimulation tests.

Alternative approaches to the semantics of graphical languages have been introduced in the literature. For instance, the semantics of Graphlog is given via rewriting into DATALOG. Our choice of giving directly a graph-based semantics is not only justified by the fact that this is a typical approach for visual languages, but also, and more significantly, by the fact that the expressive power of G-Log is higher than that of DATALOG.

Our work started as a part of the WG-Log project [8], which addresses the problem of Web information querying by enriching G-Log [31, 29] with constructs typical of hypermedia, like entry points, indexes, navigational edges, etc. A subsequent project [6, 7, 12], still in the area of semistructured information querying, addresses the problem of querying XML-specified information, and still investigates the possibilities

---

[1] Note that the use of graphs for representing information structure is common in the history of Database theory and in Artificial Intelligence: recall, for instance, the *entity-relationship model* [4], the *semantic networks*, the various graphical representations of object-oriented data like Good [19], and Graphlog [9], just to name a few. Moreover, computational models based on graphs transformations are used not only in Database theory: they are used as semantic domains for various kinds of formalisms and languages like, for example, actor systems, the π-calculus, functional programming, neural networks (see [20] for a survey on this topic).

[2] Bisimulation is usually attributed to Park, who introduced the term in [30], extending a previous notion of automata simulation by Milner ([25]). Milner employs bisimulation as the core for establishing observational equivalence of the Calculus of Communicating Systems (CCS) ([26, 27]). In the Modal Logic/Model Checking areas this notion was introduced by van Benthem (cf. [33]) as an equivalence principle between Kripke structures. In Set Theory, it was introduced as a natural principle replacing extensionality in the context of non well-founded sets [2].

of flexible query formulation. To this aim, in [28] the XML-GL language is translated into G-Log, in order to take advantage of the parametric semantics defined here. The results presented here for G-Log can thus be easily extended to WG-Log and XML-GL as well. As schemata can evolve gracefully with the evolution of their instances (applying abstract interpretation theory), in the extended setting of WG-Log and XML-GL, and more in general in the graph-based languages similar to G-Log, this will allow to trace the evolution of documents and Web pages by keeping trace of the history of their DTD's or schemata.

Our approach may remind previous works on Graphlog [9] and UnQL [5]; differences between G-Log and them are mainly related to expressive power: for instance, G-Log allows to express cyclic information and queries, and achieves its high expressive power by allowing a fully user-controlled non-determinism. Addition of entities, other than the addition of relations allowed in Graphlog, is admitted in G-Log which, in its full form, is Turing complete [29].

This issue is further dealt with in Section 7, where it becomes clear that our results can be extended to these languages as well, and in general to any graphical language whose main aim is to query and transform a graph-oriented data model by using graph-based rules.

The paper is organized as follows. Section 2 introduces the language G-Log. Section 3 explains the (concrete) operational semantics of the language, showing the three-level semantics which introduces flexibility. In Section 4 some results for the semantics proposed are given in detail; here different types of rules are analyzed and the differences between the three semantics are highlighted. In Section 5 the notion of abstract graphs (corresponding to schemata) is introduced, and the concepts of abstract interpretation are applied; in some cases query applicability can be tested directly on schemata; this means that they represent instances correctly. Moreover, this section also shows how schemata can be derived by abstraction (in $n \log n$ time) from instance sets, thus allowing to deduce a common scheme or DTD from a set of documents. Section 6 introduces a logical and model theoretic view of G-Log graphs and the relationships with the concrete operational semantics are analyzed. In Section 7 we present a comparison with similar previous approaches and we set the lines for future works. Finally, conclusions are drawn in Section 8.

2   The Language G-Log

2.1   An Informal Presentation

In this section we introduce some intuitive examples of queries in the language G-Log, in order to appreciate its expressive power and to emphasize some ambiguities we are going to tackle later on.

Consider the graph depicted in Fig. 1 (a). It represents the query 'collect all the people that are fathers of someone'. Intuitively, the boldface part of the graph (also called the 'green part') is what you try to get from the DB, while you match the rest of the graph (also called the 'red part') with a graph representing the DB instance.

The query (b) of Fig. 1 can be read as 'collect all the workers having (at least) one son that works in some town'.

Also negative requirements can be introduced in a query by means of dashed edges and nodes. This is depicted by query (c) of Fig. 1 whose meaning is 'collect all the workers having (at least) one son that works in a town different from that where his father works'.

The translation of queries $(a), (b), (c)$ into logical formulas is also illustrated in Fig. 1 (with abbreviations for predicate symbols). As observed in [29], G-Log offers the expressive power of logic, the modeling power of object-oriented DBs, and the representation power of graphs.

However, the modeling power of G-Log is heavily constrained by some arguable choices in its semantics [29]. Consider, for instance, query (d) of Fig. 2: it can be intuitively interpreted in three different ways:

- collect the people having two children, not necessarily distinct;

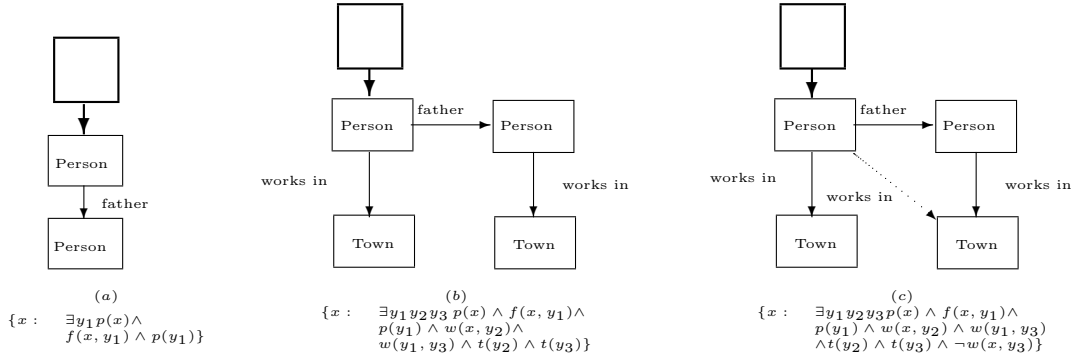- collect the people having exactly two (distinct) children;

Person

Person

father

Person

(a)
$\{x : \quad \exists y_1\, p(x) \wedge$
$f(x, y_1) \wedge p(y_1)\}$

Person — father → Person

works in

Town

works in

Town

(b)
$\{x : \quad \exists y_1 y_2 y_3\, p(x) \wedge f(x, y_1) \wedge$
$p(y_1) \wedge w(x, y_2) \wedge$
$w(y_1, y_3) \wedge t(y_2) \wedge t(y_3)\}$

Person — father → Person

works in

works in

works in

Town

Town

(c)
$\{x : \quad \exists y_1 y_2 y_3\, p(x) \wedge f(x, y_1) \wedge$
$p(y_1) \wedge w(x, y_2) \wedge w(y_1, y_3)$
$\wedge t(y_2) \wedge t(y_3) \wedge \neg w(x, y_3)\}$

Figure 1: Sample queries

- collect the people having at least two (distinct) children.

The semantics of G-Log as given in [29] uniquely selects the first option. As a consequence, queries (a) and (d) become equivalent, so there is no way to express 'collect the people that have more than one child' without making use of negative information (negated equality edges in G-Log [29]).

An even deeper problem arises when considering query (e): in G-Log it has exactly the same meaning as query (b). In other words, it is not possible to express a query like 'collect the people that work in the same town as (at least) one of their children' in a natural fashion. Actually, such a query can be expressed in G-Log, but not in a straightforward way. Of course, these problems are further emphasized when combined with negation.

In order to address this kind of ambiguities, in the following sections we revisit the semantics of G-Log taking advantage of the use of the well-known concept of bisimulation. Furthermore, we apply the abstract interpretation approach to the operational semantics defined in this way, in order to clarify the relationship between concrete (instances) and abstract (schemata) data representations.

2.2   Syntax of G-Log

In this section we introduce the basic aspects of the syntax of the G-Log language. Definitions are based on the concept of directed labeled graph, and, differently from [29, 31], rules, programs, and queries are defined independently of the context in which they are used. This simplifies the notation and allows the study of algebraic properties of programs. However, the semantics (cf. Section 3) will be given in such a way that the practical use is coherent with that of [29, 31].
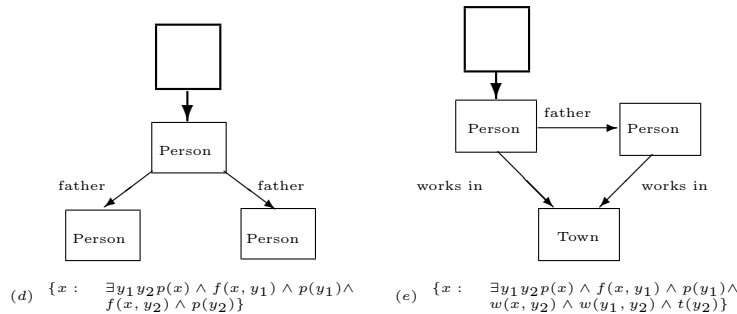
Person

father   father

Person   Person

(d)
$\{x : \quad \exists y_1 y_2\, p(x) \wedge f(x, y_1) \wedge p(y_1) \wedge$
$f(x, y_2) \wedge p(y_2)\}$

Person — father → Person

works in   works in

Town

(e)
$\{x : \quad \exists y_1 y_2\, p(x) \wedge f(x, y_1) \wedge p(y_1) \wedge$
$w(x, y_2) \wedge w(y_1, y_2) \wedge t(y_2)\}$

Figure 2: Problematical queries

4

Definition 2.1 *A* G-Log graph *is a directed labeled graph* $\langle N, E, \ell \rangle$, *where N is a (finite) set of nodes, E is a set of labeled edges of the form* $\langle m, label, n \rangle$, *where $m, n \in N$ and label is a pair of $\mathcal{C} \times (\mathcal{L} \cup \{\bot\})$, while* $\ell : N \longrightarrow (\mathcal{T} \cup \{\bot\}) \times \mathcal{C} \times (\mathcal{L} \cup \{\bot\}) \times (\mathcal{S} \cup \{\bot\})$. $\bot$ *means 'undefined', and:*

- $\mathcal{T} = \{$ *entity, slot* $\}$ *is a set of* types *for nodes;*
- $\mathcal{C} = \{$ *red, green, black* $\}$ *is a set of* colors;
- $\mathcal{L}$ *is a set of* labels *to be used as entity, slot, and relation names;*
- $\mathcal{S}$ *is a set of* strings *to be used as concrete values.*

ℓ is the composition of four single-valued functions $\ell_\mathcal{T}, \ell_\mathcal{C}, \ell_\mathcal{L}, \ell_\mathcal{S}$. When the context is clear, if $e = \langle m, \langle c, k \rangle, n \rangle$, with abuse of notation we say that $\ell_\mathcal{C}(e) = c$ and $\ell_\mathcal{L}(e) = k$. Moreover, we require that:

- $(\forall x \in N)(\ell_\mathcal{T}(x) \neq slot \rightarrow \ell_\mathcal{S}(x) = \bot)$ *(i.e., values are associated to* slot *nodes only),*
- $(\forall \langle m, label, n \rangle \in E)(\ell_\mathcal{T}(m) \neq slot)$ *(i.e.,* slot *nodes are leaves).*

Observe that two nodes may be connected by more than one edge, provided that edge labels be different.

Red (RS) and *black* edges and nodes are graphically represented by thin lines (this does not originate confusion, since there can not be red and black nodes and edges in the same graph), while *green* (GS) by thick lines. Red and green nodes are used together in queries.

Colors red and green are chosen to remind traffic lights. Red edges and nodes add constraints to a query (= stop!), while green nodes and edges correspond to the data we wish to derive (= walk!).

Result nodes play a particular rôle in queries: they have the intuitive meaning of requiring the collection of all objects fulfilling a particular property. Moreover, result nodes can occur in (instances of) web-like databases to simulate web pages connecting links. In this paper, if an entity node is labeled by *result*[3] it will be simply represented by small squares, and its outcoming edges implicitly labeled by *'connects'*. In general, an *entity* (*slot*) node n will be represented by a rectangle (oval) containing the label $\ell_\mathcal{L}(n)$.

As an instance, consider the graph (d) of Figure 2. Let 1 be the topmost node, 2 the center node, 3 the leftmost, and 4 the rightmost node. Then

$$
\begin{array}{rlll}
G & = & \langle \ N & = \ \{ \ 1, 2, 3, 4\}, \\
& & E & = \ \{ \ \langle 1, \langle GS, connects \rangle, 2 \rangle, \\
& & & \quad \langle 2, \langle RS, father \rangle, 3 \rangle, \\
& & & \quad \langle 2, \langle GS, father \rangle, 4 \rangle \}, \\
& & \ell & = \ \{ \ 1 \mapsto \langle entity, GS, result, \bot \rangle, \\
& & & \quad 2 \mapsto \langle entity, RS, Person, \bot \rangle, \\
& & & \quad 3 \mapsto \langle entity, RS, Person, \bot \rangle, \\
& & & \quad 4 \mapsto \langle entity, RS, Person, \bot \rangle \} \quad \rangle
\end{array}
$$

Definition 2.2 *Let $G = \langle N, E, \ell \rangle$ and $G' = \langle N', E', \ell' \rangle$ be G-Log graphs. We say that G is a* labeled subgraph *of G', denoted $G \sqsubseteq G'$, if $N \subseteq N'$, $E \subseteq E'$, and $\ell = \ell'|_N$ (i.e., for all $x \in N$ it holds that $\ell(x) = \ell'(x)$).*

With ε we denote the (empty) G-Log graph $\langle \emptyset, \emptyset, \emptyset \rangle$. It is immediate to see that given a G-Log graph G, then

$$\langle \{G' \text{ is a G-Log graph} : G' \sqsubseteq G\}, \sqsubseteq \rangle$$

is a *complete lattice*, where $\top \equiv G$, $\bot \equiv \varepsilon$. Moreover, given two G-Log graphs $G_1 = \langle N_1, E_1, \ell_1 \rangle \sqsubseteq G$ and $G_2 = \langle N_2, E_2, \ell_2 \rangle \sqsubseteq G$, where $\ell_1|_{N_2} = \ell_2|_{N_1}$, their l.u.b. and g.l.b. can be computed as[4]

$$
\begin{array}{rcl}
G_1 \sqcup G_2 & = & \langle N_1 \cup N_2, E_1 \cup E_2, \ell_1 \cup \ell_2 \rangle \\
G_1 \sqcap G_2 & = & \langle N_1 \cap N_2, E_1 \cap E_2, \ell_1 \cap \ell_2 \rangle
\end{array}
$$

---

[3][11] uses *entry point* nodes for this purpose.

[4]As a side remark, notice that, if G is the (complete) graph $\langle N, N \times \{\bot\} \times N, \ell \rangle$ and $n = |N|$, then the lattice contains: $\sum_{i=0}^{n} \binom{n}{i} 2^{i^2} = O(n2^{n^2})$ subgraphs. If G is not of this form, it is difficult to find the exact number; however, if $|E| = O(|N|^2)$, then the upper bound remains the same as the complete case.

Definition 2.3 *Given a G-Log graph $G = \langle N, E, \ell \rangle$, and a set C of colors, $C \subseteq \mathcal{C}$, consider the sets $N' = N \cap \ell_\mathcal{C}^{-1}(C)$ and $E' = \{\langle m, \langle c, k \rangle, n \rangle \in E : c \in C\}$. The restriction of G to the colors in C, denoted $G|_C$ is defined as $G|_C = \langle N', E', \ell|_{N'} \rangle$.*

Observe that $G|_C$ is not necessarily a graph, since, for instance, it may contain only edges and no nodes.

We introduce the notions of concrete graph, rule, and program. Through them, we aim at characterizing the instances of a semi-structured database like a WWW site.

Definition 2.4 *A G-Log concrete graph is a G-Log graph such that:*

1. $(\forall x \in N \cup E)(\ell_\mathcal{C}(x) = black)$, and

2. $(\forall n \in N)(\ell_\mathcal{T}(n) = slot \rightarrow \ell_\mathcal{S}(n) \neq \bot)$.

With $\mathcal{G}^c$ we denote the set of G-Log concrete graphs.

Definition 2.5 *A G-Log rule $R = \langle N, E, \ell \rangle$ is a G-Log graph such that:*

1. $(\forall x \in N \cup E)(\ell_\mathcal{C}(x) \neq black)$,

2. $R|_{\{GS\}} \neq \emptyset$, and

3. $(\forall n, n' \in N)((\ell_\mathcal{C}(n) = GS \wedge \ell_\mathcal{C}(n') = RS) \rightarrow (\ell_\mathcal{L}(n) \neq \ell_\mathcal{L}(n') \vee \ell_\mathcal{T}(n) \neq \ell_\mathcal{T}(n')))$.

The third condition is introduced to avoid the possibility of infinite generation of nodes (cf. Remark 4.11).[5] Notice that it can be the case that $R|_{\{RS\}} = \emptyset$. This corresponds to an *unconditional* query or update. In general, we can split the notion of rule in two concepts: *query* and *update*. Basically, queries are expected to extract information from a graph (i.e., no existing *class* of objects is modified), whereas updates are expected to build up new instances of the graph (i.e., classes and relationships can be modified).

Remark 2.6 *Observe that no restriction is imposed on the structure of the graphs. Graphs can contain cycles of any kind.*

Definition 2.7 *Given a G-Log rule $R = \langle N, E, \ell \rangle$ and a graph $G = \langle N', E', \ell' \rangle$, The rule R is a query with respect to G if the following conditions hold:*

1. $(\forall n \in N)(\ell_\mathcal{C}(n) = GS \rightarrow (\forall n' \in N')(\ell_\mathcal{L}(n) \neq \ell_\mathcal{L}(n') \vee \ell_\mathcal{T}(n) \neq \ell_\mathcal{T}(n')))$,

2. $(\forall e \in E)(\ell_\mathcal{C}(e) = GS \rightarrow (\forall e' \in E')(\ell_\mathcal{L}(e) \neq \ell_\mathcal{L}(e')))$.

The rule R is an update *with respect to G if it is not a query.*

As a matter of fact, this formal notion does not correspond exactly to the common usage of the word "query": we further distinguish two kinds of queries: *generative* queries retrieve some objects and relationships and, based on them, construct new concepts. For instance, from the notion of parent, a generative query (the transitive closure) can construct the notion of ancestor. *Pure retrieval* queries associate links to a number of objects enjoining a common property. The last notion captures the common intuition of *query*. The previous one is more related to the usual notion of *view*.

A computation can be seen as a sequence of applications of a sequence of rules to a graph. This leads to the following definition of program.

Definition 2.8 *A G-Log program is a finite list of sets of G-Log rules.*

Examples of G-Log programs can be found in Section 3.2.

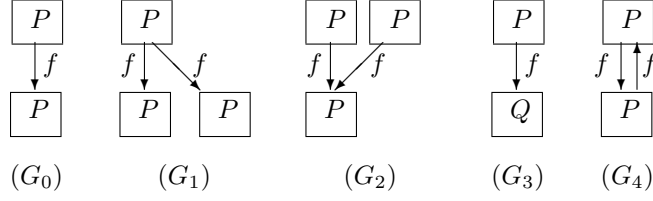[5]This corresponds to the well-known problem of OID generation in logical object oriented DB's [1].

Figure 3: Bisimilar and not bisimilar graphs

3   Operational Semantics of G-Log

We present a three-level semantics, based on the concept of bisimulation. In Section 7.2 we compare this approach with that based on the the concept of *embedding* of a graph used in [29].[6]

First, let us remind the well-known concept of bisimulation [30, 27] adapted to our setting:

Definition 3.1 *Given two G-Log graphs $G_0 = \langle N_0, E_0, \ell^0 \rangle$ and $G_1 = \langle N_1, E_1, \ell^1 \rangle$, a relation $b \subseteq N_0 \times N_1$ is said to be a* bisimulation *between G_0 and G_1 if and only if:*

1. *for $i = 0, 1$, $(\forall n_i \in N_i)(\exists n_{1-i} \in N_{1-i}) n_0 \, b \, n_1$,*

2. *$(\forall n_0 \in N_0)(\forall n_1 \in N_1)(n_0 \, b \, n_1 \rightarrow \ell_{\mathcal{T}}^0(n_0) = \ell_{\mathcal{T}}^1(n_1) \wedge \ell_{\mathcal{L}}^0(n_0) = \ell_{\mathcal{L}}^1(n_1) \wedge \ell_{\mathcal{S}}^0(n_0) \doteq \ell_{\mathcal{S}}^1(n_1))$ (where $\doteq$ means that if both labels are defined—i.e., different from $\bot$—they must be equal), and*

3. *for $i = 0, 1$, $(\forall n \in N_i)$, let $M_i(n) =_{def} \{\langle m, label \rangle : \langle n, \langle color, label \rangle, m \rangle \in E_i\}$.*

   Then, $(\forall n_0 \in N_0)(\forall n_1 \in N_1)$ such that $n_0 \, b \, n_1$, for $i = 0, 1$ it holds that

$$(\forall \langle m_i, \ell_i \rangle \in M_i(n_i))(\exists \langle m_{1-i}, \ell_{1-i} \rangle \in M_{1-i}(n_{1-i}))(m_0 \, b \, m_1 \wedge \ell_i = \ell_{1-i}).$$

We write $G_0 \overset{b}{\sim} G_1$ ($G_0 \overset{b}{\not\sim} G_1$) if b is (not) a bisimulation between G_0 and G_1. We write $G_0 \sim G_1$ ($G_0 \not\sim G_1$) if there is (not) a bisimulation between G_0 and G_1: in this case we also say that G_0 is bisimilar to G_1.

A brief explanation of the conditions above may be useful. Conditon 1 is obvious: no node in the two graphs can be left out of the relation b. Condition 2 states that two nodes belonging to relation b have same type and same label, exactly. Moreover, if they are just slots, then either one of their values is undefined, or they have also the same value. Finally, condition 3 deals with edge correspondences. If two nodes n_0, n_1 are in relation b, then every edge having n_0 as endpoint should find as a counterpart a corresponding edge with n_1 as endpoint.

As an example, consider the graphs in Fig. 3:

- $G_0 \sim G_1 \sim G_2$, as well as the reflexive, symmetric and transitive closure of this fact, since $\sim$ is an equivalence relation (cf. Lemma 3.3);

- $G_0 \not\sim G_3$ since it is impossible to 'bind' a node labeled by P with one labeled by Q. Thus, condition (2) cannot be verified;

- $G_0 \not\sim G_4$ since it is impossible to verify condition (3).

---

[6]In this paper we do not face the problem of negation (dashed nodes and edges). A line for future work is drawn in Section 7.3.

Notice that colors (represented by the $\ell_{\mathcal{C}}$ function) are not taken into account in the bisimulation definition, while the *value* fields of the label are considered only when they are defined. The last feature will allow to apply bisimulations between *schemata* and *instances* (see Section 5).

The bisimulation relation can be further refined by introducing additional conditions:

Definition 3.2 *Given two G-Log graphs $G_0 = \langle N_0, E_0, \ell_0 \rangle$ and $G_1 = \langle N_1, E_1, \ell_1 \rangle$, and $b \subseteq N_0 \times N_1$ we say that*

- *b is a* directional bisimulation*, denoted by $G_0 \overset{b}{\approx} G_1$, if $G_0 \overset{b}{\sim} G_1$ and b is a function from N_0 to N_1. We say that $G_0 \approx G_1$ if there is a b such that $G_0 \overset{b}{\approx} G_1$.*

- *b is a* bidirectional bisimulation*, denoted by $G_0 \overset{b}{\equiv} G_1$, if $G_0 \overset{b}{\sim} G_1$ and b is a bijective function from N_0 to N_1. We say that $G_0 \equiv G_1$ if there is a b such that $G_0 \overset{b}{\equiv} G_1$.*

Again in Fig. 3, we have that:

- $G_1 \approx G_0$, $G_2 \approx G_0$, while the converse is not true.

- $G_i \equiv G_i$ for $i = 0, 1, 2, 3, 4$, while $G_i \not\equiv G_j$ for $i \neq j$.

The three relations defined above will be used to define the semantics that we are going to study in the rest of the paper:

$\sim$ is used to build a semantics based on bisimulation;

$\approx$ is used to build a semantics based on the concept of bisimulation that is also a function;

$\equiv$ is used to build a semantics based on *graph isomorphism* (*injective embeddings* [29] or, equivalently, *bisimulations that are bijections*).

Some basic properties of these relations are emphasized in the following lemma.

Lemma 3.3   *1. $\sim$, $\approx$, and $\equiv$, are reflexive and transitive relations;*

2. Both $\sim$ and $\equiv$ are symmetric relations and thus, equivalence relations;

3. $\approx$ is a preorder (and not an ordering).

4. $G \equiv G'$ if and only if $G \approx G'$ and $G' \approx G$.

Proof. Follows immediately from the definition for 1 and 2. For 3, consider the graphs:
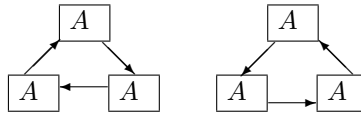
$$G_1 \qquad\qquad G_2$$

It holds that $G_1 \approx G_2$ and vice versa. However, the two graphs are not the same graph. For proving statement (4), it is sufficient to observe that, by the requirement (1) of the definition of bisimulation (Definition 3.1), the two functions ensuring that $G \approx G'$ and $G' \approx G$ are onto. This means that $|N| = |N'|$ and, thus, both functions are bijections.   □

Remark 3.4 *The operational semantics of the original definition of the language G-Log [29] is based on a different notion of matching of graphs: the so-called* embedding. *Relationships between our proposals and the embedding are explained in Section 7.*

3.1 Semantics of Rules

The first two notions that we define are the applicability of a rule and the satisfiability of a graph, given a rule.

Definition 3.5 *Let G be a concrete graph and R a rule. For ξ in $\{\sim, \vec{\approx}, \equiv\}$, R is ξ-applicable in G if $(\exists G_1 \sqsubseteq G)(R_{\{RS\}}\xi G_1)$.*

Definition 3.6 *Let G be a concrete graph and R a rule. For ξ in $\{\sim, \vec{\approx}, \equiv\}$, G ξ-satisfies R ($G \models_\xi R$) if for all $G_1 \sqsubseteq G$ such that there is b_1 with $R_{\{RS\}} \overset{b_1}{\xi} G_1$, there exist $G_2 \sqsubseteq G$ and $b_2 \supseteq b_1$ that satisfy the following conditions:*

(i) $G_1 \sqsubseteq G_2$;

(ii) $R_{\{RS,GS\}} \overset{b_2}{\xi} G_2$;

(iii) $(\forall G_3 \sqsubseteq G_2)(G_1 \sqsubseteq G_3 \wedge R_{\{RS\}}\xi G_3 \rightarrow G_3 = G_1)$.

Intuitively, G satisfies R if for any subgraph G_1 of G matching (with respect to ξ) the red part of the rule (i.e. the pre-condition), there is a way to 'complete' G_1 into a graph $G_2 \sqsubseteq G$ such that the whole rule R matches G_2. Condition (*iii*) is necessary to avoid the possibility of using other parts of G, matching with $R_{\{RS\}}$ independently, to extend G_1.

Example 3.7 *For instance, consider the graphs below.*
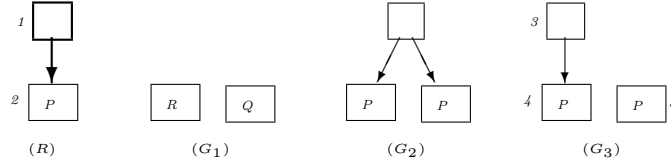
Rule R is not ξ-applicable to G_1 (so G_1 ξ-satisfies R trivially) and G_2 ξ-satisfies R, for ξ in $\{\sim, \vec{\approx}, \equiv\}$. Observe the necessity of condition (*iii*), in the case of $\sim$, to ensure that G_3 does not $\sim$-satisfy R. As a matter of fact, with $b_1 = \{\langle 2, 5\rangle\}$ $R_{RS} \overset{b_1}{\sim} G'$ where G' consists of the unique node 5. However, with $b_2 = \{\langle 2,4\rangle, \langle 2,5\rangle, \langle 1,3\rangle\}$ it holds that $R \overset{b_2}{\approx} G_3$. To achieve this bisimulation, however, b_1 has been 'unnaturally extended'. This is avoided by the last condition of Definition 3.6.

Problems like those seen in Example 3.7 come from the fact that functions can be extended to relations. This is not possible for $\vec{\approx}$, and $\equiv$ because their extensions must be functions, by definition. Thus, Definition 3.6 can be significantly simplified for $\vec{\approx}$ and $\equiv$:

Lemma 3.8 *Let G be a concrete graph and R a rule. For ξ in $\{\vec{\approx}, \equiv\}$, G ξ-satisfies R ($G \models_\xi R$) if and only if for all $G_1 \sqsubseteq G$ and for all b_1 such that $R_{\{RS\}} \overset{b_1}{\xi} G_1$ $\exists G_2 \sqsubseteq G$ and $\exists b_2 \supseteq b_1$:*

1. $G_1 \sqsubseteq G_2$ and

2. $R_{\{RS,GS\}} \overset{b_2}{\xi} G_2$.

$\square$

The notion of applicability is a pre-condition for an effective application of a rule to a concrete graph, whose precise semantics is given below:
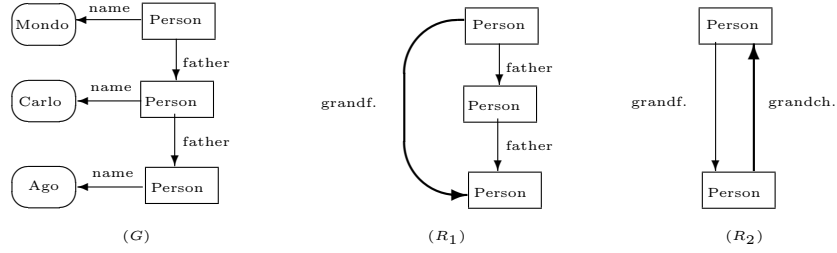
Figure 4: Non commutativity of rule applications
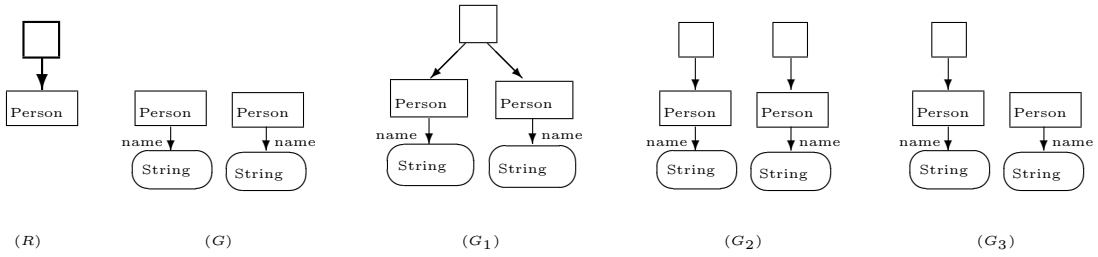
Figure 5: Application of a rule R to a graph G

Definition 3.9 *Let R be a rule. Its* operational semantics $[\![\,R\,]\!]_\xi \subseteq \mathcal{G}^c \times \mathcal{G}^c$ *is defined as follows: for ξ in $\{\sim, \vec{\sim}, \equiv\}$, $\langle G, G' \rangle \in [\![\,R\,]\!]_\xi$ if and only if :*

1. *$G \sqsubseteq G'$, $G' \models_\xi R$, and*

2. *G' is minimal with respect to property (1), namely there is no graph G'' such that $G \sqsubseteq G''$, $G'' \sqsubset G'$, and $G'' \models_\xi R$.*

Intuitively, a rule, if applicable, extends G in such a way that G satisfies R. Moreover, it is required that the extension be minimal. If R is not applicable in G, then G satisfies R trivially and $\langle G, G \rangle \in [\![\,R\,]\!]_\xi$.

Example 3.10 *Consider the graph G and the rules R_1 and R_2 of Fig. 4. The application of Rule R_2 leaves the graph G unchanged. The application of Rule R_1 uniquely adds the grandfather relation. The application of Rule R_2 after that of Rule R_1 furtherly adds the grandchild relation. Thus, rule application is not commutative.*

Example 3.11 *Consider graphs of Fig. 5. It holds that $\langle G, G_1 \rangle$ and $\langle G, G_2 \rangle$ belong to $[\![\,R\,]\!]_\sim$. $\langle G, G_3 \rangle \notin [\![\,R\,]\!]_\sim$ since $G_3 \not\models_\sim R$: this is due to condition (iii) in the Definition 3.6. Notice that $G_1 \sim G_2 \sim G_3$.*

Definition 3.12 *If G is a G-Log graph, then for ξ in $\{\sim, \vec{\sim}, \equiv\}$, $[\![\,\cdot\,]\!]_\xi(G)$ is a function from the set of the rules to the powerset of the set of G-Log graphs, defined as follows:*

$$[\![\,R\,]\!]_\xi(G) =_{def} \{G' : \langle G, G' \rangle \in [\![\,R\,]\!]_\xi\}$$

Rules can be combined to build programs according to Definition 2.8:

Definition 3.13 *Let $S = \{R_1, \ldots, R_n\}$ be a set of rules. Then, for ξ in $\{\sim, \vec{\sim}, \equiv\}$, $\langle G, G' \rangle \in [\![\,S\,]\!]_\xi$ if*

1. *$G \sqsubseteq G'$, $G' \models_\xi R_i$, for $i = 1, \ldots, n$, and*

2. *G' is minimal with respect to property (1).*

10

Let P be a program $\langle S_1, \ldots, S_n \rangle$. For ξ in $\{\sim, \vec{\sim}, \equiv\}$, $\langle G_0, G_n \rangle \in [\![\, P \,]\!]_\xi$ if and only if there are $G_1, \ldots, G_{n-1}$ such that $\langle G_i, G_{i+1} \rangle \in [\![\, S_{i+1} \,]\!]_\xi$, for $i = 0, \ldots, n-1$.

The following notion is useful in practical querying:

Definition 3.14 *Let R be a rule and G be a concrete graph such that $G \models R$. For ξ in $\{\sim, \vec{\sim}, \equiv\}$, the ξ-view of G using R, denoted by $G|_R$ is the union of all the graphs $G' \sqsubseteq G$ such that $R\, \xi\, G'$. The unfolded ξ-view of G using R ($G^{\uplus}|_R$) is the disjoint union of all the G'.*
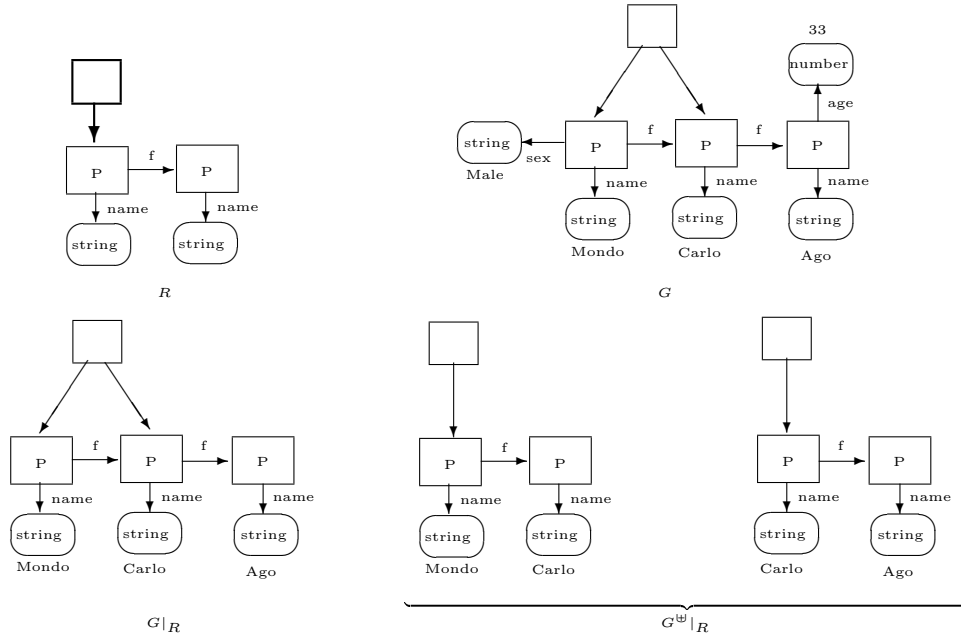
Figure 6: Rules, concrete graphs, and views

Figure 6 shows a concrete graph G satisfying a rule R, its view using R (in this case there is no difference adopting different semantics), and its unfolded view.

3.2 Programming in G-Log

Let us show now how to build up a database using the G-Log language and then, how to query it.

Suppose we want to create a new database which contains informations about *students*, the *courses* they attend and *teachers*. We use an unconditional rule, i.e. a rule without red part, since we wish to build up the database and not to modify an existing one.

For example, the graph G_0 depicted in Fig. 7 is a *G–Log generative unconditional query* (its color is only green solid); it creates a simple database with three entities: John is a student who attends the Database course and Dr. Smith is the teacher of the same course. If we apply G_0 to the initial empty concrete graph, we build up the G-Log concrete graph G_1 of Fig. 7 which ξ-*satisfies* G_0. Given G_1 we can either querying it or add more information to it.

Now, suppose we apply the rule R of Fig. 8 *'if a person teaches a course and a student attends that course then the person is a student's teacher'* to G_1. R is ξ-*applicable* to G_1 for ξ in $\{\sim, \vec{\sim}, \equiv\}$. As a matter of fact, for each ξ there is a ξ-*relation* between the red solid part of R and a subgraph of G_1; therefore, for each ξ, there is a way to expand the concrete graph in order to obtain a new graph G'_1 of Fig. 8 that matches the whole rule. In this particular case, the extension is the same.
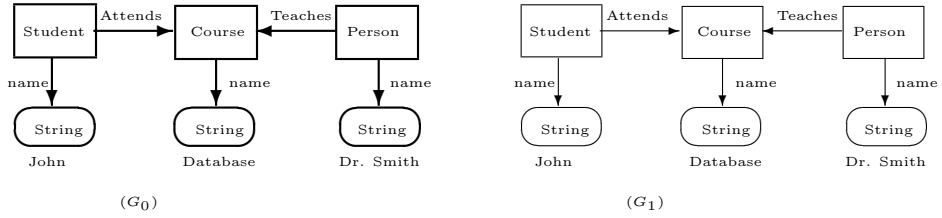
Figure 7: An update and a concrete graph
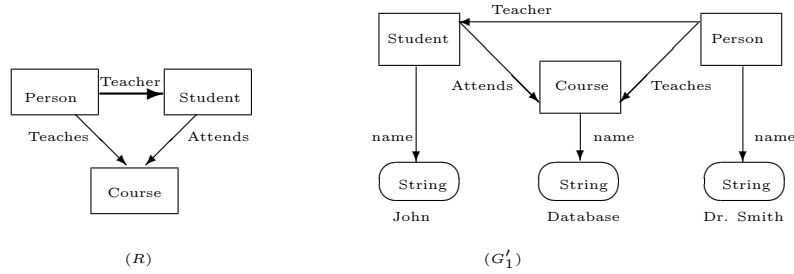
Figure 8: A G-Log rule and a concrete graph

This way, using G-Log rules, we can query a database to obtain information and complete its concrete graph adding new nodes or edges.

Moreover, G-Log allows the expression of complex queries by means of *programs* which are sequences of rules. Sometimes it is worthwhile to have the possibility of expressing *transitive properties*: in G-Log a set of two rules is enough.

For instance, the program of Fig. 9 expresses the following transitive property: 'if two students attend the same course, they are schoolfellows. And, if a student x is a schoolfellow of a student y and y is a schoolfellow of a student z then x is z's schoolfellow'.
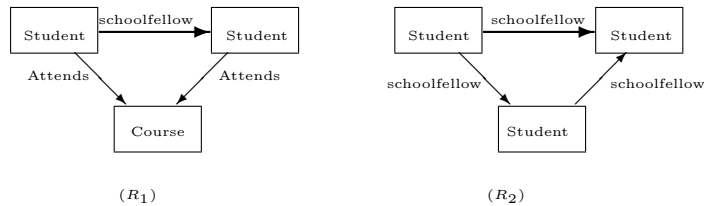
Figure 9: A G–Log program
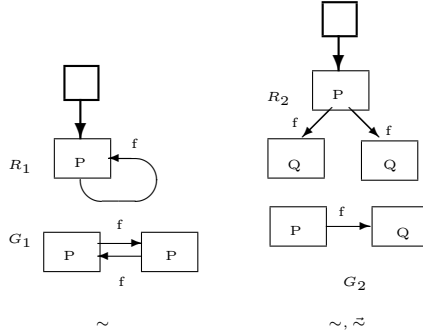
Figure 10: Applicability differences

4 Basic Semantic Results

In this subsection we analyze the main results concerning the proposed parametric semantics, in order to point out G-Log rules of a form ensuring desirable properties, first of all program determinism.

4.1 Applicability

Proposition 4.1 *For each G-Log rule R,*

1. *if R is $\equiv$-applicable, then it is $\vec{\sim}$-applicable;*

2. *if R is $\vec{\sim}$-applicable, then it is $\sim$-applicable.*

Proof. Immediate, by definition. □

Relations $\sim$, $\vec{\sim}$, and $\equiv$ have different expressivity and thus they can be compared to form an ordering (cf., also, Section 7). The ordering is strict, as follows from Figure 10: R_1 is ξ-applicable to G_1 only when ξ is $\sim$. R_2 is ξ-applicable to G_2 for $\sim$ and $\vec{\sim}$, but not for $\equiv$.

4.2 Satisfiability

The situation is a bit more intricate as far as the concept of *satisfiability* is concerned:

Proposition 4.2 *There are rules R such that the sets $\{G : G \models_\sim R\}$, $\{G : G \models_{\vec{\sim}} R\}$, and $\{G : G \models_\equiv R\}$ are pairwise distinct.*

Proof. Consider rule R_1 and the graph G_1 below. G_1 does not satisfy R for $\sim$ and $\vec{\sim}$. However, since R_1 is not $\equiv$-applicable in G_1, trivially $G_1 \models_\equiv R_1$.

On the other hand, rule R_1 is $\equiv$-applicable (hence, $\sim$- and $\vec{\sim}$-applicable, thanks to Proposition 4.1) to graph G_2. However, it only holds that $G_2 \models_\sim R_1$.
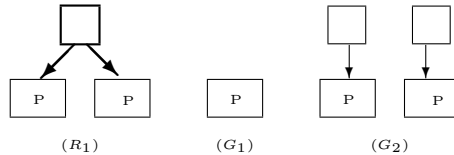
Now we prove that for some R, $\{G : G \models_{\vec{\sim}} R\}$ may contain elements that are not in the other two sets. Consider rule R_2 and graph G_3 below:

(R_2) (G_3)
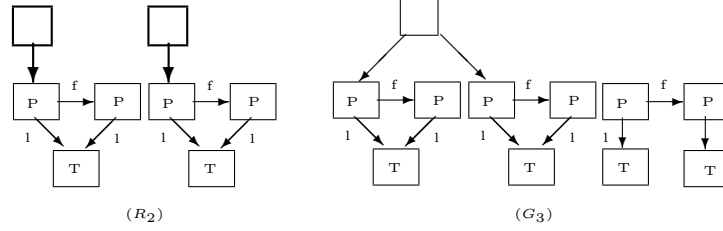
R_2 is $\equiv$-applicable to G_3 but it is not $\equiv$-satisfied. Similarly, R_2 is $\sim$-applicable to G_3 but it is not $\sim$-satisfied, due to the rightmost subgraph. Instead, G_3 $\overset{\rightarrow}{\sim}$-satisfies R_2.   □

Thus, none of the three sets is included in the other.

Proposition 4.3 *Let G be a G-Log graph and R a G-Log rule. For ξ in $\{\sim, \overset{\rightarrow}{\sim}, \equiv\}$, if R is ξ-applicable to G, then there is a G' such that 1) $G \sqsubseteq G'$, and 2) $G' \models_\xi R$, and G' is minimal with respect to the properties (1) and (2).*

Proof. Consider a rule R ξ-applicable to G. The existence of a G' fulfilling (1) and (2) is clearly ensured: for each $G_i \sqsubseteq G$ such that $G_i \xi R_{RS}$ (existing by hypothesis), consider G'_i obtained by augmenting G_i with new nodes and edges 'copying' R_{GS}. Consider $G' = G \cup \bigcup_i G'_i$. The assumption (3) of the definition of rule (Definition 2.5) ensures that the process cannot enter into loop, thus ensuring the finiteness of the graph G'.

To get one (among the various possible) minimal graph it is sufficient to remove some of the new edges and nodes (possibly collapsing them) while the satisfiability property still holds.   □

In other words, if R is ξ-applicable to G, then $[\![R]\!]_\xi(G)$ is not empty.

Corollary 4.4 *For $\xi \in \{\sim, \overset{\rightarrow}{\sim}, \equiv\}$, for any rule R and graph G, $[\![R]\!]_\xi(G) \neq \emptyset$.*

Proof. If R is not $\equiv$-applicable in G, then $[\![R]\!]_\xi(G) = \langle G, G \rangle$ by definition. Otherwise, the result follows from Proposition 4.3.   □

4.3   Simple Edge-Adding Rules

We analyze now the effect of some simple rules, *edge-adding rules*, and prove the determinism of their semantics. First of all we point out an ambiguity hidden in the graphical language.

Consider the following rule in which the green part is composed only by one edge connecting two nodes with the same label:
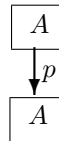
Its intuitive meaning is: any time you have two entity nodes labeled by A (necessarily distinct if we are using the $\equiv$ semantics) add an edge labeled p between them, unless one edge of this form is already present. The meaning is exactly the same as that of the following rule:

14
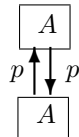
The first rule, in a sense, hides a cycle of green edges. Notice that this happens even if the two rules are not bisimilar: the existence of a bisimulation between two rules is not required for the two rules to have the same expressive power.

Table 1 shows the differences of the operational semantics of rules R_1, R_2, R_3 admitting cycles of green edges involving equivalent nodes on simple concrete graphs G_1, G_2, G_3.

We observe that:

1. The three semantics are all equivalent with respect to rule R_1;

2. For the other rules, there are always differences;

3. Rules R_2 and R_3 may be non-$\equiv$-applicable for graphs with too few nodes. This is due to the constraint on cardinality required by the graph isomorphism relation $\equiv$;

4. The semantics based on bisimulation ($\sim$) does not distinguish the three rules R_1, R_2, and R_3. This is due to the possibility given by bisimulation (a relation in general—not necessarily a function) to bind one node with a family of nodes.

5. The semantics based on $\overset{\rightarrow}{\sim}$ can not distinguish rules R_2 and R_3;

6. The semantics based on $\equiv$ distinguishes all the rules.

7. In all the examples, the application of rules is a function.

Actually, the same conclusions can be drawn whenever all the nodes belonging to a cycle are roots of ξ-equivalent and disjoint G-Log graphs. R_1 and R_2 are:

- $\sim$-equivalent if $R_1 \sim R_2$,

- $\equiv$-equivalent if $R_1 \equiv R_2$, and

- $\overset{\rightarrow}{\sim}$-equivalent if for all I, $R_1 \overset{\rightarrow}{\sim} I$ if and only if $R_2 \overset{\rightarrow}{\sim} I$.

Let us study some more general properties of rule application for simple rules. We begin with the simple cases in which R_{GS} consists only of edges.

Lemma 4.5 *For each G-Log rule R, if R_{GS} consists only of one edge and no nodes, then $[\![R]\!]_\xi$ is a function, for ξ in $\{\sim, \overset{\rightarrow}{\sim}, \equiv\}$.*

Proof. We need to prove that for each G-Log graph G, there is exactly one G' such that $\langle G, G' \rangle \in [\![R]\!]_\xi$.

If R is not ξ-applicable, then the result holds by definition choosing G' as G.

Assume R is ξ-applicable in G, for any ξ in $\{\sim, \overset{\rightarrow}{\sim}, \equiv\}$. By hypothesis, R can have only one of the following two forms:

$$(1) \qquad\qquad (2)$$
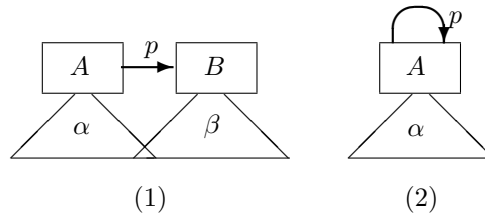
where the nodes labeled by A and B in (1) are distinct nodes (but not necessarily label A is different from label B) or in (2) they are the same node. We prove first the fact when ξ is $\sim$.

Let R be of the form (1). Its semantics is exactly that of introducing *all possible* edges labeled by p connecting subgraphs bisimilar to A, α and B, β of G, unless they are already present. This kind of extension of G is clearly unique.
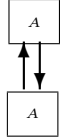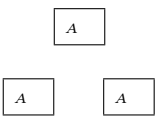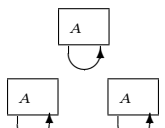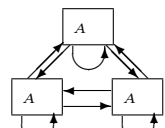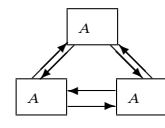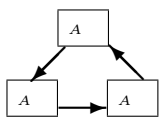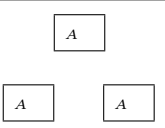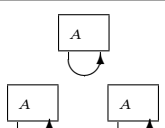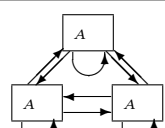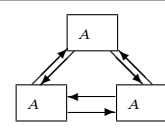
Rule	Graph	Semantics		
		$\sim$	$\tilde{\approx}$	$\equiv$
(R_1)	(G_1)	$\sim$	$\tilde{\approx}$	$\equiv$
	(G_2)	$\sim$	$\tilde{\approx}$	$\equiv$
(R_2)	(G_1)	$\sim$	$\tilde{\approx}$	$\equiv$
	(G_2)	$\sim$	$\tilde{\approx}$	$\equiv$
	(G_3)	$\sim$	$\tilde{\approx}$	$\equiv$
(R_3)	(G_1)	$\sim$	$\tilde{\approx}$	$\equiv$
	(G_2)	$\sim$	$\tilde{\approx}$	$\equiv$
	(G_3)	$\sim$	$\tilde{\approx}$	$\equiv$

Table 1: Effects of 'cyclic' rules

Let R be of the form (2). As shown in Example 3.7, condition (iii) in Definition 3.6 ensures that the only (minimal) way to generate a graph satisfying the rule is that of adding a self-loop for each node matching with A, α.

When ξ is $\stackrel{\rightarrow}{\sim}$ or $\equiv$, the situation is similar (and easier than for $\sim$ as concerns case (2)). □

Now we extend the above lemma to the case in which R contains several edges.

Proposition 4.6 *For each edge-adding rule* R, $[\![\, R \,]\!]_\xi$ *is a function, for* ξ *in* $\{\sim, \stackrel{\rightarrow}{\sim}, \equiv\}$.

Proof. R contains n green edges, with $n > 0$, by definition of rule. Consider the rules R_i, $i = 1, \ldots, n$, obtained by removing from R the green edges $1, \ldots, i-1, i+1, \ldots, n$. Each R_i is of the form analyzed by Lemma 4.5. Let G be a G-Log graph. Consider the following procedure, parametric with respect to ξ in $\{\sim, \stackrel{\rightarrow}{\sim}, \equiv\}$:

$$G'' := G;$$
repeat
$\quad G' := G'';$
$\quad\quad$ for $i = 1$ to n do
$\quad\quad\quad$ let G'' be the result of R_i applied to G'' with respect to ξ
until $G'' = G';$

The procedure is clearly terminating. Moreover, by induction on the number n of green edges, it is easy to prove that the procedure is ensured to return a unique graph G' (use Lemma 4.5), that G' ξ-satisfies R and it is the minimum graph extending G fulfilling such a property. □

4.4 Very Simple Queries

We consider simple rules, actually very used in practice:

Definition 4.7 *A rule* R *is said to be a* very simple rule *if* R_{GS} *consists in one node* μ *and one edge* $\langle \mu, lab, \nu \rangle$, *with* $\ell_{\mathcal{C}}(\nu) = RS$.

In this section we are interested in very simple rules that are queries (very simple queries—VSQ). In general, $[\![\, R \,]\!]_\xi$ applied to a concrete graph G is not a function, even when R is a very simple query with respect to G. Consider the following diagrams:[7]
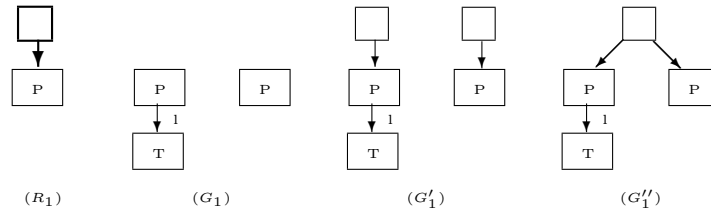
Then, $[\![\, R_1 \,]\!]_\xi(G_1) = \{G_1', G_1''\}$. However, the views of G_1' and G_1'' with respect to R_1 (see Definition 3.14) are bisimilar. So we could guess that $[\![\, \cdot \,]\!]_\xi$ is a function modulo bisimulation, at least with respect to a 'structured' subgraphs of G, i.e., graphs filtered by a rule. This does not hold in general, as follows from the following example concerning grandfathers, fathers, and sons:

---
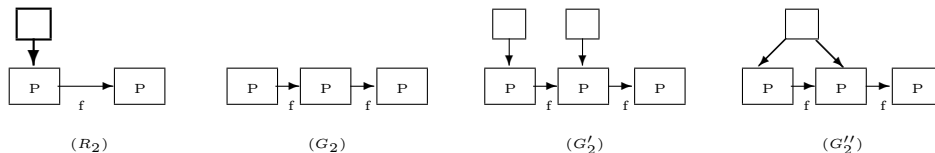
[7]As usual, square nodes can be read as *Result* nodes and outgoing edges as *connects* edges.

It holds that $[\![R_2]\!]_\xi(G_2) = \{G_2', G_2''\}$. However, G_2' and G_2'' are not bisimilar. The uniqueness (modulo bisimulation) is ensured only when the various parts of G matching with R_{RS} are all independent (unfolded views).

However, a sort of regularity of the semantics of rule application can be obtained by considering

$$G_\xi^R =_{def} \bigsqcup_{G' \in [\![R]\!]_\xi(G)} G'$$

Such a graph is unique (up to isomorphism) and it is a sort of *skeleton* from which all elements of $[\![R]\!]_\xi(G)$ can be obtained. As an instance, for the examples above:

Lemma 4.8 *Let G be a G-Log graph and R be a VSQ with respect to G. Then, for ξ in $\{\sim, \tilde{\sim}, \equiv\}$, there is a unique graph (up to isomorphism) G_ξ^R such that:*

1. *$G_\xi^R \models R$,*

2. *$\forall G' \in [\![R]\!]_\xi(G)$ it holds that $G' \sqsubseteq G_\xi^R$, and*

3. *G_ξ^R is minimal with respect to properties (1) and (2).*

Proof. If R is not ξ-applicable to G, then choose G_ξ^R as G. Otherwise since, by definition of query, no node labeled by *result* and edge labeled by *connects* is in G, whenever there is a subgraph G' of G such that $R_{RS}\xi G'$, add a node μ and an edge $\langle \mu, lab, \nu \rangle$. Moreover, keep track of all nodes μ, ν of this kind. When all the G' of that form have been processed, add edges from all nodes μ to all nodes ν, uniquely obtaining the graph G_ξ^R. □
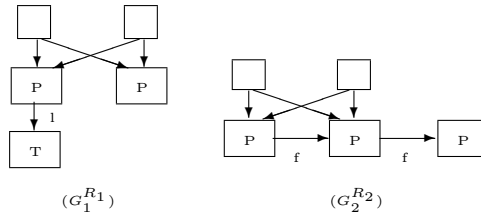
Proposition 4.9 *Let G be a G-Log graph and R be a VSQ with respect to G. For ξ in $\{\sim, \tilde{\sim}, \equiv\}$, define*

$$views(G, R, \xi) = \{G'^{\uplus}|_R : \exists G' \in [\![R]\!]_\xi(G)\}$$

Then for each $I_1, I_2 \in views(G, R, \xi)$, it holds that I_1 is isomorphic to I_2.

Proof. Assume $I_1, I_2 \in views(G, R, \xi)$, I_1 and I_2 distinct graphs. This means that there are two graphs G_1 and G_2 in $[\![R]\!]_\xi(G)$ such that $I_1 = G_1^{\uplus}|_R$ and $I_2 = G_2^{\uplus}|_R$. Since $G_i \in [\![R]\!]_\xi(G)$ it holds that for each $G' \sqsubseteq G$ such that $R_{RS}\xi G'$ there is an edge between a *Result* node (not occurring in G) and a node of G. This means, by definition of unfolded view, that a graph exactly composed by G' and the just mentioned node and edge is both in I_1 and in I_2, and, moreover, this is an isolated subgraph of both I_1 and I_2. Nodes and edges are introduced in I_1 and I_2 only in this way. This ensures that $I_1 \equiv I_2$. □

To reach a more convincing deterministic result for the semantics, we suggest to add determinism to the definition: we define the *deterministic semantics of R*:

$$[\![R]\!]_\xi^{det}(G) = G'$$

for $G' \in [\![R]\!]_\xi(G)$ and G' contains at most one node more than G.

Proposition 4.10 *Let G be a G-Log graph and R be a VSQ with respect to G. For ξ in $\{\sim, \stackrel{\leftrightarrow}{\sim}, \equiv\}$, then $[\![R]\!]_\xi^{det}(G)$ is well-defined, i.e., $[\![R]\!]_\xi^{det}(G)$ is a function.*

Proof. Assume, by contradiction, that $G_1, G_2 \in [\![R]\!]_\xi(G)$ and that they differ by G for at most one node.

If R is not ξ-applicable, then $G_1 = G_2 = G$ by definition.

Assume R is ξ-applicable. Since R is a query with respect to G, no result nodes are in G. Thus both G_1 and G_2 contains exactly one (result) node μ more than G. Without loss of generality, we can assume that it is the same node in the two graphs. New (connects) edges have been introduced from μ to the various subgraphs equivalent to R_{RS}. It is immediate to check that an edge of this form belongs to G_1 if and only if it belongs to G_2, unless one of them is not in $[\![R]\!]_\xi(G)$. $\qquad\square$

$[\![R]\!]_\xi^{det}(G)$ can therefore be seen as a *privileged* answer to a query. Actually, it contains exactly all the information we need and does not introduce redundant nodes.

We conclude this section with a consideration that explains the rationale behind the condition (3) of being a rule.

Remark 4.11 *Consider the graph R in Figure 11, that does not fulfill requirement (3) of being a rule. It intuitively says that for all nodes labeled A you need to have a node labeled by A connected with it by an edge labeled by p. The application of R to the trivial graph G generates a denumerable family $G'', G''', \ldots$ of graphs satisfying R. However, none of them is minimal. Moreover, notice that the graph G' does not satisfy R, as condition (3) of the definition of bisimulation is not satisfied.*
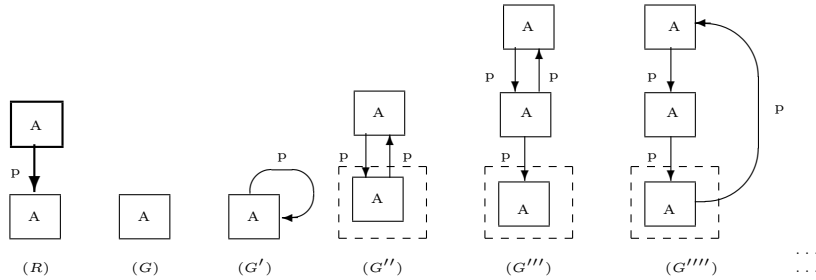
Figure 11: Infinite generation

5 Abstract Graphs and Semantics

In order to represent sets of instances sharing the same structure, we introduce now the notion of abstract graph. Following the Abstract Interpretation approach [10, 16], we see that abstract graphs can be used as a domain to abstract the computation of G-Log programs over concrete graphs. This can also be seen as an alternative view of reasoning on schemata and instances of a database or a WWW site, coherently with the *Dataguide* approach of [18].

Definition 5.1 *A (G-Log) abstract graph is a G-Log graph such that:*

1. $(\forall x \in N \cup E)(\ell_\mathcal{C}(x) = black)$,

2. $(\forall x \in N)(\ell_\mathcal{S}(x) = \bot)$, *i.e., an abstract graph has no values.*

With $\mathcal{G}^\mathcal{A}$ we denote the set of G-Log abstract graphs.
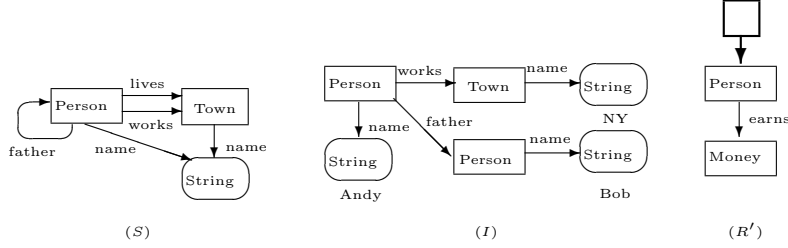
Figure 12: A schema, an instance, and a rule

Let us use once more the notion of bisimulation to re-formulate the G-Log concepts of instance and schema. Intuitively, an abstract graph represents a concrete graph if it contains its skeleton while disregarding multiplicities and values.

Definition 5.2 *A concrete graph I is* an instance *of an abstract graph G if $(\exists I' \sqsupseteq I)(G \sim I')$. In this case G is said to be a* schema *for I. I' is said to be a* witness *of the relation schema-instance.*

In Fig. 12 there is an example of application of the definition above. (S) represents (I). To build the witness (I'), add to (I) an edge labeled by *works* linking the entity node *Person* of *Bob* with the *entity node Town*. Moreover, add edges labeled by *lives* from the two nodes labeled *Person* to the node labeled *Town*, and add also an edge reverse to the *father* edge. It is easy to check that a bisimulation from S to I' is uniquely determined.

The notions of *applicability* and *satisfiability* for abstract graphs are the same as in Definitions 3.6 and 3.8. This also holds for the operational semantics definitions for rules and programs. Anyway, the semantics based on bisimulation $\sim$ is, in a sense, the less precise and, thus, it is the most suited for abstract computations.

The following properties can be immediately derived from the definitions above.

Lemma 5.3

(a) If I is an instance of G with witness I', then for all I'' such that $I \sqsubseteq I'' \sqsubseteq I'$ it holds that I'' is an instance of G.

(b) If I is a concrete graph, G is an abstract graph, with $I \sim G$, then I is an instance of G.

(c) If I is a concrete graph, G, G' are abstract graphs, with $G \sim G'$, then I is an instance of G if and only if I is an instance of G'.

We have already observed after Definition 2.2 that given a G-Log graph G_0, the set $\langle\{G$ is a G-Log graph : $G \sqsubseteq G_0\}, \sqsubseteq\rangle$ is a *complete lattice*.[8] In the rest of this section we assume that every (concrete and abstract) graph belongs to this lattice. Under this hypothesis we may properly deal with the $\sqsubseteq$ relation between graphs.

A Galois connection ([10]) between $\mathcal{G}^{\mathcal{A}}$ and $\wp(\mathcal{G}^{\mathcal{C}})$ can be obtained by considering the concretization function $\gamma : \mathcal{G}^{\mathcal{A}}_{/\sim} \longrightarrow \wp(\mathcal{G}^{\mathcal{C}}) :$

$$\gamma(G) = \{I \;:\; I \text{ is an instance of } G\}$$

and its adjoint abstraction function $\alpha : \wp(\mathcal{G}^{\mathcal{C}}) \longrightarrow \mathcal{G}^{\mathcal{A}}_{/\sim}$ defined by

$$\alpha(S) = \sqcup\{G \in \mathcal{G}^{\mathcal{A}} \;:\; \gamma(G) \subseteq S\}.$$

---

[8]Assume, for instance, to deal with Web sites. G_0 can be chosen as the graph obtained by disjoint union of all the instances and schemata of all Web sites.

Algorithmically, given a set of instances S, we may build up its abstraction $\alpha(S)$ by taking the union of all their nodes and edges. Moreover, applying standard techniques, we can build the minimum graph $\sim$-equivalent to that graph. This graph is unique up to isomorphism, and it can be computed without knowning G_0. Using the techniques in [32], this simplification can be performed in time $O(m \log n + n)$, where m is the number of edges and n the number of nodes.

The abstraction function for rules can be obtained exactly in the same way as for concrete graphs. Thus, when R is a rule, $\alpha(R)$ denotes the graph obtained by deleting values (i.e., $\ell_S = \bot$) from R and then computing the minimum graph $\sim$-equivalent to it.

The following two auxiliary results will be useful in order to prove monotonicity and injectivity of the function γ just defined.

Lemma 5.4 *If $G_1 = \langle N_1, E_1, \ell_1 \rangle \sqsubseteq G_2 = \langle N_2, E_2, \ell_2 \rangle$ and $G_1 \sim G' = \langle N', E', \ell' \rangle$, then there is $G'' \sqsupseteq G'$ such that $G'' \sim G_2$.*

Proof. Without loss of generality, assume that $N' \cap N_2 = \emptyset$. Let b' such that $G_1 \overset{b'}{\sim} G'$. Let $N'' = N'$, $E'' = E'$, $\ell'' = \ell'$, $b'' = b'$.

- for all $n \in N_2 \setminus N_1$ let $N'' = N'' \cup \{n\}$, $b'' = b'' \cup \{(n, n)\}$, and $\ell'' = \ell'' \cup \{(n, \ell_2(n))\}$;

- for all $\langle m, \lambda, n \rangle$ in $E_2 \setminus E_1$, let $E'' = E'' \cup \{\langle \mu, \lambda, \nu \rangle : m b'' \mu, n b'' \nu\}$.

It is immediate to check that $G_2 \overset{b''}{\sim} G'' = \langle N'', E'', \ell'' \rangle$. □

Lemma 5.5 *If $G \sqsubseteq G_1 \sim G_2 \sqsubseteq G_3 \sim G$, then $G \sim G_1 \sim G_2 \sim G_3$.*

Proof. It is sufficient to prove that $G \sim G_1$; the remaining part of the claim follows by the fact that $\sim$ is an equivalence relation. Let a and b be the two bisimulations such that $G_1 \overset{a}{\sim} G_2$ and $G_3 \overset{b}{\sim} G$. By Lemma 5.4, there is G_0 such that $G_1 \sqsubseteq G_0$ and $G_0 \overset{a'}{\sim} G_2$, where a' extends a as in the proof of that lemma. We will refer to a' simply as a and we call $c = a \circ b$. It is easy to verify that c is monotonic; thus we can build two infinite descending chains:

$$G_1 \sqsupseteq c(G_1)(\sqsupseteq G) \sqsupseteq c(c(G_1)) \sqsupseteq c(c(c(G_1))) \ldots$$
$$G_0 \sqsupseteq c(G_0)(= G) \sqsupseteq c(c(G_0)) \sqsupseteq c(c(c(G_0))) \ldots$$

Since the graphs are finite, there must be an integer n such that

$$c^n G_1 = c^{n+1} G_1 \wedge c^n G_0 = c^{n+1} G_0$$

(assuming that the graph G is nonempty, the fixed points above must be non empty). Moreover, by construction, it holds that:

$$G_1 \sim c(G_1) \sim c(c(G_1)) \sim c(c(c(G_1))) \ldots$$
$$G_0 \sim c(G_0)(= G) \sim c(c(G_0)) \sim c(c(c(G_0))) \ldots$$

In particular, $G_1 \sim c^n(G_1)$ and $G \sim c^n(G_0)$. Since the application of relation c is monotonic, it holds that

$$c^i(G_1) \sqsubseteq c^i(G_0),$$

Thus, in particular, $c^n(G_1) \sqsubseteq c^n(G_0)$. On the other hand, since $c(G_0) = G$ and $G \sqsubseteq G_1$, it holds that

$$c^{i+1}(G_0) \sqsubseteq c^i(G_1).$$

Thus, $c^n(G_0) = c^{n+1}(G_0) \sqsubseteq c^n(G_1)$. This means that $c^n(G_1) = c^n(G_0)$ and, moreover, that $G \sim c^n(G_1) \sim G_1$. □

Theorem 5.6 *Function γ is monotonic, i.e. for any pair of abstract graphs G, G', $G \sqsubseteq G'$ implies $\gamma(G) \subseteq \gamma(G')$.*

Proof. Let $I \in \gamma(G)$. By definition of γ, there exists $I' \sqsupseteq I$ such that $I' \sim G \sqsubseteq G'$. By applying Lemma 5.4, there exists $I'' \sqsupseteq I'$ such that $I'' \sim G'$. By transitivity of the ordering relation, we get $I'' \sqsupseteq I$. Hence, $I \in \gamma(G')$. $\qquad\square$

Theorem 5.7 *Function γ is injective, i.e. for any pair of abstract graphs G, G', $G \not\sim G'$ implies $\gamma(G) \neq \gamma(G')$.*

Proof. Assume that $\gamma(G_1) = \gamma(G_2)$, and let $I_1 \in \gamma(G_1)$ with $I_1 \sim G_1$. By the assumption, $I_1 \in \gamma(G_2)$ too. Hence, by the definition of γ, $\exists I_1' \sqsupseteq I_1$ such that $I' \sim G_2$. Therefore,

$$G_1 \sim I_1 \sqsubseteq I_1' \sim G_2.$$

Now, let $I_2 \in \gamma(G_2)$ with $I_2 \sim G_2$. By the same reasoning, there exists $I_2' \sqsupseteq I_2$ such that $I_2' \sim G_1$. Therefore

$$G_1 \sim I_1 \sqsubseteq I_1' \sim G_2 \sim I_2 \sqsubseteq I_2' \sim G_1.$$

By lemma 5.5 we immediately get $G_1 \sim G_2$, concluding the proof.

$\qquad\square$

Theorem 5.8 *(Correctness) Let G, G' be abstract graphs and R a rule such that $\langle G, G' \rangle \in [\![\,\alpha(R)\,]\!]_\sim$. If $I \in \gamma(G)$ and $\langle I, I' \rangle \in [\![\,R\,]\!]_\sim$, then $I' \in \gamma(G')$, i.e., the following diagram commutes:*

$$
\begin{array}{ccc}
G & \overset{\alpha(R)}{\to} & G' \\
\downarrow \gamma & & \downarrow \gamma \\
I & \overset{R}{\to} & I'
\end{array}
$$

Proof. By the hypotheses and by Lemma 5.4, there exist $\hat{I}$ and $\hat{I}'$ such that the following diagram holds:

$$
\begin{array}{ccccccccc}
R & \rightleftharpoons & I' & & \hat{I}' & \sim & G' & \models & \alpha(R) \\
& & \sqcup & & \sqcup & & \sqcup & & \\
R|_{\{RS\}} & \sqsubseteq & I & \sqsubseteq & \hat{I} & \sim & G & \sqsupseteq & \alpha(R)|_{\{RS\}}
\end{array}
$$

By the definition of satisfiability of the rule R, we may build I'' such that $I' \sqsubseteq I'' \sim \hat{I}'$. In order to build such a I'', extend I' only with arcs and nodes belonging to $\hat{I}'$; minimality conditions on G' (and thus on $\hat{I}'$) avoid redundancies. Hence, from the diagram above we get $I' \sqsubseteq I'' \sim \hat{I}' \sim G'$, i.e. $I' \in \gamma(G')$.

$\qquad\square$

Theorem 5.8 guarantees the correctness of abstract computations: the application of a rule abstraction to an abstract graph safely represents the application of the corresponding concrete rule to any of its instances. The practical impact of this result is quite interesting. Consider the abstract graph S and the rule R' in Fig. 12. Since $\alpha(R')$ is not applicable to S, we can immediately conclude that the same rule is not applicable to any instance of S. Therefore, we may apply rules to abstract graphs in order to build complex queries, and then, once checked that they are applicable to the abstract graph we can turn to the concrete cases to get the desired answer. This is particularly interesting when the instance resides on a remote site.

Moreover, suppose we use G-Log rules to specify site instance evolution during the site life. Then, the application of the same rule to the site schema returns automatically the schema corresponding to the new site instance.[9]

---

[9]Of course, in this context we are interested in those site updates that would affect the schema, since schema-invariant updates do not need to be traced.

Remark 5.9 *According to standard definition of schema, Definition 5.1 may be further enforced with the condition:*

  3. $(\forall x, y \in N)(\ell_\mathcal{T}(x) \neq \ell_\mathcal{T}(y) \vee \ell_\mathcal{L}(x) \neq \ell_\mathcal{L}(y))$, i.e. there is no repetition of nodes.

In this case, given a set S of instances, we may build up its abstraction $\alpha'(S)$ computing $\alpha(S)$ and then by collapsing all the nodes in it having the same type and label. The same technique can be applied to a rule R. α' can be seen as an abstraction less precise than α; by construction, it holds that $\alpha'(\alpha(S)) = \alpha'(S)$. The concretization function γ remains the same.

When R is a query, Theorem 5.8 still holds using α' in place of α. In Figure 13 we present the concrete graph I, the abstract graph $G = \alpha(I)$ and the schema $S = \alpha'(I)$. For each rule R_i, $\alpha(R_i) = \alpha'(R_i)$ is obtained by removing the concrete value label (in these cases, Udine and Verona). It holds that:

- *$\{I'\} = [\![R_1]\!](I)$, $\{I\} = [\![R_2]\!](I) = [\![R_3]\!](I)$;*

- *$\{G'\} = [\![\alpha(R_1)]\!](G) = [\![\alpha(R_2)]\!](G)$, $\{G\} = [\![\alpha(R_3)]\!]G$;*

- *$\{S'\} = [\![\alpha'(R_1)]\!](S) = [\![\alpha'(R_2)]\!](S) = [\![\alpha'(R_3)]\!](S)$.*

When, as in this case, rule application is deterministic, the two levels of abstractions can be summarized by the following diagram:

$$
\begin{array}{ccc}
S & \xrightarrow{\alpha'(R)} & S_f \\
& & \sqcup_| \\
\uparrow \alpha' & & \alpha'(G_f) \\
& & \uparrow \alpha' \\
G & \xrightarrow{\alpha(R)} & G_f \\
& & \sqcup_| \\
\uparrow \alpha & & \alpha(I_f) \\
& & \uparrow \alpha \\
I & \xrightarrow{R} & I_f
\end{array}
$$

This leads to a hierarchy of abstraction in the spirit of [16].

6 Logical Semantics of G-Log

Aim of this section is to provide a model theoretic characterization of the language G-Log. First, we show how to automatically extract a first-order formula from a G-Log graph. Then we show that G-Log concrete graphs are simply representations of Herbrand structures: they are models of the formulae associated with the rules they satisfy.

As said in Section 2.2, *result* nodes and their outgoing edges labeled by *connects* are represented without writing explicitly the labels. We write $\phi(x_1, \ldots, x_n)$ to denote that ϕ is a first-order formula with free variables among $x_1, \ldots, x_n$. Moreover, $[\ell_\mathcal{N}(n)](x_1, \ldots, x_n)$ denotes the atom $p(x_1, \ldots, x_n)$ where p is $\ell_\mathcal{N}(n)$. Similarly for $\ell_\mathcal{L}$ and $\ell_\mathcal{S}$.

6.1 Formulae for G-Log Rules

In this subsection we describe how to obtain a first-order formula from each G-Log rule.

Definition 6.1 *A G-Log formula is a closed first-order formula of the following form:*

$$\forall x_1 \cdots x_h \, (B_1(x_1, \ldots, x_h) \rightarrow \exists z_1 \cdots z_k \, B_2(x_1, \ldots, x_h, z_1, \ldots, z_k))$$

where x_i, z_i are variables and the B_i are conjunctions of atoms.
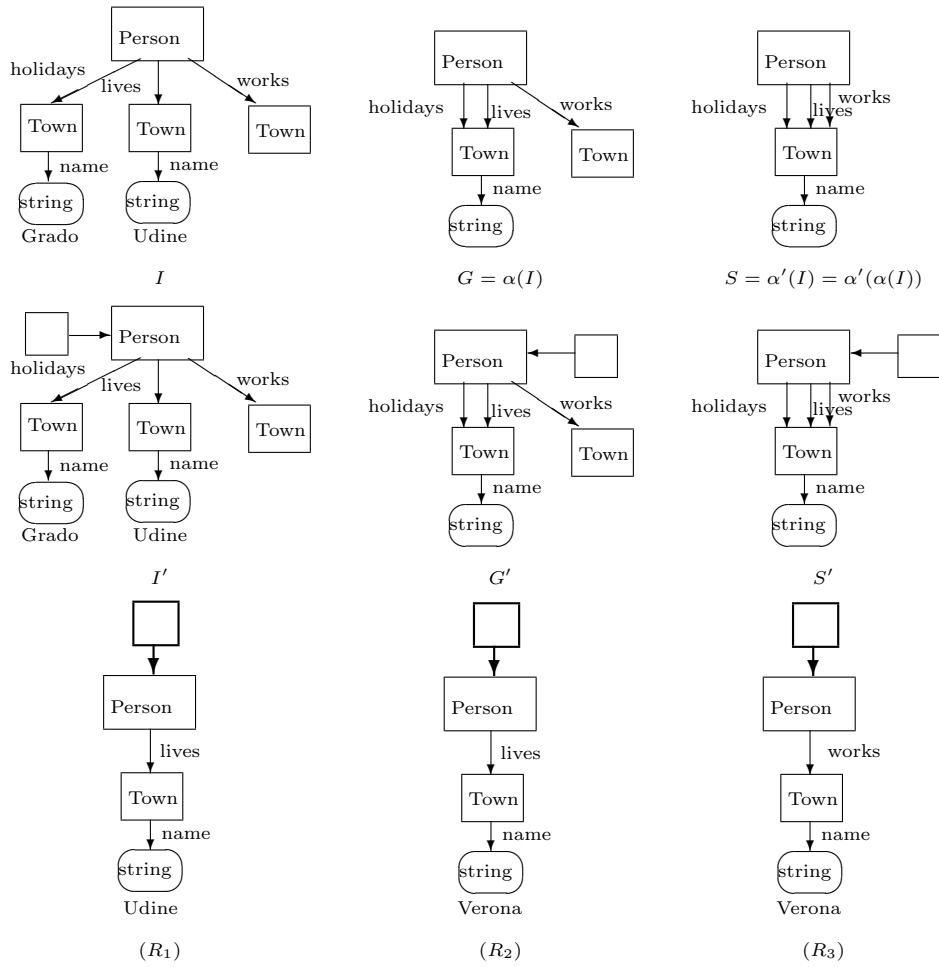
Figure 13: Two levels of abstraction

Remark 6.2 *Observe that the existential quantification of the variable on the r.h.s. of the implication causes that the formula can not be encoded as a simple Horn clause of DATALOG.*

We show how to associate a G-Log formula to every G-Log rule. We begin with the semantics based on directional bisimulation $\vec{\sim}$; then we show how to modify the technique according to the other two semantics.

Definition 6.3 *Let $R = \langle N, E, \ell \rangle$ be a G-Log rule; we define the G-Log formula $\Phi_R^{\vec{\sim}}$ and the formula $\Psi_R^{\vec{\sim}}$ as follows:*

1. *$\forall n \in N$ associate a distinct variable $\nu(n)$.*

2. *$\forall n \in N$ let φ_n be the formula: $[\ell_{\mathcal{L}}(n)](\nu(n)) \wedge [\ell_{\mathcal{T}}(n)](\nu(n)) \wedge [\ell_{\mathcal{S}}(n)](\nu(n))$. If $\ell_{\mathcal{S}}(n) = \bot$, then the last conjunct is omitted.*

3. *$\forall e = \langle m, \langle c, \ell_{\mathcal{L}}(e) \rangle, n \rangle \in E$ let φ_e be the formula: $[\ell_{\mathcal{L}}(e)](\nu(m), \nu(n))$.*

4. *Let $n_1, \ldots, n_h$ be the nodes of N such that $\ell_{\mathcal{C}}(e) = RS$,*

5. *Let $n_1', \ldots, n_k'$ be the nodes of N such that $\ell_{\mathcal{C}}(e) = GS$,*

6. *The formula $\Phi_R^{\vec{\sim}}$ is:*

$$
\forall \nu(n_1) \cdots \nu(n_h) \left(\begin{array}{c} \left(\bigwedge_{n \in N, \ell_{\mathcal{C}}(n) = RS} \varphi_n \wedge \bigwedge_{e \in E, \ell_{\mathcal{C}}(e) = RS} \varphi_e \right) \rightarrow \\[2em] \exists \nu(n_1') \cdots \nu(n_k') \left(\bigwedge_{n \in N, \ell_{\mathcal{C}}(n) = GS} \varphi_n \wedge \bigwedge_{e \in E, \ell_{\mathcal{C}}(e) = GS} \varphi_e \right) \end{array} \right)
$$

7. *The formula $\Psi_R^{\vec{\sim}}$ is:*

$$
\exists \nu(n_1) \cdots \nu(n_h) \left(\bigwedge_{n \in N, \ell_{\mathcal{C}}(n) = RS} \varphi_n \wedge \bigwedge_{e \in E, \ell_{\mathcal{C}}(e) = RS} \varphi_e \right)
$$

(R)
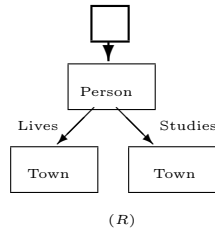
Figure 14: G–Log rule

For instance, the formula $\Phi_R^{\vec{\sim}}$ associated to the rule R of Fig. 14 is:[10]

$$
\forall x_1 x_2 x_3 \left(\begin{array}{l} \mathrm{Person}(x_1) \wedge \mathrm{Town}(x_2) \wedge \mathrm{Lives}(x_1, x_2) \wedge \mathrm{Town}(x_3) \wedge \mathrm{Studies}(x_1, x_3) \rightarrow \\ \exists z_1 (\mathrm{result}(z_1) \wedge \mathrm{connects}(z_1, x_1)) \end{array} \right)
$$

---

[10]We omit the type information for the sake of readability.
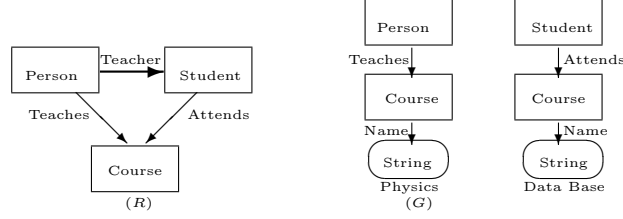
Figure 15: A G–Log rule and a concrete graph

Logical formulae corresponding to G-Log graphs are different if we study the other two semantics. With the semantics based on the concept of graph isomorphism $\equiv$, rule R of Fig. 14 represents the query 'collect all the people living and studying in two *different* towns'.

In the construction of $\Phi_R^{\equiv}$ we need to force the fact that the two towns must be distinct. This can be done by adding an inequality constraint between the variables identifying the nodes x_2 and x_3:

$$\forall x_1 x_2 x_3 \left(\begin{array}{l} \text{Person}(x_1) \wedge \text{Town}(x_2) \wedge \text{Lives}(x_1, x_2) \wedge \text{Town}(x_3) \wedge \text{Studies}(x_1, x_3) \wedge x_2 \neq x_3 \rightarrow \\ \exists z_1 (\text{result}(z_1) \wedge \text{connects}(z_1, x_1)) \end{array} \right)$$

More generally, we will require that all nodes of the graph are distinct:

Definition 6.4 *Given a G-Log rule $R = \langle N, E, \ell \rangle$, and the formula*

$$\Phi_R^{\vec{\sim}} = \forall \nu(n_1) \cdots \nu(n_h) \left(B_1 \rightarrow \exists \nu(n'_1) \cdots \nu(n'_k) B_2 \right) ,$$

then the formula $\Phi_R^{\equiv}$ is:

$$\forall \nu(n_1) \cdots \nu(n_h) \left(\begin{array}{l} \left(B_1 \wedge \bigwedge_{1 \leq i < j \leq h} \nu(n_i) \neq \nu(n_j) \right) \rightarrow \\ \exists \nu(n'_1) \cdots \nu(n'_k) \left(B_2 \wedge \bigwedge_{1 \leq i < j \leq k} \nu(n'_i) \neq \nu(n'_j) \wedge \bigwedge_{1 \leq i \leq h, 1 \leq j \leq k} \nu(n_i) \neq \nu(n'_j) \right) \end{array} \right)$$

Similarly, formula $\Psi_R^{\equiv}$ can be obtained by adding $\bigwedge_{1 \leq i < j \leq h} \nu(n_i) \neq \nu(n_j)$ to the conjuncts of $\Psi_R^{\vec{\sim}}$.

Consider now the semantics based on $\sim$, the rule R, and the concrete graph G of Fig. 15. R is $\sim$-applicable (Definition 3.5) to (G) but is not ξ-applicable for ξ in $\{\equiv, \vec{\sim}\}$.

The G-Log formula $\Phi_R^{\vec{\sim}}$:

$$\forall x_1 x_2 x_3 \left(\begin{array}{l} \text{Person}(x_1) \wedge \text{Student}(x_2) \wedge \text{Course}(x_3) \wedge \text{Teaches}(x_1, x_3) \wedge \text{Attends}(x_2, x_3) \rightarrow \\ \text{Teacher}(x_1, x_2) \end{array} \right)$$

represents the query 'if a person teaches a course and a student attends the *same* course, then the person is that student's teacher'. The semantics based on bisimulation requires a weaker condition, since the constraint 'the same' cannot be forced. This means that, in the $\sim$-semantics the rule requires that whenever a student attends a course and a person teaches some (other?) course, the person is that student's teacher.

Definition 6.5 *Given a G-Log rule R, we define* the unfolding of R, *briefly unf(R), to be the graph obtained by replacing every subgraph of R of the form*
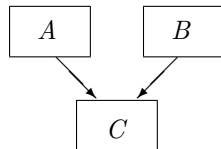
with the subgraph:

Then we set $\Phi_R^{\sim} = \Phi_{unf(R)}^{\vec{}}$ and $\Psi_R^{\sim} = \Psi_{unf(R)}^{\vec{}}$.

For instance, the formula $\Phi_R^{\sim}$ for the rule R of Figure 15 is:

$$\forall x_1 x_2 x_3 x_4 \left(\begin{array}{l} \text{Person}(x_1) \wedge \text{Student}(x_2) \wedge \text{Course}(x_3) \wedge \\ \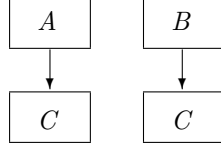text{Course}(x_4) \wedge \text{Teaches}(x_1, x_3) \wedge \text{Attends}(x_2, x_4) \rightarrow \\ \text{Teacher}(x_1, x_2) \end{array} \right)$$

that does not constraint the courses to be the same.

In Section 6.3 we formally prove that the logical semantics of rules we are describing is consistent with the operational semantics.

6.2 Concrete Graphs as Models

In this subsection we show, independently of the operational rule, how to obtain a first-order formula, from a concrete graph; in particular, we show that concrete graphs are representations of the least Herbrand model (modulo isomorphism) of the Skolemization of that formula.

Definition 6.6 *Let $G = \langle N, E, \ell \rangle$ be a concrete graph. As in Definition 6.3, to every $n \in N = \{n_1, \ldots, n_h\}$ associate a variable $\nu(n)$ and to every node n and edge e associate the formulae φ_n and φ_e, respectively. Then, the formula Φ_G associated to G is:*

$$\exists \nu(n_1) \cdots \nu(n_h) \bigwedge_{n \in N} \varphi(n) \wedge \bigwedge_{e \in E} \varphi(e)$$

Definition 6.7 *Let $G = \langle N, E, \ell \rangle$ be a concrete graph. We associate to G the structure $M_G = \langle D, I \rangle$ built as follows:*

1. *for every node $n \in N$ introduce a constant c_n; let $D = \{c_n : n \in N\}$.*

2. *$I(p(c_n)) = \mathsf{true}$ if and only if $p(\nu(n))$ is a conjunct of Φ_G.*

3. *$I(p(c_m, c_n)) = \mathsf{true}$ if and only if $p(\nu(m), \nu(n))$ is a conjunct of Φ_G.*

M_G is the *least* Herbrand model of the Skolemization of the formula Φ_G.

For example, let G be the concrete graph of Fig. 15, and c_i, $i = 1, \ldots, n$ the constants introduced for the nodes of the concrete graph. Then M_G can be expressed by the set of facts that are true:

$$\begin{array}{ll} \text{Person}(c_1), \text{entity}(c_1), & \text{Course}(c_2), \text{entity}(c_2) \\ \text{String}(c_3), \text{slot}(c_3), \text{Physics}(c_3), & \text{Student}(c_4), \text{entity}(c_4), \\ \text{Course}(c_5), \text{entity}(c_5) & \text{String}(c_6), \text{slot}(c_6), \text{Data Base}(c_6), \\ \text{Teaches}(c_1, c_2), \text{Name}(c_2, c_3), & \text{Attends}(c_4, c_5), \text{Name}(c_5, c_6) \end{array}$$

6.3 Model Theoretic Semantics

In this subsection we highlight the relationships between the operational semantics of Section 3 and the logical view of G-Log graphs presented in Subsections 6.1 and 6.2.

Proposition 6.8 (Applicability) *Let G be a concrete graph and R a rule. Then R is ξ-applicable to G if and only if $M_G \models \Psi_R^\xi$.*

Proof. We prove first the claim when ξ is $\vec{\sim}$. R is $\vec{\sim}$-applicable to G means that there is $G_1 \sqsubseteq G$ such that $R_{RS} \vec{\sim} G_1$. Thus, there is a function f from the nodes of R_{RS} to those of G_1 fulfilling the requirements of $\vec{\sim}$. Using that f we find exactly the constants c_i of the domain D obtained by skolemization of Φ_G to be assigned to the existentially quantified variables $\nu(n_i)$ of $\Psi_R^{\vec{\sim}}$ to ensure that $M_G \models \Psi_R^{\vec{\sim}}$. Similarly, starting from an assignment ensuring $M_G \models \Psi_R^{\vec{\sim}}$ we can build a function f such that $R_{RS} \vec{\sim} G_1$ for some $G_1 \sqsubseteq G$.

To conclude the proof, notice that when computing $\Psi_{\vec{\equiv}}^\xi$ and $\Psi_\sim^\xi$ we have kept into account the constraint to map distinct nodes into distinct objects, and the possibility given by the unfolding of a node to be mapped into distinct objects, respectively. □

Proposition 6.9 (Satisfiability) *Let G be a concrete graph and R a rule. Then G ξ-satisfies R if and only if $M_G \models \Phi_R^\xi$.*

Proof. We prove first the claim when ξ is $\vec{\sim}$. G $\vec{\sim}$-satisfies G when for all $G_1 \sqsubseteq G$ such that $R_{RS} \vec{\sim} G_1$ there is a $G_2 \sqsupseteq G_1$ such that $R \vec{\sim} G_2$. But this is exactly the meaning of the formula $\Phi_R^{\vec{\sim}}$.

To conclude the proof, notice that the way to compute Ψ_R^ξ when ξ is $\sim$ or $\equiv$ ensures that the result holds. □

Proposition 6.10 (Rule application) *Let G be a G-Log concrete graph and R a rule. Then, for $\xi \in \{\sim \vec{\sim}, \equiv\}$,*

$$
\begin{aligned}
G' \in [\![R]\!]_\xi(G) \quad \rightarrow \quad & (M_G \not\models \Psi_R^\xi \wedge G = G') \vee \\
& (M_G \models \Psi_R^\xi \wedge M_G \models \Phi_R^\xi \wedge G = G') \vee \\
& (M_G \models \Psi_R^\xi \wedge M_G \not\models \Phi_R^\xi \wedge G' \neq G \wedge M_{G'} \models \Phi_R)
\end{aligned}
$$

Proof. The proof is by case analysis. Assume $G' \in [\![R]\!]_\xi(G)$.

1. If R is not ξ-applicable to G then $G' = G$ by definition. From Proposition 6.8 it holds that $M_G \not\models \Psi_R^\xi$. Since the l.h.s. of the implication of Φ_R^ξ is false, then trivially $M_G \models \Phi_R^\xi$.

2. If, conversely, R is ξ-applicable to G (and from Proposition 6.8 $M_G \models \Psi_R^\xi$) either:

    (a) G ξ-satisfies R (and thus, by Proposition 6.9, $M_G \models \Phi_R^\xi$), or

    (b) G does not ξ-satisfy R (and thus, by Proposition 6.9, $M_G \not\models \Phi_R^\xi$).

    In the former case $G' = G$; in the latter there is a $G' \sqsupseteq G$ such that G' ξ-satisfies R. Thus, by Proposition 6.9, $M_{G'} \models \Phi_R^\xi$.

□

Notice that it can be the case that

$$M_G \models \Psi_R^\xi \wedge M_G \not\models \Phi_R^\xi \wedge G' \neq G \wedge M_{G'} \models \Phi_R$$

but $G' \notin [\![R]\!]_\xi(G)$. This happens when G' is not a minimal extension of G. Thus, the converse direction of th above proposition is not always true. However:

Corollary 6.11 *Let G be a G-Log concrete graph and R a rule. Then, for $\xi \in \{\sim, \vec{\sim}, \equiv\}$,*

$$\exists G'(G' \in [\![R]\!]_\xi(G) \wedge G' \neq G) \quad \leftrightarrow M_G \models \Psi_R^\xi \wedge M_G \not\models \Phi_R^\xi \wedge (\exists G' \sqsupseteq G)(M_{G'} \models \Phi_R)$$

Proof. The ($\rightarrow$) direction follows immediately from Proposition 6.10. Assume now that $M_G \models \Psi_R^\xi \wedge M_G \not\models \Phi_R^\xi \wedge (\exists G' \sqsupseteq G)(M_{G'} \models \Phi_R)$. From Propositions 6.8 and 6.9 we know that R is ξ-applicable to G and G does not ξ-satisfy R. Then, by Proposition 4.3 this is sufficient to ensure that there is a minimal G' extending G and ξ-satisfying R. $\qquad\square$

To sum up, given a rule R and a graph G, the model-theoretic interpretation of the rule application is that of finding a (minimal) $G' \sqsupseteq G$ such that $M_{G'} \models \Phi_R^\xi$.

More generally, the effect of the application of the consecutive rules $R_1, \ldots, R_n$ to an initial concrete graph G is that of producing a (non-deterministic) path of the form:

$$G \stackrel{R_1}{\Rightarrow} G_1 \stackrel{R_1}{\Rightarrow} \cdots \stackrel{R_n}{\Rightarrow} G_n$$

where $M_{G_i} \models \Phi_{R_j}^\xi$ for all $j \leq i$.

As a final remark, also for abstract graphs it is possible to develop a logical semantics. However, while a concrete graph leads to an existential formula, an abstract graph leads to universally quantified formulae in which a lot of *closure* properties are to be stored. The difficulty of handling those formulae requires further work.

7 Comparisons and Future Work

7.1 Related proposals

A graphical language that shows many similarities with G-Log is Graphlog [9], a declarative language with graph-based data model and rules. Graphlog is a deterministic language intended for the relational model, it is not Turing-complete, and allows no sequences of rules. Moreover, Graphlog query graphs are acyclic and the queries, which require patterns to be present in (or absent from) the database graph, are supposed to extend databases only with new edges (i.e. can only define new relations). Conversely, G-Log was originally developed as a language and data model for *complex objects with identity* [1], and in its full form it is Turing complete ([31]). The structure and the meaning of queries in the two languages are rather similar, but cycles are allowed in G-Log, and G-Log rules enable the user to extend the databases both with entities and relations.

Similarities can also be found between our approach and previous works on UnQL [5], where the notion of bisimulation is used for investigating query decomposition. However, differences between G-Log and UnQL are quite deep. For instance, when assigning semantics to the language basic blocks, we allow information to be located in graph nodes, while UNQL locates information on edges; more importantly, G-Log queries are written directly in the graph formalism, while UNQL describes data instances graphically, and the query language of UNQL is SQL-like. Moreover, G-Log allows to express cyclic information and queries, and achieves its high expressive power by allowing a fully user-controlled non-determinism.

Anyway, keeping in mind these differences the results of our work can be also applied to Graphlog and UNQL, and in general to any graphical language whose main aim is to query and transform a graph-based data model by using graph-based rules.

For further comparisons between graphical query languages see [29].

7.2 Relationship with the Original G-Log Semantics

In this subsection we wish to point out the connections of a bisimulation-based semantics with the embedding-based semantics of [31, 29]. To complete the Definition 3.2, b is a directional *pseudo-bisimulation*, denoted
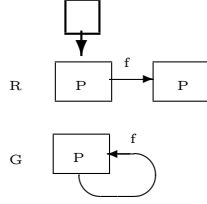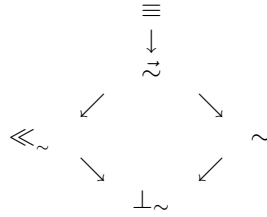
Figure 16: R is $\ll_\sim$-applicable to G

by $G_0 \overset{b}{\ll_\sim} G_1$, if there is a function b from the nodes of G_0 to the nodes of G_1 fulfilling conditions 1 and 2 of the definition of bisimulation and, moreover, condition 3 for $i = 0$. We say that $G_0 \ll_\sim G_1$ if there is a b such that $G_0 \overset{b}{\ll_\sim} G_1$. $\ll_\sim$ is used to build a semantics based on the notion of *embedding* as given in [29].

It is immediate to extend the Proposition 4.1 proving that if R is $\overset{\leftrightarrow}{\sim}$-applicable, then it is $\ll_\sim$-applicable. However, also this implication is strict: in Figure 16 it is represented a rule R applicable to a graph G only by this semantics.

Thus, the naturally induced ordering among relations is depicted by:

$$
\begin{array}{ccc}
 & \equiv & \\
 & \downarrow & \\
 & \overset{\leftrightarrow}{\sim} & \\
 \swarrow & & \searrow \\
\ll_\sim & & \sim \\
 \searrow & & \swarrow \\
 & \bot_\sim &
\end{array}
$$

The bottom element, say $\bot_\sim$, of the above graph exists: $G \bot_\sim G'$ if there is a relation (not necessarily a function!) b fulfilling conditions 1 and 2 of the definition of bisimulation and, moreover, condition 3 for $i = 0$. The first Example of Fig. 10 denotes a case of applicability for $\sim$ but not for $\ll_\sim$.

7.3 G-Log Graphs with Negation

Among the future work (and actually, under development) we plain to extend the semantics in order to deal with rules and programs with negation (i.e., containing red dashed nodes and edges—c.f. Section 2.1).

Intuitively, dashed edges express negative information; consider, for instance, R and G as in Fig. 17. R intuitively means 'collect all the people that do not live in a town named Verona'. The fact that graphs G' and G'' satisfy R can be formalized by extending the definitions of [29] concerning negation according to our semantics. However,

- G' can be obtained from G by using a sort of *failure rule* or, almost equivalently, by applying the *Closed World Assumption*: we infer that 'Ago does not live in Verona' from the fact that we cannot derive that this fact is true.

- G'' is obtained by adding the hypothesis 'Ago lives in Verona' that ensures that no subset of G fulfills R.

This kind of non-determinism is dealt with in [29] and [31] by limiting the G-Log programs to queries. We plan to study this and the related issues in the near future.
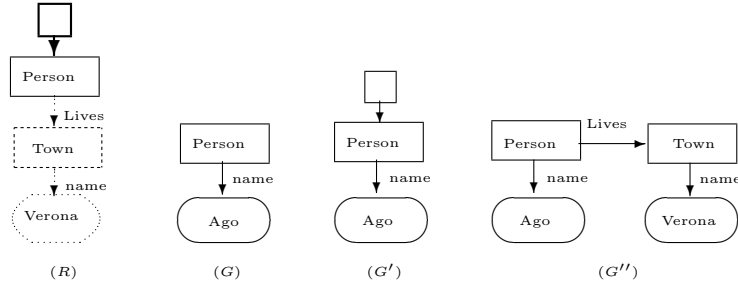
Figure 17: Negation as failure

7.4 Computational Issues

During the implementation of the operational semantics of the language G-Log, two typical graph problems must be faced: given two graphs G_1 and G_2,

1. to verify if $G_1 \xi G_2$, and

2. to verify if there is $G_3 \sqsubseteq G_1$ such that $G_3 \xi G_2$.

In the case of the relation $\equiv$, problem 1 is known as *graph isomorphism*, while problem 2 is the *subgraph isomorphism*. The former is in NP but still it is not known whether it is NP-complete or it is in P.[11] The latter in NP-complete [17].

In the case of bisimulation ($\sim$), problem 1 is polynomial ($O(m \log n + n)$, where m is the number of edges and n the number of nodes.; see, e.g. [32]).

An interesting work is to characterize all the remaining problems from a computational point of view, and then use these results to improve the performances of the old implementation of the languages G-Log and WG-Log based on the notion of embedding (cf. Section 7.2).

8 Conclusions

We have presented a new version of the semantics of the language G-Log, a graph-based query language originally designed for the representation and querying of object-based data. G-Log embodies the formal basis of the WG-Log system, which proposes a language and an architecture for querying and restructuring Web site data and, more generally, semistructured information. The results obtained in this paper allow a deeper understanding of some subtle ambiguities of the original semantics of G-Log, while proposing three alternative semantics which improve on the complexity of query computation in a significant way. Moreover, given that we use WG-Log schemata in order to represent sets of sites having the same structure, the results on abstraction provide the following important applicative consequences in the WG-Log context:

- *graceful tolerance of data dynamics*, i.e. an easy mechanism for schema updates resulting from instance evolution over time (Correctness Theorem);

- *efficient checking of instance correctness* with respect to a given schema (abstraction and concretization processes, functions α and γ);

- *efficient checking (i.e. at the schema level) of query applicability* to a certain instance (Correctness Theorem);

---

[11] Actually, it is one of the candidates for membership in the (hypothetical) intermediate class NPI [17].

- *semi-automatic integration of heterogeneous datasources*: components of a heterogeneous database can be translated into a language similar to G-Log, by a standard wrapper at the instance level, and later a unified, schematic representation of the whole set of data can be automatically derived (abstraction process, function α).

As a conclusion, we believe that all the advantages afforded by the adoption of the bisimulation semantics perspective well account for the choice of working directly on the graph-based representation of G-Log, rather that on its logical counterpart.

Some important issues are only tackled in this paper; noticeably, queries involving negation have not been deeply examined yet: this is an issue for future research, together with the study of appropriate algorithms to implement efficiently the various semantics. Recently it has been shown how to implement in linear time the task of finding a subgraph bisimilar to a given one (one of the key actions to be implemented for the operational semantics) for a wide family of graphs [14].

Acknowledgements

References

[1] A. Abiteboul and P. Kanellakis. Object Identity as a Query Language Primitive. In *Proceedings of the 1989 SIGMOD International Conference on the Management of Data*, Sigmod Record, vol. 19, June 1990.

[2] P. Aczel. *Non-well-founded sets.* Vol. 14 of *Lecture Notes, Center for the Study of Language and Information.* Stanford, 1988.

[3] G. Arocena and A. Mendelzon. WebOQL: Restructuring documents, databases, and webs. In *Proceedings of the Fourteenth International Conference on Data Engineering.* IEEE Computer Society Press, pages 24–33, 1998.

[4] P. P. Chen. The entity-relationship model: toward a unified view of data. *ACM Trans. on DB systems*, 1(1):9–36, 1976.

[5] P. Buneman, S. B. Davidson, G. G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. In *In H. V. Jagadish, I. S. Mumick (Eds.): Proceedings of the 1996 ACM SIGMOD Int. Conf. on Management of Data, Montreal, Canada*, 1996.

[6] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: a Graphical Language for Querying and Restructuring XML Documents. In *Proceedings of the Eight International World Wide Web Conference WWW8"*, Toronto, Canada, May, 1999.

[7] S. Comai. Graphical Query Languages for Semi-structured Information. PhD Thesis, Politecnico di Milano, 1999.

[8] S. Comai, E. Damiani, R. Posenato, and L. Tanca. A Schema-based Approach to Modeling and Querying WWW Data. In *Proceedings of Int. Conf. on Flexible Query Answering Systems*, FQAS'98, Roskilde, Denmark, May 13–15, 1998.

[9] M. P. Consens and A. O. Mendelzon. Graphlog: a Visual Formalism for Real Life Recursion. In Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Nashville, Tennessee, April 2–4, 1990.

[10] P. Cousot and R. Cousot. Abstract interpretation: a unified framework for static analysis of programs by construction of approximation of fixpoints. In *Proceedings of Fourth ACM POPL*, pp. 238–252, 1977.

[11] E. Damiani and L. Tanca. Semantic Approches to Structuring and Querying Web Sites. In *Proceedings of Seventh IFIP Work. Conf. on Database Semantics (DS-97)*, 1997.

[12] E. Damiani, L. Tanca. Blind Queries to XML Data. In *Proceedings of 11th International Conference, DEXA 2000*, Lecture Notes in Computer Science, Vol. 1873 pp. 345–356, 2000.

[13] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. In *Proceedings of QL'98 — The Query Languages Workshop*, December, 1998.

[14] A. Dovier and E. Quintarelli. Model-Checking Based Data Retrieval. Technical Report, Politecnico di Milano, May 2000. Available at www.sci.univr.it/∼quintare/Papers/DQ01-RR.ps.gz.

[15] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, Sept. 1997.

[16] G. Filè, R. Giacobazzi, and F. Ranzato, A Unifying View on Abstract Domain Design, *ACM Computing Surveys*, 28(2):333–336, 1996.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[18] R. Goldman and J. Widom. Dataguides: enabling querying formulation and optimization in semi-structured databases. In *VLDB'97, Proceedings of 23rd Int'l Conf. on Very Large Data Bases*, pp. 436–445, 1997.

[19] M. Gyssens, J. Paredaens, J. V. den Bussche, and D. Van Gucht. A graph-oriented object database model. *IEEE Transactions on Knowledge and Data Engineering*, August 1994.

[20] R. Heckel, and G. Engels. Graph Transformation and Visual Modeling Techniques. Bulletin of EATCS, 71:186–202, June 2000.

[21] P. C. Kanellakis and S. A. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Information and Computation*, 86(1):43–68, 1990.

[22] A. Lisitsa and V. Sazanov. Bounded Hyperset Theory and Web–like Data Bases. Research report, 97–21, DIMACS, 1997.

[23] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: a database management system for semistructured data. *SIGMOD Record*, 23(3):54–66, Sept. 1997.

[24] A. Mendelzon, G. Mihaila, and T. Milo. Querying the world wide web. In *Proceedings of Fourth Conference on Parallel and Distributed Information Systems*, Dec. 1996.

[25] R. Milner. An algebraic definition of simulation between programs. In Second International Joint Conference on Artificial Intelligence, pages 481–489. London, 1971.

[26] R. Milner. A Calculus of Communicating Systems. Lecture Notes in Computer Science, volume 92, Springer-Verlag, Berlin, 1980.

[27] R. Milner. Operational and Algebraic Semantics of Concurrent Processes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 19. Elsevier Science 1990.

[28] B. Oliboni, and L. Tanca. Querying XML specified WWW sites: links and recursion in XML-GL. In *Proceedings of Sixth International Conference on Rules and Objects in Databases*, pp. 1167–1181. London, July 2000.

[29] J. Paredaens, P. Peelman, and L. Tanca. G–Log: A Declarative Graphical Query Language. *IEEE Trans. on Knowledge and Data Eng.*, 7:436–453, 1995.

[30] D. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, Number 104 of *Lecture Notes in Computer Science*. Springer Verlag, 1980.

[31] P. Peelman. *G-Log: a deductive language for a graph–based data model*. PhD thesis, Antwerpen University, 1993.

[32] R. Paige and R. E. Tarjan. Three Partition Refinements Algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[33] J. van Benthem. Modal Correspondence Theory. PhD dissertation, Universiteit van Amsterdam, Instituut voor Logica en Grondslagenonderzoek van Exacte Wetenschappen, pages 1–148, 1978.