Operational and Abstract Semantics of a Query Language for Semi-Structured Information

Agostino Cortesi, Agostino Dovier, Elisa Quintarelli, and Letizia Tanca

Abstract

The paper revisits the semantics of G-log, a graph-oriented query language for semi-structured data. An operational semantics based on the notion of bisimulation is given both at the concrete level (instances) and at the abstract level (schemata). In this setting, some subtle ambiguities in the semantics of G-log queries can be successfully addressed.

Keywords: Databases, Semantics, Abstract Interpretation, World Wide Web.

1 Introduction

The increasing amount of information currently accessible through the Web presents new challenges to academic and industrial research on Data Bases. In this context, data are either structured, when coming from relational or object-oriented databases, or completely unstructured, when they consist of simple collections of text or image files. A number of research projects are currently addressing the problem of accessing in a uniform way this kind of semi-strucured data. Among these, we can cite LOREL [14], UnQL [4], WebSQL [15], WebOQL [2], StruQL [9], ADOOD [11].

Our contribution is part of the WG-log project [8, 5], which addresses the problem by extending the graph-based data-model G-log [18, 17], a query language based on a *data model for complex objects with identity* [1].

The *expressive power* of a query language is a measure of the number of queries that can be expressed by a program in that language. The *modeling power* is a perhaps more intuitive criterion and indicates how easily a query can be expressed in the language: it depends, of course, on the degree of correspondence between the objects of the real world and their representations in the database. The development of the language G-log aimed at increasing both these parameters for representing and querying complex data. Its recent extension WG-log [8] is particularly suited for semistructured data distributed over the Web.

The use of graphs for representing database schemata is common in the history of Database theory: recall, for instance, the *entity-relationship model* [3], the *semantic*

First author is with Dip. di Matematica Applicata e Informatica, Università Ca' Foscari, Via Torino 155, 30173 Venezia – Mestre (Italy), `cortesi@dsi.unive.it`

Other authors are with Dip. Scientifico-Tecnologico, Università di Verona, Strada Le Grazie 3, 37134 Verona (Italy), `(dovier|quint|tanca)@sci.univr.it`

The work has been partially supported by the MURST projects *Metodologie e tecnologie per la gestione di dati e processi su reti Internet e Intranet*, and *Tecniche formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software*, and by the CNR *Progetto Coordinato MIDA*.

networks, the various graphical representations of object-oriented schemata like Good [12], and Graphlog [6], just to name a few.

Most of the models and languages for the representation and querying of semistructured information cited so far share an analytical approach to data representation, lacking a synthetic notion of schema. Conversely, both G–Log and WG–Log focus on a concept very close to that of *database schema* in order to model semistructured information.

The main aim of this paper is to revisit the semantics of G-log in order to clarify some subtle ambiguities in the semantics of G-log queries. An operational semantics based on the notion of bisimulation [16, 13] is given both at the concrete level (instances) and at the abstract level (schemata). The relation between instances and schemata is clarified using Abstract Interpretation [7], that provides a systematic approach to guarantee the correctness of operating on schemata with respect to the corresponding concrete computations on instances. A refinement of the concrete semantics is introduced in order to deal with queries expressing negative constraints.

The results presented here for G–log can be easily estended to WG–log as well. As schemata can evolve gracefully with the evolution of their instances, in the extended setting of WG–log this will allow to trace the dynamical evolution of Web sites by keeping trace of the history of their schemata.

Similarities can be found between our approach and previous works on UnQL [4], where the notion of simulation is used for investigating query decomposition. However, differences between G-Log and UnQL are quite deep. For instance, at a syntactical level, we allow information to be located in nodes, and, more importantly, G-log queries are written directly in the graph formalism. Moreover, G-log allows to express cyclic queries and information, too.

The structure of the paper is as follows. Section 2 introduces the language G–log by means of intuitive examples. The syntax and the concrete semantics of the language are discussed in Section 3. Section 4 introduces abstract graphs (corresponding to schemata) and studies how they represent sets of concrete graphs (instances). Refinements of the concrete semantics are presented in Section 5 in order to deal with negation. Finally, Section 6 contains our Conclusions.

2 The language G-log: 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 outline 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 logic formulas is almost straightforward, as illustrated in Fig. 1. As observed in [18], G-log offers the expressive power of logic, the

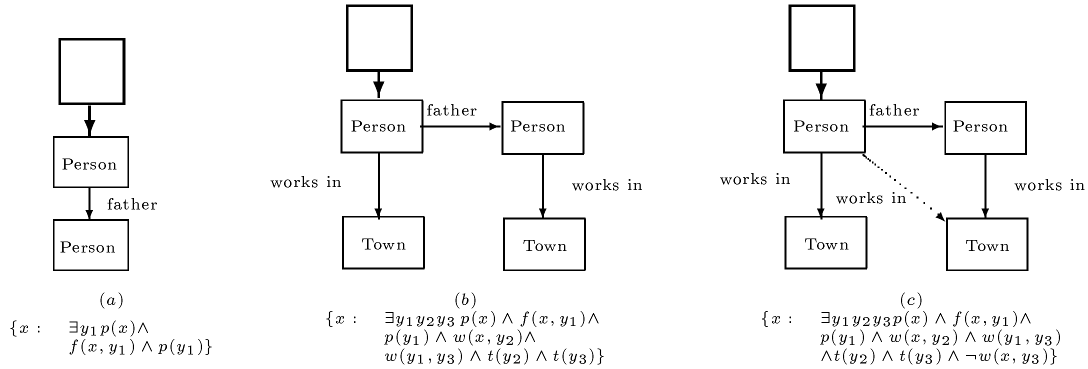


Figure 1: Sample queries

modelling power of object-oriented DBs, and the representation power of graphs.

However, the modelling power of G-log is heavily constrained by some arguable choices in its semantics [18]. 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;
- collect the people having at least two (distinct) children.

The semantics of G-log 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 [18]).

An even deeper problem arises when considering query (e). The semantics of G-log intepprets it exactly 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

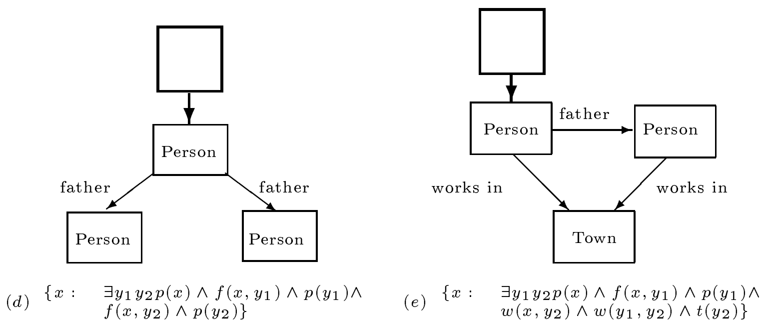


Figure 2: Problematical queries

defined in this way, in order to clarify the relationship between concrete (instances) and abstract (schemata) data representations.

3 Syntax and Concrete Semantics

In this section we introduce the basic aspects of the syntax of the G-log language. Definitions are based on the concept of directed labelled multigraph, and, differently from [18, 17], 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. Sect. 3.1) will be given in such a way that the practical use is coherent with that of [18, 17].

Definition 3.1 *A* G-log graph *is a directed labelled (multi)graph* $\langle N, E, \ell\rangle$*, where* N *is a set of nodes,* E *is a set of labelled edges of the form* $\langle m, \mathit{label}, n\rangle$*, where* $m, n \in N$ *and* label *is a pair of* $\mathcal{C} \times (\mathcal{L} \cup \{\perp\})$*, while* $\ell : N \longrightarrow (\mathcal{T} \cup \{\perp\}) \times \mathcal{C} \times (\mathcal{L} \cup \{\perp\}) \times (\mathcal{S} \cup \{\perp\})$*.* $\perp$ *means 'undefined', and:*

- $\mathcal{T} = \{\ \mathit{entity}, \mathit{slot}\ \}$ *is a set of types for nodes;*
- $\mathcal{C} = \{\ \mathit{red}, \mathit{green}, \mathit{red\ dashed}, \mathit{black}\ \}$ *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* $\ell_{\mathcal{T}}, \ell_{\mathcal{C}}, \ell_{\mathcal{L}}, \ell_{\mathcal{S}}$*. Moreover, 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:*

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

Observe that it is allowed that more edges connect two nodes, provided that their labels be different. Thus, it is not correct to view E simply as a subset of N^2 and ℓ as a function from $N \cup E$ into the domain of 4-tuples defined. Removing labels, E is in fact a multiset.

Red (RS) and *black* edges and nodes are graphically represented by thin lines (this does not originate confusion, since there cannot be red and black nodes and edges in the same graph), *green* (GS) by thick lines, and *red dashed* (RD) by dashed (dotted) thin 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 role 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. If an entity node is labelled by *result*[1] it will be simply represented by small

[1] [8] uses *entry point* nodes for this purpose.

squares, and its oucoming edges implicitly labelled 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

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

Definition 3.2 *A G-log graph* $G = \langle N, E, \ell\rangle$ *is a* labelled subgraph *of a G-log graph* $G' = \langle N', E', \ell'\rangle$ *, denoted* $G \sqsubseteq G'$*, if* $N \subseteq N'$*,* $E \subseteq E'$*, and* $\ell = \ell'|_N$*.*

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 $\top \equiv G$, $\perp \equiv \varepsilon$. Moreover, given two G-log graphs $G_1 = \langle N_1, E_1, \ell_1\rangle \subseteq G$ and $G_2 = \langle N_2, E_2, \ell_2\rangle \subseteq G$, their l.u.b. and g.l.b. can be simply computed as follows:[2]

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

Definition 3.3 *Given a G-log graph* $G = \langle N, E, \ell\rangle$*, if* $C \subseteq \{RS, RD, GS\}$*, let* $N' = N \cap \ell_{\mathcal{C}}^{-1}(C)$ *and* $E' = \{\langle m, \langle c, k\rangle, n\rangle \in E : c \in C\}$*. We define* $G|_C = \langle N', E', \ell|_{N'}\rangle$*.*

We define the notion of concrete graph (which will take in particular the role of instance of a DB or a WWW site), of (concrete) rule and program.

Definition 3.4 *A* G-log concrete graph *is a G-log graph such that:*

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

2. $(\forall n \in N)(\ell_{\mathcal{T}}(n) = \mathit{slot} \to \ell_{\mathcal{S}}(n) \neq \perp)$*.*

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

Definition 3.5 *A* G-log rule $R = \langle N, E, \ell\rangle$ *is a G-log graph such that:*

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

[2] As a side remark, notice that, if G is the (complete) graph $\langle N, N \times \{\perp\} \times N, \ell\rangle$ 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.

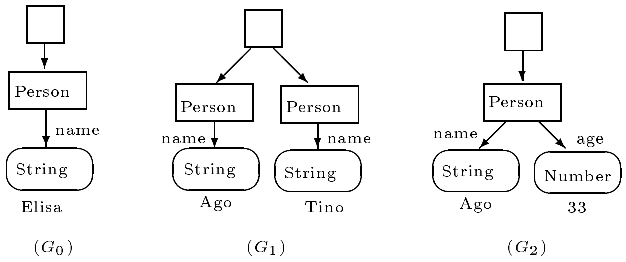


Figure 3: Examples of bisimulations

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

Definition 3.6 *A* G-log program *(or* query*) is a finite list of sets of G-log rules.*

We adapt the well-known concept of bisimulation [19, 16] to our aims:

Definition 3.7 *Given two G-log graphs* $G_0 = \langle N_0, E_0, \ell^0 \rangle$ *and* $G_1 = \langle N_1, E_1, \ell^1 \rangle$, $b \subseteq N_0 \times N_1$ *is said to be a* bisimulation *between* G_0 *and* G_1 *iff:*

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

2. *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, $(\forall n_0 \in N_0)(\forall n_1 \in N_i)$ *such that* $n_0 \ b \ n_1$, *for* $i = 0, 1$ *it holds that*
   $$(\forall \langle m_i, \ell_i \rangle \in M_i(n_i))(\exists \langle m_{1-i}, \ell_{1-i} \rangle \in M_{1-i}(n_{1-i}))(m_0 \ b \ m_1 \wedge \ell_i = \ell_{1-i}\rangle) .$$

We write $G \overset{b}{\sim} G'$ ($G \overset{b}{\not\sim} G'$) *if* b *is (not) a bisimulation between* G *and* G'. *We write* $G \sim G'$ ($G \not\sim G'$) *if there is (not) a bisimulation between* G *and* G'.

It is easy to see that $\sim$ is an equivalence relation. Thus, it makes sense to deal with $\mathcal{G}^c / \sim$.

Notice that colors ($\ell_{\mathcal{C}}$) are not taken into account in the bisimulation search, whilst the *value* fields of the label are considered only when they are defined. This last feature will allow to find bisimulations between *schemata* and *instances*.

As an instance, consider the graphs in Fig. 3: it holds that $G_0 \sim G_1$, while $G_i \not\sim G_2$, for $i = 0, 1$.

3.1 Concrete Semantics

We use the concept of bisimulation to reformulate the semantics of G-log programs, given in [18] using the *ad hoc* concept of embedding of a graph. Initially, we assume that no red dashed edges and nodes occur in rules (no negation). The extension to graphs with negation is the subject of Sect. 5.

Definition 3.8 *Let* G *be a concrete graph and* R *a rule.* R is applicable in G *iff* $(\exists G_1 \sqsubseteq G)(G_1 {\sim} R_{\{RS\}})$. G verifies R *(*$R \models G$*) iff*

$$(\forall G_1 \sqsubseteq G) \left(\begin{array}{l} (G_1 \sim R_{\{RS\}}) \rightarrow \\ (\exists G_2 \sqsubseteq G) \left(\begin{array}{l} G_1 \sqsubseteq G_2 \wedge G_2 \sim R_{\{RS,GS\}} \wedge \\ (\forall G_3 \sqsubseteq G_2)(G_1 \sqsubseteq G_3 \wedge G_3 \sim R_{\{RS\}} \rightarrow G_3 = G_1) \end{array} \right) \end{array} \right) .$$

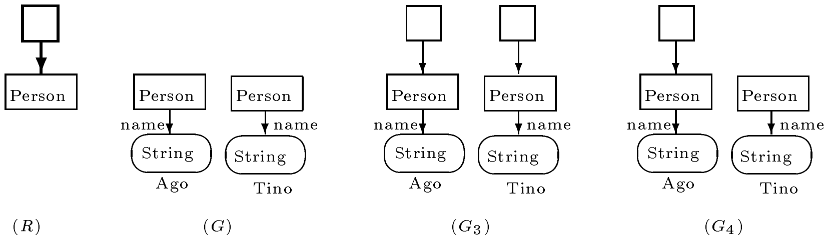


Figure 4: A rule and three concrete graphs

Intuitively, G verifies R if for any subgraph G_1 of G matching (w.r.t. bisimulation) with the red (pre-condition) part of the rule, there is a way to 'complete' G_1 in a graph $G_2 \sqsubseteq G$ such that the whole rule R matches with G_2. The further requirement: $(\forall G_3 \sqsubseteq G_2)\dots$ is necessary to avoid the possibility of using other parts of G, matching with $R_{\{RS\}}$ independently, to extend G_1 (see the example after Def. 3.9).

The notion of applicability is a sort of pre-condition for an effective application of a rule to a concrete graph, whose precise semantics is given below:

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

1. $G \sqsubseteq G'$, $R \models G'$, *and*

2. G' *is minimal w.r.t. property (1), namely there is no graph* G'' *such that* $G \sqsubseteq G''$, $G'' \sqsubset G'$, $R \models G''$.

Intuitively, a rule, if applicable, extends G in such a way that G verifies R. Moreover, it is required that the extension is minimal. If R is not applicable in G, then $\langle G, G \rangle \in [\![R]\!]$, i.e. there is no 'effect'. [3]

For example, consider graphs (R), (G), (G_3), and (G_4) of Fig. 4 and (G_1) of Fig. 3. It holds that $\langle G, G_1 \rangle$ and $\langle G, G_3 \rangle$ belong to $[\![R]\!]$. $\langle G, G_4 \rangle \notin [\![R]\!]$ since $R \not\models G_4$: this is due to the condition $(\forall G_3 \sqsubseteq \dots)$ in the Definition 3.8. Notice that $G_1 \sim G_3 \sim G_4$.

Moreover, observe that the effect of a rule (viewed as a 'query') is to find all possible solutions: in other words, the semantics is set-oriented.

In general, $[\![\cdot]\!]$ is not a function, not even modulo bisimulation (apply the rule R to the concrete schema G deprived of the *slot* node labelled by *Tino* and its entering edge).

The problem here is the generation of more than one node to collect objects having a different degree of specification. This corresponds to the problem of *invented values* in complex object databases with identity [1]. It can be easily proved that:

Theorem 3.10 *Let R be a rule having green edges but no green nodes, and G a concrete graph. Then, $[\![R]\!]$ applied to G returns graphs all equivalent in $\mathcal{G}^c/\sim$.*

The effect of a rule of the form above is that of "adding" edges having the same types and labels of those in the green part of R to the graph G. To extend the result of Theorem 3.10 to all rules, a viable approach is to make the further assumption in the

[3] The injective embedding used in [17] to give a semantics of G-log is a particular case of bisimulation. It is possible to check that viewing G as 'source graph' and G' as 'target graph', the two definitions are equivalent.

semantics of a rule that for each *green* node of R at most one corresponding node is introduced in G'.

Rules can be combined to build programs as shown in Def. 3.6. Their semantics are the following:

Definition 3.11 *Let S be a set of rules $\{R_1, \ldots, R_n\}$. Then $\langle G, G' \rangle \in [\![S]\!]$ if*

1. *$G \sqsubseteq G'$, $R_i \models G'$, for $i = 1, \ldots, n$, and*
2. *G' is minimal w.r.t. property (1).*

Let P be a program $\langle S_1, \ldots, S_n \rangle$. $\langle G_0, G_n \rangle \in [\![P]\!]$ iff there are $G_1, \ldots, G_{n-1}$ such that $\langle G_i, G_{i+1} \rangle \in [\![S_{i+1}]\!]$, for $i = 0, \ldots, n-1$.

Due to lack of space, the programs analyzed in this paper are simply formed by a single rule. The full power of programming with graphs is well explained in [17].

4 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 [7, 10], 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.

Definition 4.1 *A* (G-log) abstract graph *is a G-log graph such that:*

1. $(\forall x \in N \cup E)(\ell_{\mathcal{C}}(x) = \mathit{black})$,
2. $(\forall x \in N \cup E)(\ell_{\mathcal{S}}(x) = \perp)$, *i.e., an abstract graph has no values.*

With $\mathcal{G}^{\mathcal{A}}$ we denote the set of G-log abstract graphs.

Let us use once more the notion of bisimulation to re-formulate the G-log concepts of instance and schema.

Definition 4.2 *A concrete graph I is* an instance *of an abstract graph G (G represents I) iff $(\exists I' \sqsupseteq I)(G \sim I')$. 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. 5 there is an example of application of the definition above. (S) represents (I). To build the witness (I'), add to (I) an edge labelled by *works* linking the entity node *Person* of *Bob* with the *entity* node *Town*. Moreover, add edges labelled by *lives* from the two nodes labelled *Person* to the node labelled *Town*. It is easy to check that a bisimulation from S to I' is uniquely determined.

The notions of *applicability* and *verifiability* for abstract graphs are the same as in Def. 3.8. This also holds for the operational semantics definitions for rules and programs.

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

Lemma 4.3 *(a) If I is an instance of G with witness I', then forall I'' s.t. $I \sqsubseteq I'' \sqsubseteq I'$ it holds that I'' is an instance of G.*

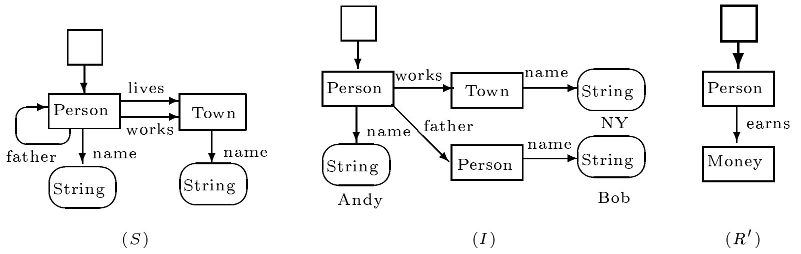


Figure 5: A schema, and instance, and a rule

(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'.

The next theorem states that given a set of instances there is always a schema that represents all of them and is minimal (modulo bisimulation).

Theorem 4.4 *(Abstraction) Let S be a set of concrete graphs. Then there is an abstract graph G such that I is an instance of G for all $I \in S$ and for any G' fulfilling such a property there is $G'' \sqsupseteq G$ such that $G'' \sim G'$.*

Given a set S of concrete graphs, let α be the abstraction function giving G as in Theorem 4.4 (modulo bisimulation).

Observe that if S is a singleton, i.e. $S = \{I\}$, then I is an instance of the abstract graph $\alpha(\{I\})$ obtained from I by turning every value (i.e., results of $\ell_{\mathcal{S}}$) associated to its nodes and its edges to $\perp$. In general, through the function α we get the best schema for a family of instances.

A Galois insertion ([7]) between $\mathcal{G}^{\mathcal{A}}$ and $\wp(\mathcal{G}^{\mathcal{C}})$ can be obtained by considering the abstraction function α defined above and the adjoint concretization function $\gamma : \mathcal{G}^{\mathcal{A}}_{/\sim} \longrightarrow \wp(\mathcal{G}^{\mathcal{C}})$:

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

Lemma 4.5 *Function γ is monotonic, i.e. for any pair of abstract graphs G, G', $G \sqsubseteq G'$ implies $\gamma(G) \subseteq \gamma(G')$.*

Lemma 4.6 *Function γ is injective, i.e. for any pair of abstract graphs G, G', $G \not\sim G'$ implies $\gamma(G) \neq \gamma(G')$.*

Theorem 4.7 *(Correctness) Let G, G' be abstract graphs and R a rule without slots such that $\langle G, G' \rangle \in [\![R]\!]$. If $I \in \gamma(G)$ and $\langle I, I' \rangle \in [\![R]\!]$, then $I' \in \gamma(G')$, i.e., the following diagram commutes:*

$$\begin{array}{ccc} G & \stackrel{R}{\to} & G' \\ \downarrow \gamma & & \downarrow \gamma \\ I & \stackrel{R}{\to} & I' \end{array}$$

Theorem 4.7 guarantees the correctness of abstract computations: the application of a rule to an abstract graph safely represents the application of the same 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. 5. Since (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.

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.[4]

5 Negation

Let us turn to extend the semantics in order to deal with rules and programs with negation (i.e., containing red dashed nodes and edges). We assume the following syntactical restriction on rules with negation: if an egde $\langle m, n\rangle \in R|_{\{RD\}}$, then

- $\ell_{\mathcal{C}}(m) = \ell_{\mathcal{C}}(n) = RS$, or
- $\ell_{\mathcal{C}}(m) = RS$, $\ell_{\mathcal{C}}(n) = RD$, and any $G_1 \sqsubseteq G$ rooted by n is a subgraph of $R|_{\{RD\}}$.

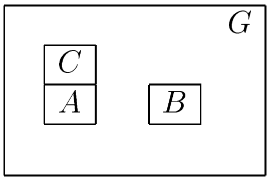
We start by giving the notion of applicability of a rule containing negation, thus completing the Def. 3.8:

Definition 5.1 *Let G be a concrete graph and R a rule such that $R|_{\{RD\}} \neq \emptyset$ (i.e., R has a red dashed part). $G_1 \sqsubseteq G$* meets *R* in *G iff the following conditions hold:*

$$\begin{array}{ll} (a) & G_1 \sim R|_{\{RS\}} \\ (b) & (\forall G_2 \sqsubseteq G)(G_1 \sqsubset G_2 \not\sim R|_{\{RS,RD\}}) \\ (c) & (\forall G_3 \sqsubset G_1)(G_3 \sim R|_{\{RS\}} \Rightarrow G_3 \text{ meets } R \text{ in } G) \end{array}$$

Observe that, since G_3 is strictly included in G_1, then the notion is well-defined. Moreover, R is applicable in *G iff $(\exists G_1 \sqsubseteq G)$ s.t. $G_1 \sqsubseteq G$ meets R in G.*

Some words are needed to justify the complexity of the formula above. As for the case of absence of negation, the notion of applicability is aimed at identifying the possibility of a meaningful application of a rule R on a concrete schema. This concept has been defined here by using an auxiliary concept (meet). In order to meet a rule, a subgraph G_1 of G must match the red solid part of the rule (this is expressed by condition (a)). Moreover, condition (b) guarantees that G_1 cannot be 'extended' inside G to match with the negative part of the query. Unfortunately, this is not sufficient. The third (inductive) requirement is introduced to avoid situations of the form:



Assume that:
$G_1 = A \cup B, G_1 \sim R_{\{RS\}}, A \sim R_{\{RS\}}, B \sim R_{\{RS\}}$,
$A \cup C \sim R_{\{RS,RD\}}$, and $A \cup B \cup C \not\sim R_{\{RS,RD\}}$.

[4]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.

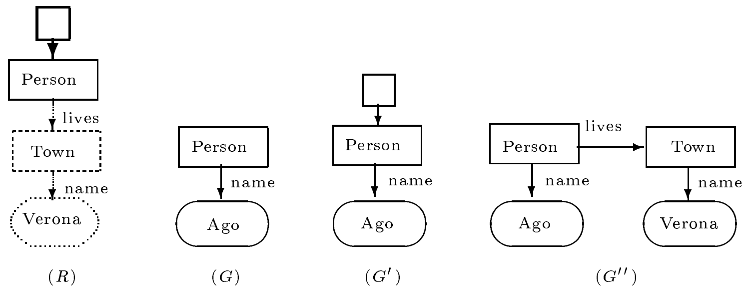


Figure 6: Negation as failure

Although $G_1 = A \cup B$ seems to meet the rule, it hides a subgraph, A, that can be extended to C in order to falsify the preconditions (positive and negative) given by R.

Let us see how to use this concepts to complete the notion of *verify* (Def. 3.8) in the presence of negation.

Definition 5.2 *Let G be a concrete graph and R a rule such that $R|_{\{RD\}} \neq \emptyset$. G* verifies *R ($R \models G$) iff*

- $(\forall G_1 \sqsubseteq G)(G_1 \not\sim R|_{\{RS,RD,GS\}})$, *and*
- $(\forall G_1 \sqsubseteq G)$ *s.t. G_1 meets R in G:*

$$\begin{array}{ll}(\exists G_2 \sqsubseteq G) & (G_1 \sqsubseteq G_2 \wedge G_2 \sim R|_{\{RS,GS\}} \wedge \\ & (\forall G_3 \sqsubseteq G_2)(G_1 \sqsubseteq G_3 \wedge G_3 \sim R|_{\{RS,GS\}} \rightarrow G_3 = G_1)\end{array}$$

The first requirement ensures that there is no subgraph of G in contradiction with the negative request of the rule. The second is, in fact, the same as that used in the Def. 3.8 for the simplest case of absence of negation.

The semantics of the application of a rule R to a concrete graph G is, intuitively, that of finding the least $G' \sqsupseteq G$ s.t. G' verifies R. The presence of negation requires further considerations. Let R and G be as in Fig. 6. The graphs G and G' both verify R. 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 can't derive that this fact is true.
- G'' is obtained by adding the hypothesis '*Ago* lives in *Verona*' that ensures that no subset of G meets R.

The choice of G-log is the first one, and it is reflected in the condition 2 of the following definition:

Definition 5.3 *Let R be a rule. Then $\langle G, G' \rangle \in [\![R]\!]$ if and only if:*

1. *$G \sqsubseteq G'$, $R \models G'$,*
2. *for all $G_1 \sqsubseteq G$: G_1 meets R in G iff G_1 meets R in G';*
3. *G' is minimal w.r.t. properties (1) and (2).*

The extension of the abstract semantics to deal with negation is currently under development.

6 Conclusions

The contribution of this paper to the semantics of G-log is part of a larger project, where computational and logical aspects ask to be integrated in order to result in a solid and usable query-language for semi-structured data. As already said in the Introduction, our contribution is expected to have a positive impact on WG-log as well. This is object of our current (and future) efforts. Moreover, we plan to get further refinements of our semantics in order to relax some restrictions and thus increase the expressive power of the language.

Acknowledgements

We wish to thank S. Comai, E. Damiani, and R. Posenato for stimulating discussions, and the anonymous referees for their suggestions.

References

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

[2] G. Arocena and A. Mendelzon. In *WebOQL: Restructuring documents, databases, and webs*, 1998.

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

[4] 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.): Proc. of the 1996 ACM SIGMOD Int. Conf. on Management of Data, Montreal, Canada*, 1996.

[5] S. Comai, E. Damiani, R. Posenato, and L. Tanca. A Schema-based Approach to Modeling and Querying WWW Data. Research report, University of Verona, 1997.

[6] M. P. Consens and A. O. Mendelzon. The G+// graphlog visual query system. *SIGMOD Record (ACM Special Interest Group Management of Data)*, June 1990.

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

[8] E. Damiani and L. Tanca. Semantic Approches to Structuring and Querying Web Sites. In *Procedings of 7th IFIP Work. Conf. on Database Semantics (DS-97)*, 1997.

[9] 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.

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

[11] F. Giannotti, G. Manco, and D. Pedreschi. A Deductive Data Model for Representing and Querying Semistrucured Data. In *APPIA-GULP-PRODE'97. Joint Conference on Declarative Programming*, pp. 129–139, 1997.

[12] 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.

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

[14] 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.

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

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

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

[18] 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.

[19] D. Park. Concurrency and automata on infinite sequences. In *LNCS 104*. Springer Verlag, 1980.