ISSN 1884-0760

GRACE TECHNICAL REPORTS

Simulation-based Graph Schema for View Updatability Checking of Graph Queries

Keisuke Nakano Soichiro Hidaka Zhenjiang Hu
Kazuhiro Inaba Hiroyuki Kato

GRACE-TR 2011–01 May 2011

CENTER FOR GLOBAL RESEARCH IN
ADVANCED SOFTWARE SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF INFORMATICS
2-1-2 HITOTSUBASHI, CHIYODA-KU, TOKYO, JAPAN

WWW page: http://grace-center.jp/

The GRACE technical reports are published as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, notwithstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may not be reposted without the explicit permission of the copyright holder.

Simulation-based Graph Schema for View Updatability Checking of Graph Queries

Keisuke Nakano
The University of Electro-Communications
1-5-1 Chofugaoka, Chofu-shi
Tokyo 182-8585, Japan
ksk@cs.uec.ac.jp

Soichiro Hidaka Zhenjiang Hu Kazuhiro Inaba Hiroyuki Kato
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku
Tokyo 101-8430, Japan
{hidaka,hu,kinaba,kato}@nii.ac.jp

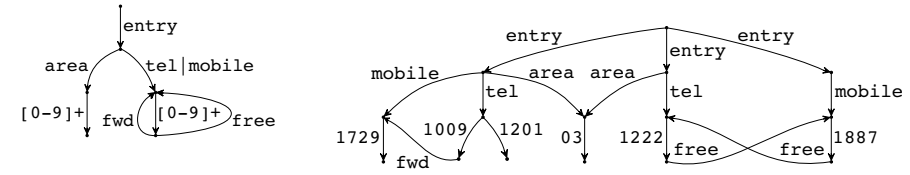
May 30, 2011

Abstract

View updating problem is concerned with translating a view update into a corresponding update against the base data source. In our previous work, we solve the view updating problem in which both sources and views are represented by graph-structured data for general purposes. Since the solution is based on a sort of program inversion techniques, it often requires expensive computation to find the translation of view updating. The problem is that the expensive computation may be in vain when the updated view is invalid in the sense that either there is no candidate of corresponding sources or the corresponding source does not conform to user's intention. In this paper, we present a method for checking view updatability in order to know whether the updated view is valid or not before computing the corresponding sources. To achieve a simple computation of view updatability checking, we introduce a new graph schema whose conformance is defined by *graph simulation*. Although the idea of our schema comes from the simulation-based graph schema proposed by Buneman et al., our schema can describe necessity of out-going edges, which was impossible in their schema. This improvement helps us to give a precise solution for view updatability checking.

1 Introduction

View updating problem is concerned with translating a view update into a corresponding update against a base data source. The problem has been extensively studied for decades, but it remains as a big challenge [2, 9, 11, 10, 17, 3], particularly when we want to treat graph structured data instead of relational database. It is, however, natural to adopt graph structured data to represent compound information in which their components are related in a complicated manner. For instance, in model-driven software



(a) UnCAL schema S_1 (b) Graph data G_{pb}

Figure 1: UnCAL graph schema of Buneman et al. and graph data

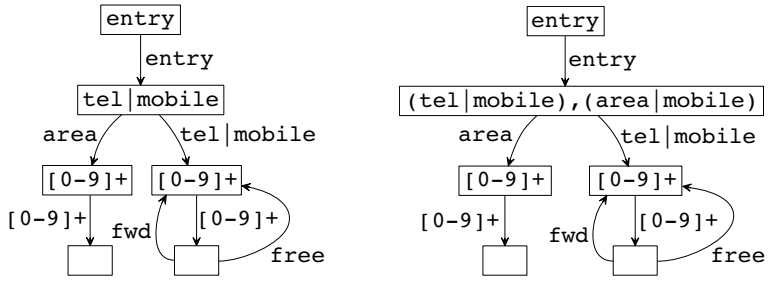
development, a relation between software components forms a graph as in UML [1] and MOF [20]. View updating on graph structures assists a *bidirectional* software development in which we can consistently manage multiple components.

Our previous work [13] gave a solution to the view updating problem in which both sources and views are represented by graph structures. In our solution, we employed a graph query language UnCAL [6] to describe view definitions which are transformations (queries) over graph structured data. One of the main advantages of UnCAL is its solid semantics foundation and efficient implementation. We do not have to care either termination or evaluation order even if an input graph data has cycle and shared nodes. Another advantage of UnCAL is that it provides a user-friendly interface language UnQL in which one can use the select-where syntax like SQL to query to graph data with regular path patterns. We solved the view updating problem for UnCAL by using trace information and an automatic inversion technique [13].

One limitation of the current view updating for UnCAL is its inefficieny in dealing with *invalid* view updates; it tries to propagate any view updates to the source. Since view updating often requires expensive computation, the attempt may fail and be in vain. Some updated view may have no corresponding updates. Some updated view may induce undesirable updates. These invalid view updates had better be detected before running the procedure.

In this paper, we propose a method for checking whether view updatability is valid or not before running view update translation. Our idea is to reduce the view updatability checking problem to computation of the range of a graph query followed by validation whether the updated view is in this range. Our contributions are two-fold. First, we introduce new graph schema, called *VU-schema*, to specify a set of graph data of desirable sources. Second, we show how to compute the range of a given UnCAL query for the source graphs conforming to such schema. Since the computed range is also represented by an analogous form to our schema, we can check view updatability by inspecting if an updated view conforms to the range.

More concrerely, our graph schema is *simulation-based*, where the schema conformance is checked through the existence of a graph simulation. The idea has been proposed by Buneman et al. [4] who themselves invented the graph algebra UnCAL. Figure 1 shows their graph schema S_1, which we call *UnCAL schema*, and a graph G_{pb} conforming to S_1, which represents a part of some phonebook data. The UnCAL schema is edge-labeled, similarly to UnCAL graph data. The difference between schemas and graphs is that edge labels in a schema are *predicates* on edge labels in graph data. In the schema S_1, regular languages are used as the predicates: for instance, `[0-9]+` is satisfied by any sequence of numeral characters and `tel|mobile` is satisfied by both `tel` and `mobile`. The schema conformance is checked by finding



(a) VU-schema S_2 (b) VU-schema S_3

Figure 2: Examples of VU-schemas

a simulation, that is, informally, a relation from nodes in the data to those in the schema with checking the predicates on their outgoing edges.

The problem of the UnCAL schema is that they cannot force existence of some specific edges. For example, suppose that G_{pb} does not have the edge labeled with `03`. The graph still conforms to S_1 even though it is strange as a database for phone books. In extreme case, even the graph obtained by removing all edges except `entry` edges conforms to S_1. The problem has already been pointed out in [4], which is caused by the unidirectionality of the simulation that is a basis of their schema conformance. It is not desirable to use the UnCAL schema in the context of view updating.

Our graph schema, VU-schema, overcomes the weakness of the UnCAL schema so that it can enforce the existence of specific edges. VU-schema is represented by a graph which has labels at not only edges but also nodes. Figure 2 shows two examples S_2 and S_3 of the VU-schemas. While each every label is a predicate like that of UnCAL schemas, every node label is a finite set of predicates each of which requires the existence of an outgoing edge satisfying it in data graphs. The schema conformance of a data graph against a VU-schema is checked by finding a simulation between the data graph and the schema after assigning a set of labels of its outgoing edges to each node in the data graph. For example, the second node from the top in the VU-schema S_2 has a single predicate `tel|mobile` as its label. It requires for the corresponding node to have an outgoing edge labeled with either `tel` or `mobile`, while the node is not required to have an `area` edge. On the other hand, the second node from the top in the VU-schema S_3 has a set of two predicates `(tel|mobile)` and `(area|mobile)`. It requires for the corresponding node to have outgoing edges, one of which is labeled with either `tel` or `mobile` and another of which is labeled with either `area` or `mobile`. Therefore, the node must have both `tel` and `area` edges when it has no `mobile` edge. The UnCAL schema of Buneman's et al. can be regarded as a special one of VU-schema in which every node is labeled with an empty set of predicates. Note that all the schemas S_1, S_2 and S_3 have the same graph structure. In VU-schemas, we can represent various sorts of constraints by varying a set of predicates at the node label.

The rest of this paper is organized as follows. After explaining the graph models of UnCAL in Section 2, including the basic concepts of graph simulation and a known algorithm for finding a simulation, we briefly review UnCAL, the graph algebra being used to write graph queries whose updatability is to be checked, in Section 3. Then, we move to our framework, introducing a new graph schema, VU-schema, for describing

richer constraints that can enforce existence of specific edges in Section 4, and showing how the range of a graph query can be efficiently computed in Section 5. Finally, we discuss the related work in Section 6 and concludes the paper in Section 7.

2 UnCAL graphs and simulation

We start with introducing the notion of edge-labeled graphs, called *UnCAL graph* [6]. In this section, we present the UnCAL graphs and a simulation on them. Throughout the paper, let $\mathcal{L}$ denote a finite set of edge labels in data graphs and let $\mathcal{M}$ denote a set of markers used for connecting UnCAL graphs. Every marker is written as a symbol starting with $\&$ and the set $\mathcal{M}$ forms a monoid $(\&, \cdot)$ with a unit element $\&$ which is called *default marker*.

2.1 UnCAL graphs

An UnCAL graph is a quadruple (V, E, I, O) where V is a set of nodes, $E \subseteq V \times (\mathcal{L} \cup \{\varepsilon\}) \times V$ is a set of edges, I is a partial function from $\mathcal{M}$ to V, and $O \subseteq V \times \mathcal{M}$. When $v = I(\&x)$ is defined for some $\&x \in \mathcal{M}$, we call v *input node*; When $(v, \&y) \in O$ for $v \in V$, we call v *output node*. These nodes play an important role to synthesize graphs as shown later. Intuitively, an input node is seen as a root node of graphs and an output node is used to connect with an input node of the same marker. Every UnCAL graph has at least one input node. We write $u \xrightarrow{a} v$ for an edge (u, a, v). For $a \in \mathcal{L}$, $E[a]$ denotes a subset of E whose edge has label a. An edge labeled with the special symbol ε, called ε edge, behaves like empty transition in automata theory and is written as $u \dashrightarrow v$. We write $u \overset{\varepsilon*}{\dashrightarrow} v \in E$ when there exist edges $u \dashrightarrow v_1$, $v_1 \dashrightarrow v_n$, and $v_n \dashrightarrow v$ in E for some $v_1, \ldots, v_n$. Similarly, we write $u \xrightarrow{\varepsilon*.a} v$ when $u \overset{\varepsilon*}{\dashrightarrow} t$ and $t \xrightarrow{a} v$ with some t.

2.2 Extended simulation and bisimulation

The notion of simulation on UnCAL graphs is defined as *extended simulation* which is aware of markers and ε edges. Let $G_1 = (V_1, E_1, I_1, O_1)$ and $G_2 = (V_2, E_2, I_2, O_2)$ be UnCAL graphs. A relation $\prec \subseteq V_1 \times V_2$ is called extended simulation if it satisfies all the following conditions:

(a) if $u_1 \prec u_2$ and $u_1 \xrightarrow{\varepsilon*.a} v_1$, then there exists $v_2 \in V_2$ such that $v_1 \prec v_2$ and $u_2 \xrightarrow{\varepsilon*.a} v_2$;

(b) if $u_1 \prec u_2$ and $u_1 = I_1(\&x)$, then $u_2 = I_2(\&x)$;

(c) if $u_1 \prec u_2$, $u_1 \overset{\varepsilon*}{\dashrightarrow} v_1$ and $(u_1, \&y) \in O_1$, then $u_2 \overset{\varepsilon*}{\dashrightarrow} v_2$ and $(u_2, \&y) \in O_2$;

(d) if $I_1(\&x)$ is defined, then $I_1(\&x) \prec I_2(\&x)$.

We write $G_1 \prec G_2$ when there exists an extended simulation between their nodes.

Henzinger et al. proposes a procedure to find a simulation between nodes of graphs [12]. Figure 3 shows the procedure which is specialized for UnCAL graphs to find an extended simulation. We use $pre_i(S)$ $(i = 1, 2)$ for a function which returns $\{u \xrightarrow{a} v \in E_i \mid a \in \mathcal{L},\ v \in S\}$ if S is a set of nodes and $\{u \in V_i \mid u \xrightarrow{a} v \in S\}$ if S is a set of edges. The procedure assumes that input UnCAL graphs have no ε

input: two UnCAL graphs $G_1 = (V_1, E_1, I_1, O_1)$ and $G_2 = (V_2, E_2, I_2, O_2)$ with no ε edge
output: a partition Π of $V_1 \cup E_1$ and a simulating function $r : \Pi \to 2^{V_2} \cup 2^{E_2}$.

$\Pi := \{V_1\} \cup \{E_1[a] \mid a \in \mathcal{L}\}$.
$r(V_1) := V_2$.
$r(E_1[a]) := E_2[a]$ $(a \in \mathcal{L})$.
while there exists $P, Q \in \Pi$ such that $P \cap pre_1(Q) \neq \emptyset$ and $r(P) \not\subseteq pre_2(r(Q))$ **do**
let $(P', P'') = (P \cap pre_1(Q), P \setminus pre_1(Q))$ **in**
$\Pi := (\Pi \setminus \{P\}) \cup \{P'\}$
$r(P') := r(P) \cap pre_2(r(Q))$
if $P'' \neq \emptyset$ **then** $\Pi := \Pi \cup \{P''\}$; $r(P'') := r(P)$.

Figure 3: Procedure *FindSim* for a simulation

edge since all ε edges can be eliminated in a way similar to ε transition removal for automata. See [12] for further detail of the procedure itself. For partition Π and function r which are an output of the procedure, we define a simulation by $\{(v_1, v_2) \mid U \in \Pi,\ v_1 \in U \cap V_1,\ v_2 \in r(U)\}$. Although the procedure *FindSim* does not consider conditions (b),(c) and (d) on markers, they can be checked in the body of the **while** loop to find an extended simulation.

When both relation R and its inverse relation R^{-1} are extended simulations, we call R *extended bisimulation*. The reader may recall *weak bisimulation* [18] as an analogous definition of bisimulation of graphs containing empty transitions like ε edges of UnCAL graphs. Note that the notions of extended bisimulation and weak bisimulation are incomparable. The detailed comparison is found in [6]. In the rest of the paper, we say just simulation and bisimulation for extended simulation and extended bisimulation, respectively.

We say that a graph query f is *bisimulation-generic* when $f(G_1)$ and $f(G_2)$ are bisimilar for any bisimilar UnCAL graphs G_1 and G_2. Every graph queries represented in the graph algebra UnCAL is bisimulation-generic. UnCAL identifies graphs with any bisimilar graphs.

2.3 Finding a simulation over infinite graphs

Henzinger et al. originally presents the aforementioned procedure for finding a simulation over *infinite* graphs [12]. This is because that the termination of the procedure does not require finiteness of graphs. When we have a finite representation of infinite graphs with some conditions mentioned later, we can apply the procedure *FindSim* for infinite graphs. Following Henzinger et al., we introduce the notion of 'effectiveness' of graphs.

First let us define effectiveness of classes. For a set S, a class $\mathcal{C} \subseteq 2^S$ is called *effective representation over* S when

(1) $S \in \mathcal{C}$;

(2) $a \in T$ is decidable for any $a \in S$ and any $T \in \mathcal{C}$;

(3) $T = \emptyset$ is decidable for any $T \in \mathcal{C}$;

(4) $\mathcal{C}$ is effectively closed under boolean operations.

For instance, a set of all regular languages is an effective representation over a set of strings, and a set of finite number of intervals of real numbers is an effective representation over real numbers.

Now we define effectiveness of UnCAL graphs. We say that an UnCAL graph $G = (V, E, I, O)$ is *effective* when there exists an effective representation $\mathcal{C}$ over $V \cup E$ such that $V \in \mathcal{C}$, $E[a] \in \mathcal{C}$ for all $a \in \mathcal{L}$, and $\mathcal{C}$ is effectively closed under the pre_i operation in the procedure *FindSim*. The procedure *FindSim* always terminates when both G_1 and G_2 are effective UnCAL graphs as shown in [12]. We can observe that it is not necessary for G_2 to be exactly effective. We introduce *pseudo-effective representation* for classes by replacing condition (4) into

(4)′ $\mathcal{C}$ is effectively closed under union and intersection, and $P \subseteq Q$ is decidable for $P, Q \in \mathcal{C}$.

Since it only requires decidability of subsumption instead of effective closedness under complement, we can claim that pseudo-effectiveness has a relaxed form of effectiveness. As we define *pseudo-effective graphs* in a way similar to effective graphs, the observation above leads the following theorem as shown in [12].

Theorem 2.1 *For an effective UnCAL graph G_1 and a pseudo-effective UnCAL graph G_2, the procedure FindSim terminates when a finite simulation exists between them.*

3 UnCAL: graph algebra

We present a graph algebra UnCAL [5, 6] used for specifying graph queries on UnCAL graphs. In this section, we introduce eight constructors for UnCAL graphs and syntax and semantics of UnCAL algebra.

Every UnCAL graph can be constructed by the following eight constructors:

$\{\}$ a single node graph with no edge;

$\{l : G\}$ a graph with a single input node whose out-going edge is labeled with l and points to a graph G;

$G_1 \cup G_2$ a graph consisting of input nodes each of which points to the corresponding input node of G_1 and G_2 with ε edges;

$\&x := G$ a graph obtained by replacing input markers $\&z$ in G with $\&x \cdot \&z$;

$\&y$ a graph of a single output node whose marker is $\&y$;

$G_1 \oplus G_2$ a graph consisting of two graphs G_1 and G_2 whose input markers are disjoint;

G_1 @ G_2 a graph obtained by gluing each output node in G_1 with the corresponding input node in G_2;

$\mathsf{cycle}(G)$ a graph obtained by connecting each output node in G with the corresponding input node in G using ε edges.

A graph algebra UnCAL is defined by adding two constructs, a conditional branch and structural recursion to the graph constructors above. The syntax of UnCAL is given by

$$
\begin{array}{rll}
M ::= & \{\} \mid \{L : M\} \mid M \cup M \mid \&x := M \mid \&y & \\
& \mid M \oplus M \mid M \mathbin{@} M \mid \mathsf{cycle}(M) & \text{(CONSTRUCTORS)} \\
& \mid \mathsf{if}\ P(L)\ \mathsf{then}\ M\ \mathsf{else}\ M & \text{(CONDITIONAL)} \\
& \mid \$g & \text{(GRAPH VARIABLE)} \\
& \mid \mathsf{rec}(\lambda\langle \$l, \$g\rangle.M)(M) & \text{(RECURSION)} \\
L ::= & \$l & \text{(LABEL VARIABLE)} \\
& \mid C & \text{(LABEL CONSTANT)} \\
& \mid L + L & \text{(LABEL CONCATENATION)}
\end{array}
$$

where M and L range over UnCAL expressions and label expressions, respectively. The symbol P ranges over effectively representable predicates on $\mathcal{L}$. For example, it includes the predicate $p_{=\mathtt{a}}$ that "whether the label equals to a" and the predicate $p_{\mathtt{a}\star\mathtt{b}\star}$ that "whether the label matches a regular expression $\mathtt{a}\star\mathtt{b}\star$". Although we restrict label operations to only concatenation in the syntax above, it is easy to introduce other simple string operation or arithmetics. An expression $\mathsf{rec}(\lambda\langle \$l, \$g\rangle.M_1)(M_2)$ is a structural recursion on UnCAL graphs obtained by evaluation of M_2. The function part $\mathsf{rec}(\lambda\langle \$l, \$g\rangle.M_1)$ represents a function f satisfying

$$
\begin{array}{rcl}
f(\{\}) & = & \{\} \\
f(\{L : G\}) & = & M_1[\$l := L, \$g := G] \mathbin{@} f(G) \\
f(G_1 \cup G_2) & = & f(G_1) \cup f(G_2)
\end{array}
$$

where $M_1[\$l := L, \$g := G]$ denotes a substitution of L and G for $\$l$ and $\$g$ in M_1. Since all UnCAL graphs can be expressed in the form of $\{l_1 : G_1\} \cup \ldots \cup \{l_n : G_n\}$, the expression $f(M_2)$ can be evaluated using the three rules no matter what is obtained by evaluation of M_2. The second clause shows that each output node of graphs obtained from $M_1[\$l := L, \$g := G]$ is glued with the corresponding input node of graphs obtained from $f(G)$ with its succeeding subgraph G. This recursive function can be implemented through memoizing the result for each subgraph even for the case where input graphs are circular because the function should return the same result for the same subgraph. Indeed, it is known that there is more efficient way to implement structural recursions of UnCAL as mentioned later. It is easy to write mutual recursive functions by using multiple input nodes.

We present the formal semantics of UnCAL expressions in Fig. 4, where $[\![M]\!]_\rho$ denotes a result of evaluation of an UnCAL expression M under an environment ρ. An environment ρ is a function mapping from label variables and graph variables to labels and graphs, respectively, and the initial environment maps from a special variable $\$db$ to the source graph. $dom(f)$ denotes the domain of function f and $sub(G, u)$ denotes a subgraph of G whose root is u. We use μ and ν for Skolem functions to generate new nodes associated with their arguments. The conditions on input/output markers described in the definition of the semantics are statically inferred [6]. Hence, a set Z in the definition for rec can be statically computed from M_1.

Here, we give a brief explanation only for the rec construct which shows a characteristic feature of the semantics of UnCAL in Fig. 4. This semantics is called *bulk semantics* [6], which differs from the semantics described by the informal explanation

$$[\![\{\}]\!]\rho = (\{v\}, \emptyset, \{\& \mapsto v\}, \emptyset) \quad \text{where } v \text{ is fresh}$$

$$[\![\{L : M\}]\!]\rho = (\{v\} \cup V, \{v \xrightarrow{l} u \mid u = I(\&)\} \cup E, \{\& \mapsto v\}, O)$$
$$\text{where} \quad (V, E, I, O) = [\![M]\!]\rho, \quad l = [\![L]\!]\rho, \quad \text{and } v \text{ is fresh}$$

$$[\![M_1 \cup M_2]\!]\rho = (\ \{v_m \mid m \in Z\} \cup V_1 \cup V_2,$$
$$\{v_m \dashrightarrow u_k \mid u_k = I_k(m),\ k = 1, 2\} \cup E_1 \cup E_2,$$
$$\{m \mapsto v_m \mid m \in Z\}, O_1 \cup O_2)$$
$$\text{where} \quad (V_1, E_1, I_1, O_1) = [\![M_1]\!]\rho, \quad (V_2, E_2, I_2, O_2) = [\![M_2]\!]\rho,$$
$$Z = dom(I_1) = dom(I_2), \quad \text{and } v_m \text{ with } m \in Z \text{ are fresh}$$

$$[\![\&z := M]\!]\rho = (V, E, \{\&z \cdot m \mapsto v \mid m \in dom(I), I(m) = v\}, O)$$
$$\text{where} \quad (V, E, I, O) = [\![M]\!]\rho$$

$$[\![\&z]\!]\rho = (v, \emptyset, \{\& \mapsto v\}, \{(v, \&z)\}) \quad \text{where } v \text{ is fresh}$$

$$[\![M_1 \oplus M_2]\!]\rho = (V_1 \cup V_2, E_1 \cup E_2, I_1 \cup I_2, O_1 \cup O_2)$$
$$\text{where} \quad (V_1, E_1, I_1, O_1) = [\![M_1]\!]\rho, \quad (V_2, E_2, I_2, O_2) = [\![M_2]\!]\rho,$$
$$\text{and } dom(I_1) \text{ and } dom(I_2) \text{ are disjoint}$$

$$[\![M_1 \ @\ M_2]\!]\rho = (V_1 \cup V_2, \{v_1 \dashrightarrow v_2 \mid (v_1, m) \in O_1,\ I_2(m) = v_2\} \cup E_1 \cup E_2, I_1, O_2)$$
$$\text{where} \quad (V_1, E_1, I_1, O_1) = [\![M_1]\!]\rho, \quad (V_2, E_2, I_2, O_2) = [\![M_2]\!]\rho,$$

$$[\![\mathsf{cycle}(M)]\!]\rho = (\ \{v_m \mid I(m) = u\} \cup V,$$
$$\{v_m \dashrightarrow u \mid I(m) = u\} \cup \{v \dashrightarrow u \mid (v, m) \in O,\ I(m) = u\} \cup E,$$
$$\{m \mapsto v_m \mid m \in dom(I)\}, \{(v, m) \in O \mid m \notin dom(I)\})$$
$$\text{where} \quad (V, E, I, O) = [\![M]\!]\rho \quad \text{with } v_m \in dom(I) \text{ are fresh}$$

$$[\![\mathsf{if}\ P(L)\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2]\!]\rho = \begin{cases} [\![M_1]\!]\rho & \text{if } P([\![L]\!]) \\ [\![M_2]\!]\rho & \text{otherwise} \end{cases}$$

$$[\![\$g]\!]\rho = \rho(\$g)$$

$$[\![\mathsf{rec}(\lambda\langle \$l, \$g\rangle.M_1)(M_2)]\!]\rho = (V, E_{fromE} \cup E_{fromV}, I, O)$$
$$\text{where} \quad G_2 = (V_2, E_2, I_2, O_2) = [\![M_2]\!]\rho,$$
$$(V_e, E_e, I_e, O_e) = [\![M_1]\!](\rho \cup \{\$l \mapsto d, \$g \mapsto sub(G_2, u)\})$$
$$\text{for } e = \_ \xrightarrow{d} u \in E_2,$$
$$Z = I_e \cup O_e,$$
$$V = \{\mu(u, e) \mid u \in V_e, e \in E_2\} \cup \{\nu(m, v) \mid m \in Z, v \in V_2\},$$
$$E_{fromE} = \{\mu(u, e) \xrightarrow{d} \mu(v, e) \mid u \xrightarrow{d} v \in E_e\},$$
$$E_{fromV} = \{\nu(m, v) \dashrightarrow \mu(u, e) \mid e = v \longrightarrow \_ \in E_2, I_e(m) = u\},$$
$$\cup \{\mu(u, e) \dashrightarrow \nu(m, v) \mid e = \_ \longrightarrow v \in E_2, (u, m) \in O_e\},$$
$$E_\varepsilon = \{\nu(m, v) \dashrightarrow \nu(m, u) \mid m \in Z, v \dashrightarrow u \in E_2\},$$
$$I = \{n \cdot m \mapsto \nu(m, v) \mid I_2(n) = v, m \in Z\},$$
$$O = \{(\nu(m, v), n \cdot m) \mid (v, n) \in O_2, m \in Z\}$$

$$[\![\$l]\!]\rho = \rho(\$l) \qquad [\![C]\!]\rho = C \qquad [\![f(L_1, \ldots, L_n)]\!]\rho = f([\![L_1]\!]\rho, \ldots, [\![L_n]\!]\rho)$$

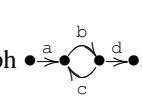
Figure 4: Bulk semantics of UnCAL algebra

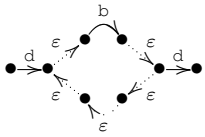
above, called recursive semantics. In the bulk semantics, we can evaluate a rec expression by two steps: first we compute a 'local result' for each edge of the input graph independently, then we combine those results to obtain the output graph. Buneman et al. showed the bulk semantics coincides with the recursive semantics. We adopt the bulk semantics because it makes the range computation easier.

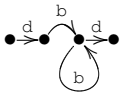
In the bulk semantics, an UnCAL expression $\mathsf{rec}(\lambda\langle \$l, \$g\rangle.M_1)(M_2)$ is evaluated in the following way, where μ and ν are used for Skolem functions to generate a new node: (1) obtain a graph G_2 by evaluating M_2; (2) for each edge e in G_2, evaluate M_1 with binding $\$l$ and $\$g$ to the edge label and the succeeding graph, respectively. Replace each node v with $\mu(v, e)$ in the graph to obtain a graph G_e; (3) generate a node $\nu(m, v)$ for each node v and marker m in G_2; (4) for each edge $e = v \longrightarrow \_$ in G_2, connect node $\nu(m, v)$ with input nodes of marker m using ε edges; (5) for each edge $e = \_ \longrightarrow v$ in G_2, connect output nodes of marker m with node $\nu(m, v)$ using ε edges; (6) for each input/output node of marker n in G_2, a node $\nu(m, v)$ becomes input/output node of marker $n \cdot m$ in the result of the rec expression.

Example 3.1 An UnCAL expression

$$\mathsf{rec}\left(\begin{array}{l}\lambda\langle \$l, \$g\rangle.\ \mathsf{if}\ P_{=\mathtt{a}}(\$l)\ \mathsf{then}\ \{\mathtt{d} : \&\} \\ \qquad\qquad\ \mathsf{else\ if}\ P_{=\mathtt{c}}(\$l)\ \mathsf{then}\ \& \\ \qquad\qquad\ \mathsf{else}\ \{\$l : \&\}\end{array}\right)(\$db)$$

replaces every a edge into a d edge and contract all c edges. When the variable $\$db$ binds a graph 

, the expression returns a graph 

which is bisimilar to 

which do not have ε edges.

4 Simulation-based Graph Schema

Validity of view updating depends on whether the corresponding source satisfies constraints specified by users. We introduce a new kind of graph schema, VU-schema, to represent the constraints. Since our graph schema has a structure analogous to UnCAL graphs, the range of an UnCAL query corresponding to a source schema is easily computed as shown in the next section. In this section, we first review the UnCAL graph schema [4] and its limitation. After that, we shall define our graph schema which compensates the drawback.

Both the UnCAL schema and VU-schema are given using predicates on a set $\mathcal{L}$ of labels. As we use subsets of $\mathcal{L}$ as the predicates, $a \in A$ means that a satisfies a predicate A. We fix an effective representation $\mathcal{C}$ over $\mathcal{L}$ and consider predicates that are elements of $\mathcal{C}$ in the rest of the paper.

4.1 UnCAL Schema

An UnCAL schema developed by Buneman et al. is given by an edge-labeled graph, denoted by a quadruple (V, E, I, O), as same as UnCAL graphs. Each label is a predicate on $\mathcal{L}$, that is an element of $\mathcal{C}$. The schema conformance of an UnCAL graph G

against an UnCAL schema S is checked by the presence of simulation from the nodes of G to those of S. Since the definition of simulation assumes that the domains of edge labels of two graphs are the same, we modify the UnCAL schema S for its edge labels to range over $\mathcal{L}$ through the expansion procedure below before finding a simulation.

Definition 4.1 *The* expansion S^∞ *of an UnCAL schema* $S = (V, E, I, O)$ *is given by an UnCAL graph* (V, E^∞, I, O) *with a possibly infinite number of edges, where* $E^\infty = \{u \xrightarrow{a} v \mid u \xrightarrow{A} v \in E,\ a \in A)\}$. *We say* an UnCAL graph G conforms to S *if* $G \prec S^\infty$. □

From the finiteness of a set of edges of an UnCAL schema S and the effectiveness of predicates on S, we can check the schema conformance by the procedure *FindSim*. Note that we can use a finite representation of expansion of the UnCAL schema for implementation. We do not have to take into account its infiniteness.

We may directly compute a simulation $G \prec S$ by replacing the condition (a) with the following one as shown in [4]:

(a)′ if $u_1 \prec u_2$ and $u_1 \xrightarrow{\varepsilon * . a} v_1$, then there exists v_2 such that $v_1 \prec v_2$, $u_2 \xrightarrow{\varepsilon * . A} v_2$, and $a \in A$.

It is obvious that this definition is equivalent to the schema conformance based on expansion.

One of major advantages of the UnCAL schema is that we can easily compute the range of an UnCAL expression for a given schema. The range is obtained by the evaluation of UnCAL expressions on the UnCAL schema in a way similar to that on UnCAL graphs because the UnCAL schema has the same structure as an UnCAL graph. UnCAL schema has another advantage that we can check subsumption between two UnCAL schemas using the same simulation algorithm as the conformance checking.

Even though it has these nice properties, UnCAL Schema is not suitable to check view updatability. Buneman et al. themselves pointed out that UnCAL schema cannot enforce the presence of some label [4]. This comes from the definition of the schema conformance of a graph G against an UnCAL schema S, which requires the existence of the corresponding node of (expansion of) S for all nodes of G, but not the other way around. In this sense, all edges in the UnCAL schema are *optional*. UnCAL schema cannot describe *necessary* edges. Since a node with no edge is simulated by any node according to the condition (a) (or (a)′), even the single-node graph with no edge conforms to any UnCAL schema. It is not desirable for view updatability checking because we always have to allow for the source graph to be updated into the single node graph that has no information at all.

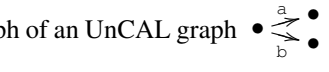
4.2 VU-schema

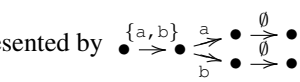
To solve the problem that the UnCAL schema cannot describe necessity of edges, we revise the definition of the graph schema and its conformance. Our schema, called VU-schema, is represented by a graph in which not only edges but also nodes are labeled. A VU-schema is formally represented by a quintuple $(V, E, I, O, \lceil - \rceil)$, where V, E, I and O are the same as the UnCAL schema and $\lceil - \rceil$ is a labeling function from the set V of nodes to the powerset $2^{\mathcal{C}} (\subseteq 2^{2^{\mathcal{L}}})$ of predicates.

Each edge label of a VU-schema is a predicate in $\mathcal{C}$ like UnCAL schema. Each node label of a VU-schema is a finite set of predicates in $\mathcal{C}$ each of which requires the corresponding node of a data graph to have an outgoing edge whose label satisfies

the predicate. The schema conformance for VU-schema is defined by preprocessing both the data graph and the schema though that for the UnCAL schema is defined by preprocessing the schema only. In the following definition, for predicates (that is, subsets of $\mathcal{C}$) $A_1, \ldots, A_n, A$, we write $\{A_1, \ldots, A_n\} \models A$ when $(A \cap A_1 \neq \emptyset) \wedge \cdots \wedge (A \cap A_n \neq \emptyset)$.

Definition 4.2 *The* look-ahead graph *of an UnCAL graph $G = (V_G, E_G, I_G, O_G)$ is given by an UnCAL graph $(V_G \cup V_G^\circ, E^\circ, I_G, O_G)$, where $V_G^\circ = \{v^\circ \mid v \in V_G\}$ and $E^\circ = \{v \xrightarrow{A} v^\circ \mid v \in V_G,\ A = \{a \in \mathcal{L} \mid v \xrightarrow{a} u \in E_G\}\} \cup \{v^\circ \xrightarrow{a} u \mid v \xrightarrow{a} u \in E_G\}$. The* expansion *of a VU-schema $S = (V_S, E_S, I_S, O_S, \lceil - \rceil)$ is given by an UnCAL graph $(V_S \cup V_S^\infty, E_S^\infty, I_S, O_S)$ with a possibly infinite number of nodes and edges, where $V_S^\infty = \{v^\infty \mid v \in V_S\}$ and $E_S^\infty = \{v \xrightarrow{A} v^\infty \mid \lceil v \rceil \models A,\ A$ is finite$\} \cup \{v^\infty \xrightarrow{a} u \mid v \xrightarrow{A} u,\ a \in A\}$. We say G* conforms to *S if $G^\circ \prec S^\infty$.* □

Informally, simulation is checked after transforming the UnCAL graph so that each of its nodes conveys the information of all labels of the outgoing edges in one additional edge. For a node with no outgoing edge, the empty set is assigned as an edge label in the look-ahead graph. For example, the look-ahead graph of an UnCAL graph 

is represented by 

.

The relation $\models$ in the definition above is used to enforce the presence of edges whose label satisfies a predicate of node labels of the VU-schema. The advantage of VU-schema is that we can specify both necessity and optionality of some edges. Even if an edge label does not satisfy all predicates at the corresponding node in the VU-schema, the edge is allowed to exist as far as it satisfies the label of the corresponding edge in the VU-schema.

If S^∞ is pseudo-effective for any VU-schema S, we can employ the procedure *FindSim* to check the schema conformance from the effectiveness of G° and Theorem 2.1. In order for S^∞ to be pseudo-effective, it is necessary that for any finite sets $P, Q \subseteq \mathcal{C}$, the subsumption between $\{A \subseteq \mathcal{L} \mid P \models A,\ A$ is finite$\}$ and $\{A \subseteq \mathcal{L} \mid Q \models A,\ A$ is finite$\}$ is decidable. It is shown by the following lemma.

Lemma 4.3 Let $P = \{P_1, \ldots, P_m\}$ and $Q = \{Q_1, \ldots, Q_n\}$ be a finite subset of $\mathcal{C}$. The following two statements are equivalent and both are decidable:

(I) For any finite set $A \subset \mathcal{L}$, $P \models A$ implies $Q \models A$.

(II) For any $1 \leq i \leq n$, there exists $1 \leq j \leq m$ such that $P_j \subseteq Q_i$.

PROOF. Since (II) immediately implies (I), it suffices to show that (I) implies (II). We prove it by contradiction. Suppose that (I) holds but (II) does not. From the negation of (II), we can take $1 \leq i_0 \leq n$ such that $P_j \cap \overline{Q_{i_0}} \neq \emptyset$ for all $1 \leq j \leq m$. When A is a finite set obtained by selecting an element of $P_j \cap \overline{Q}_{i_0}$ for each j, it is obvious that $P \models A$ and $Q \not\models A$. This contradicts (I). We can conclude the equivalence between (I) and (II). Since (II) is decidable from the effectiveness of $\mathcal{C}$, (I) is also decidable. ■

From the discussion above, we can claim that S^∞ is pseudo-effective for all VU-schema S. It implies that the schema conformance of an UnCAL graph G against a VU-schema S can be checked by using the procedure *FindSim*. From these facts and Theorem 2.1, the termination of the procedure depends on the existence of a finite

simulation between G° and S^∞. It is easy to see the existence of the finite simulation because it suffices to consider a finite subgraph of S^∞ whose edge label occurs in G°, which is finite. Therefore, we obtain the termination of the procedure *FindSim* for the schema conformance against VU-schemas.

Theorem 4.4 *It is decidable whether a finite UnCAL graph conforms to a VU-schema.*

5 Range computation of UnCAL

The range of an UnCAL expression e for a VU-schema S is a set of possible results of the evaluation of e where the source graph conforms to S. We compute the range in a way similar to the method for the UnCAL schema presented by Buneman et al [4]. The basic idea is that we almost directly evaluate an UnCAL expression for a given VU-schema S by exploiting the fact that the schema has an analogous structure to an UnCAL graph. In this section, we present a method for computing the range as a VU-schema in order to check the view updatability by the schema conformance. Since the current definition of VU-schema is less expressive to represent the precise range, we extend VU-schema following the method of Buneman et al.

5.1 Extension of VU-schema

Even though VU-schema is enough expressive to describe a specification of UnCAL graphs, VU-schema cannot represent the precise range of an UnCAL expression. This is because VU-schema cannot describe dependencies between subgraphs. It may remind the readers that the range of tree transformations for a regular tree language cannot be generally represented by any regular tree language [23]. For example, consider a VU-schema for the source $\emptyset \xrightarrow[\texttt{[0-9]+}]{} \emptyset$ and an UnCAL expression $\mathsf{rec}(\lambda\langle \$l, \$g\rangle.\{\$l : \{\$l : \{\}\}\})(\$db)$. The schema is conformed to by a graph of a single edge labeled with numeral sequences; the UnCAL expression duplicates each edge from the root, e.g., for UnCAL graphs $\bullet \xrightarrow[3]{} \bullet$ and $\bullet \overset{1}{\underset{2}{\rightrightarrows}} \bullet$, the expression returns $\bullet \xrightarrow[3]{} \bullet \xrightarrow[3]{} \bullet$ and $\bullet \overset{1}{\underset{2}{\rightrightarrows}} \bullet \xrightarrow[2]{} \bullet$, respectively. The exact range should be "a set of graphs in which the first and the second edge from the root have the same label of a numerical sequence. There is no VU-schema representing the range, however. A naïve method may overestimate the range by the VU-schema $\emptyset \xrightarrow[\texttt{[0-9]+}]{} \emptyset \xrightarrow[\texttt{[0-9]+}]{} \emptyset$, which is not precise because it contains an impossible view $\bullet \xrightarrow[1]{} \bullet \xrightarrow[2]{} \bullet$. This problem has already been pointed out by Buneman et al. [4]. They also gave a (partial) solution to it by introducing *scoped variables*, which is applicable to our VU-schema.

Dependency among edge labels can be represented by shared variables. In the example above, the range is represented by

$$\emptyset \xrightarrow[\texttt{[0-9]+} \cap \{a | a{=}x\}]{} \emptyset \xrightarrow[\texttt{[0-9]+} \cap \{a | a{=}x\}]{} \emptyset$$

using a shared variable x. It accepts graphs in which the first and the second edge have the same label. We can introduce multiple variables in an extended VU-schema. Each variable x in the extended VU-schema $S = (V, E, I, O, \lceil - \rceil)$ has a *scope*, denoted by a VU-schema $S_x = (V_x, E_x, I_x, O_x, \lceil - \rceil_x)$, which obeys the following rule:

(1) $V_x \subseteq V$, $E_x \subseteq E$, $dom(I_x) \subseteq V_x$, $O_x \subseteq V_x \times \mathcal{M}$, and $\lceil - \rceil_x \subseteq \lceil - \rceil$;

(2) if $u \xrightarrow{A} v \in E_x$, then $u, v \in V_x$;

(3) if $u \xrightarrow{A} v \in E$, $u \notin V_x$ and $v \in V_x$, then $v \in dom(I_x)$;

(4) if $m \in \mathcal{M}$ and $I(m) \in V_x$, then $I_x(m) = I(m)$;

(5) if $u \xrightarrow{A} v \in E$, $u \in V_x$ and $v \notin V_x$, $(u, m) \in O_x$ for some $m \in \mathcal{M}$;

(6) if $u \xrightarrow{A} v \in E_x$, then x may be used as a label constant in the set A;

(7) if $v \in V_x$ and $A \in \lceil v \rceil$, then x may be used as a label constant in the set A;

(8) the scopes of two distinct variables are either related by subsumption or disjoint.

We call *k-variable VU-schema* for an extended VU-schema with k scoped variables. The ordinary VU-schemas presented in Section 4 can be seen as 0-variable VU-schemas. The schema conformance against a k-variable VU-schema is defined by expansion and simulation in the same way as that against the ordinary VU-schema. While we expand schemas for each edge in Definition 4.1 and Definition 4.2, we expand a k-variable VU-schema for each scope at the same time. The expansion of k-variable VU-schema is inductively defined as follows.

Definition 5.1 Let $S = (V, E, I, O, \lceil - \rceil)$ be a k-variable VU-schema with scoped variables $x_0, \ldots, x_{k-1}$. The expansion S^∞ of S is a (possibly infinite) UnCAL graph given by S_k in the following procedure:

(i) Elimination of node labels: a VU-schema S_0 is given by the expansion S^∞ of S as shown Definition 4.2, where $x_0, \ldots, x_{k-1}$ are regarded as label constants. For each variable x_i, all node labels are eliminated in a scope S_{x_i} and $(S_0)_{x_i}$ is naturally defined.

(ii) Elimination of variables: $S_{i+1} = (V_{i+1}, E_{i+1}, I, O)$ is obtained by removing a variable x_i from $S_i = (V_i, E_i, I, O)$, where

$$V_{i+1} = (V_i \setminus (V_i)_{x_i}) \cup \{v_a \mid v \in (V_i)_{x_i},\ a \in \mathcal{L}\,(v_a = v \text{ for } v \in (I_i)_{x_i})\},$$
$$E_{i+1} = (E_i \setminus (E_i)_{x_i}) \cup \{u_a \xrightarrow{l[x_i:=a]} v_a \mid u \xrightarrow{l} v \in E_{x_i},\ a \in \mathcal{L}\}$$

in which $l[x := a]$ denotes a label obtained by replacing x with a in l.

We say *a UnCAL graph G conform to S* if $G^\circ \prec S^\infty$.

The expanded schema S^∞ for k-variable VU-schema is pseudo-effective since the number of steps for the expansion is finite. The order of variables does not affect the result of the expansion because of the scope rule (8).

Another improvement for computing of precise ranges is the introduction of ε edges labeled with predicates in $\mathcal{C}$. This will be used for the range of if expressions in UnCAL. Since the expansion of these ε edges replaces each predicate of their labels with labels that satisfies it, ε edges labeled with the empty set are removed and the other ε edges are left. We will show their usage in the next subsection.

We have to remark that these improvements are not sufficient for the computation of the exact range, unfortunately. A k-variable schema can describe dependencies between edge labels but not between structures of subgraphs. This problem is inevitable

because of the expressiveness of (k-variable) VU-schema, which is similar to the well-known problem as a forward typechecking of XML transformation [23]. Hence, we just show the soundness of view updatability checking that "the view is in the computed range if there exists the corresponding source" in this paper. We compromise the completeness which is the opposite. Since Buneman et al. have shown "the computed range is the best approximation in UnCAL schema" instead of the completeness [4, Theorem 16], we expect the same result for our VU-schema. This is left as our future work.

5.2 Range computation

We can compute the range of an UnCAL expression for a given VU-schema in the same way as the evaluation of the expression because the VU-schema has an analogous structure to an UnCAL graph. Though it can be seen as a kind of abstract interpretation [8], we do not require a complicated domain such as CPO or complete lattices employed by general abstract interpretation since recursion in UnCAL can be evaluated in a simple bulk semantics as shown in Section 3.

The range of an UnCAL expression is computed as a k-variable VU-schema where k depends on the number of rec constructs in the expression. In Fig. 5 and Fig. 6, we define the range computation function $\mathcal{I}_\rho$ and the label function $\mathcal{L}_\rho$ with a variable environment ρ. The environment ρ is a function which maps label variables and graph variables to labels and VU-schema, respectively. The initial environment is a function $\{\$db \mapsto S_{in}\}$ with a VU-schema S_{in} for the specification of source graphs. $\mathcal{I}_\rho$ is a function from UnCAL expressions to k-variable VU-schemas. $\mathcal{L}_\rho$ is a function from label expressions to labels. Skolem functions μ and ν in the definition of $\mathcal{I}$, which are used in that of $[\![-]\!]$ in Fig. 4. Note that the function $\mathcal{I}$ does not compute node labels which stands for necessity of edges in VU-schema. We put them for the k-variable VU-schema of the computed range after removal of ε edges, which will be explained later.

Since the ranges of most of UnCAL expressions are computed in a way similar to their semantics, we explain the range computation only for if and rec constructs. Our range computation follows the method presented in [4]* except that necessity of edges has to be concerned.

The range of UnCAL expression $\mathsf{if}\ P(L)\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2$ depends on the result of the condition $P(L)$. If we simply compute it by union of the ranges for M_1 and M_2, the computed range can easily be imprecise. For example, consider an UnCAL expression $\mathsf{rec}(\lambda\langle \$l, \$g\rangle.\ \mathsf{if}\ P_{=\mathtt{a}}(\$l)\ \mathsf{then}\ \&\ \mathsf{else}\ \{\$l : \&\})(\$db)$ which represents a graph query which contracts all a edges. If we take union of ranges for then- and else clauses as the range of the whole expression, we lose information which edges are labeled with a in the source. Therefore, we introduce special ε edges labeled with predicates corresponding the condition for the if expression. Informally, when P is a predicate (that is, a set of labels) for the conditional branch, the range of the if expression is $\emptyset \overset{P}{<\!\!-\!\!\succ}$ the range of M_1 / $\overset{\overline{P}}{<\!\!-\!\!\succ}$ the range of M_2.

The range of UnCAL expression $\mathsf{rec}(\lambda\langle \$l, \$g\rangle.M_1)(M_2)$ is computed through the bulk semantics presented in Section 3. We first compute the range S_{M_2} of the argument M_2, evaluate the expression M_1 for each of the edge of S_{M_2}, and combine all the results. For each edge e of S_{M_2}, we introduce a scoped variable z_e, whose scope is

*The concrete range computation for UnCAL schemas is found their technical report of the same title.

$$\mathcal{I}_\rho(\{\}) = (\{v\}, \emptyset, \{\& \mapsto v\}, \emptyset, -) \quad \text{where } v \text{ is fresh}$$

$$\mathcal{I}_\rho(\{L : M\}) = (\{v\} \cup V, \{v \xrightarrow{A} u \mid u \in dom(I)\} \cup E, \{\& \mapsto v\}, O, -)$$
$$\text{where} \quad (V, E, I, O, -) = \mathcal{I}_\rho(M), \quad A = \mathcal{L}_\rho(L), \quad \text{and } v \text{ is fresh}$$

$$\mathcal{I}_\rho(M_1 \cup M_2) = (\,\{v_m \mid m \in M\} \cup V_1 \cup V_2,$$
$$\{v_m \overset{\mathcal{L}}{\dashrightarrow} u_k \mid u_k = I_k(m),\ k = 1, 2\} \cup E_1 \cup E_2,$$
$$\{m \mapsto v_m \mid m \in M\}, O_1 \cup O_2, -)$$
$$\text{where} \quad (V_1, E_1, I_1, O_1, -) = \mathcal{I}_\rho(M_1)$$
$$(V_2, E_2, I_2, O_2, -) = \mathcal{I}_\rho(M_2)$$
$$M = dom(I_1) = dom(I_2)$$
$$v_m \text{ with } m \in M \text{ are fresh}$$

$$\mathcal{I}_\rho(\&z := M) = (V, E, \{\&z \cdot m \mapsto v \mid I(m) = v\}, O, -)$$
$$\text{where } (V, E, I, O, -) = \mathcal{I}_\rho(M)$$

$$\mathcal{I}_\rho(\&z) = (\{v\}, \emptyset, \{\& \mapsto v\}, \{(v, \&z)\}, -) \quad \text{where } v \text{ is fresh}$$

$$\mathcal{I}_\rho(M_1 \oplus M_2) = (V_1 \cup V_2, E_1 \cup E_2, I_1 \cup I_2, O_1 \cup O_2, -)$$
$$\text{where} \quad (V_1, E_1, I_1, O_1, -) = \mathcal{I}_\rho(M_1)$$
$$(V_2, E_2, I_2, O_2, -) = \mathcal{I}_\rho(M_2)$$
$$I_1 \text{ and } I_2 \text{ are disjoint}$$

$$\mathcal{I}_\rho(M_1 \mathbin{@} M_2) = (\,V_1 \cup V_2,$$
$$\{v_1 \overset{\mathcal{L}}{\dashrightarrow} v_2 \mid (v_1, m) \in O_1, I_2(m) = v_2\} \cup E_1 \cup E_2, I_1, O_2, -)$$
$$\text{where} \quad (V_1, E_1, I_1, O_1, -) = \mathcal{I}_\rho(M_1)$$
$$(V_2, E_2, I_2, O_2, -) = \mathcal{I}_\rho(M_2)$$

$$\mathcal{I}_\rho(\mathsf{cycle}(M)) = (\,\{v_m \mid I(m) = u\} \cup V,$$
$$\{v_m \overset{\mathcal{L}}{\dashrightarrow} u \mid I(m) = u\} \cup \{v \overset{\mathcal{L}}{\dashrightarrow} u \mid (v, m) \in O, I(m) = u\} \cup E,$$
$$\{m \mapsto v_m \mid I(m) = u\}, \{(v, m) \in O \mid m \notin dom(I)\}, -)$$
$$\text{where} \quad (V, E, I, O, -) = \mathcal{I}_\rho(M)$$

$$\mathcal{I}_\rho(\$g) = \rho(\$g)$$

Figure 5: Range computation of UnCAL expressions (except if and rec)

$$
\begin{array}{l}
\mathcal{I}_\rho(\mathsf{if}\ P(L)\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2) = \\
\quad (\{v_m \mid m \in M\} \cup V_1 \cup V_2, \\
\quad \{v_m \overset{P \cap \{\mathcal{L}_\rho(L)\}}{\dashrightarrow} u_1 \mid u_1 = I_1(m)\} \cup \{v_m \overset{\overline{P} \cap \{\mathcal{L}_\rho(L)\}}{\dashrightarrow} u_2 \mid u_2 = I_2(m)\} \cup E_1 \cup E_2, \\
\quad \{m \mapsto v_m \mid m \in M\}, O_1 \cup O_2, -) \\
\quad \text{where} \quad (V_1, E_1, I_1, O_1, -) = \mathcal{I}_\rho(M_1) \\
\qquad\qquad (V_2, E_2, I_2, O_2, -) = \mathcal{I}_\rho(M_2) \\
\qquad\qquad M = dom(I_1) = dom(I_2) \\
\qquad\qquad v_m \text{ with } m \in M \text{ are fresh}
\end{array}
$$

$$
\begin{array}{l}
\mathcal{I}_\rho(\mathsf{rec}(\lambda\langle \$l, \$g\rangle.M_1)(M_2)) = (V, E_{fromE} \cup E_{fromV}, I, O, -) \\
\quad \text{where} \quad G_2 = (V_2, E_2, I_2, O_2, -) = \mathcal{I}_\rho(M_2), \\
\qquad\qquad G_e = (V_e, E_e, I_e, O_e, -) = \mathcal{I}_{\rho \cup \{\$l \mapsto (z_e : A), \$g \mapsto sub(G_2, u)\}}(M_1) \\
\qquad\qquad\quad \text{with scoped variables } z_e \text{ for } e = \_ \xrightarrow{A} \_ \in E_2 \\
\qquad\qquad Z = I_e \cup O_e \\
\qquad\qquad V = \{\mu(u, e) \mid u \in V_e, e \in E_2\} \cup \{\nu(m, v) \mid m \in Z, v \in V_2\} \\
\qquad\qquad E_{fromE} = \{\mu(u, e) \xrightarrow{A} \mu(v, e) \mid u \xrightarrow{A} v \in E_e\} \\
\qquad\qquad E_{fromV} = \ \{\nu(m, v) \overset{A \cap \{z_e\}}{\dashrightarrow} \mu(u, e) \mid e = v \xrightarrow{A} \_ \in E_2, I_e(m) = u\} \\
\qquad\qquad\qquad\qquad \cup \{\mu(u, e) \overset{\mathcal{L}}{\dashrightarrow} \nu(m, v) \mid e = \_ \longrightarrow v \in E_2, (u, m) \in O_e\} \\
\qquad\qquad\quad \text{with fresh variables } v_{\langle m,e\rangle} \text{ for } m \in Z \text{ and } e \in E_2, \\
\qquad\qquad E_\varepsilon = \{\nu(m, v) \overset{A}{\dashrightarrow} \nu(m, u) \mid m \in Z, v \overset{A}{\dashrightarrow} u \in E_2\} \\
\qquad\qquad I = \{n \cdot m \mapsto \nu(m, v) \mid I_2(n) = v, m \in Z\}, \\
\qquad\qquad O = \{(\nu(m, v), n \cdot m) \mid (v, n) \in O_2, m \in Z\}
\end{array}
$$

$$
\mathcal{L}_\rho(\$l) = \rho(\$l), \quad \mathcal{L}_\rho(C) = C, \quad \mathcal{L}_\rho(f(L_1, \ldots, L_n)) = f(\mathcal{L}_\rho(L_1), \ldots, \mathcal{L}_\rho(L_n))
$$

Figure 6: Range computations for if, rec and label expressions

G_e, to compute the range of M_1. Intuitively, z_e corresponds to an edge label of a graph conforming to S_{M_2}, which is bound to $\$l$. This scoped variable is used to share the edge label on computing the range G_e, that is a 'local' result of M_1 at the edge e.

The procedure of the range computation explained so far is the same as the original one for UnCAL schemas [4]. For the range computation on VU-schema, we have to take into account node labels for necessary edges. They are put after removal of ε edges for the computed range.

We remove ε edges by replacing all occurrence of $u \dashrightarrow u_1 \dashrightarrow \ldots \dashrightarrow u_n \xrightarrow{A} v$ with $u \xrightarrow{A} v$, which is a traditional way for ε transition removal in automata theory. Since an ε edge in VU-schemas is labeled with a subset of $\mathcal{L}$, we integrate all concerning labels in the replacement. For a path $u \overset{A_1}{\dashrightarrow} u_1 \overset{A_2}{\dashrightarrow} \ldots \overset{A_n}{\dashrightarrow} u_n \xrightarrow{A} v$ in VU-schema, we replace it with $u \xrightarrow{A'} v$ where $A' = \{a \in A \mid A_1 \neq \emptyset, \ldots, A_n \neq \emptyset\}$. This replacement may be applied in multiple times for the same non-ε edge like ε transition removal in automata theory.

Now we are ready to put node labels on the VU-schema after the ε edge removal. Let u be a node in the VU-schema and E_u a set of all outgoing edges of u. For a scoped variable z_e, we know that it comes from a non-ε edge e from the procedure of the range computation. When P_e denotes a set of predicates labeled for a source node of e, the label $\lceil u \rceil$ contains a predicate

$$A \setminus \{a \in A \mid z_e \in \bigcap_{P \in P_e} \overline{P}\}$$

for each edge $e' = u \xrightarrow{A} \_ \in E_u$. The predicate forces to have an edge which is originated from some necessity edges. Since these kinds of predicates is defined for each edge in E_u, the cardinality of $\lceil u \rceil$ coincides with that of E_u.

Let $\mathcal{I}^+$ be a range computing function which returns VU-schemas after adding node labels to the result of the function $\mathcal{I}$. The definition of $\mathcal{I}$ basically follows that of $[\![-]\!]$. It can be shown by simple induction on the structure of UnCAL expression that $[\![M]\!]_{\rho'}$ conforms to $\mathcal{I}^+_\rho(M)$ conforms to for any UnCAL expression M and any environments ρ and ρ' such that $\rho'(\$g)$ conforms to $\rho(\$g)$ for all graph variables $\$g$ and $\rho'(\$l) = \rho(\$l)$ for all label variables $\$l$. Therefore we have the following theorem.

Theorem 5.2 Let S be a VU-schema, G be a finite UnCAL graph conforming to S, and Q be an UnCAL expression. Then the UnCAL graph $[\![Q]\!]_{\{\$db \mapsto G\}}$ conforms to $\mathcal{I}^+_{\{\$db \mapsto S\}}(Q)$.

From the decidability of the schema conformance on k-variable VU-schema, this theorem shows that it is decidable whether there exists the corresponding source for the updated view. Note that our view updatability checking is sound but not complete, however. Since the computed range is just an approximation, there can be the case where no source graph corresponds to an updated view in the range. As aforementioned, it is impossible to compute the exact range in the present VU-schema.

6 Related work

Many models of graph transformation have been proposed in the literature [22, 16]. The reason why we choose UnCAL to describe graph queries is that UnCAL has a solution to the view updating problem [13].

We introduce a simulation-based graph schema, VU-schema, following UnCAL schema [4]. It helps us to easily compute the range of graph queries. We improve UnCAL schema so that it can describe both necessity and optionality of edges.

There exists other work to describe a specification of graphs. Klarlund and Schwartzbach [15] proposed *graph types* to describe a shape of graphs for the same purpose as ours. A graph type is represented by a tree with pointers which is analogous to a rooted graph we deal with. However, we cannot use graph types in the context of view updatability checking of UnCAL graph queries because two bisimilar graphs that UnCAL regards as equivalent ones can have different graph types.

Computing the image of graph transformation as a VU-schema can be considered as a kind of type inferences. It reminds us of a relation with a type inference for polymorphic records [21, 19] by identifying edge labels with fields of records (or objects). Major difference is that none of them can deal with dynamically evaluated values for the names of fields. In contrast, VU-schema allows edge labels to have any kind of values, e.g., strings, integers and reals which can be a result of evaluation. Additionally, their record types have the same problem on graph equivalence in UnCAL as graph types.

Another possible approach to describing graph structures is to use monadic second-order logic (MSO). Although MSO formulae can describe more flexible constraints on the shape of graphs than VU-schema, we must carefully choose an appropriate subclass of MSO theory to be decidable. Inaba et al.[14] attack the validation problem of UnCAL by reducing it into decidable MSO formulae. They cannot deal with all expressions (of UnCAL) as we can tackle, however, because that graph transformation defined by an MSO formula is linear-size increase from the definition [7], i.e., the size of outputs is linearly bounded with that of inputs. It is easy to define quadratic-size increase transformation through nested structural recursion.

7 Conclusion

We have proposed a new graph schema, VU-schema, and a novel method for computing the range of a graph query for sources conforming to a schema in order to check view updatability for the query. VU-schema is simulation-based in the sense that the schema conformance is checked by finding simulation relation between a given schema and graph. The idea is borrowed from UnCAL schema of Buneman et al., but have improved their schema so that it can enforce necessity and optionality of certain edges. Our framework presented in this report will be implemented as a part of our system, GRoundTram (`http://www.biglab.org/`), which involves view updating on graph queries.

References

[1] M. Alanen and I. Porres. Differences and union of models. In *UML 2003 - The Unified Modeling Language. Model Languages and Applications. 6th International Conference, San Francisco, CA, USA, Proceedings*, volume 2863 of *LNCS*, pages 2–17. Springer, 2003.

[2] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems*, 6(4):557–575, 1981.

[3] A. Bohannon, J. N. Foster, B. C. Pierce, A. Pilkiewicz, and A. Schmitt. Boomerang: resourceful lenses for string data. In G. C. Necula and P. Wadler, editors, *POPL '08: ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages*, pages 407–419. ACM, 2008.

[4] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proceedings of the 6th International Conference on Database Theory*, volume 1186 of *LNCS*, pages 336–350, 1997.

[5] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of ACM SIGMOD international conference on Management of Data*, pages 505–516. ACM, 1996.

[6] P. Buneman, M. F. Fernandez, and D. Suciu. UnQL: a query language and algebra for semistructured data based on structural recursion. *VLDB Journal: Very Large Data Bases*, 9(1):76–110, 2000.

[7] B. Courcelle. Monadic second-order definable graph transductions: A survey. *Theoretical Computer Science*, 126(1):53–75, 1994.

[8] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '77, pages 238–252, New York, NY, USA, 1977. ACM.

[9] U. Dayal and P. A. Bernstein. On the correct translation of update operations on relational views. *ACM Transactions on Database Systems*, 7(3):381–416, 1982.

[10] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. In *POPL '05: ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages*, pages 233–246, 2005.

[11] G. Gottlob, P. Paolini, and R. Zicari. Properties and update semantics of consistent views. *ACM Transactions on Database Systems*, 13(4):486–524, 1988.

[12] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of 20th Symposium on Foundations of Computer Science*, pages 453–462. IEEE Computer Society Press, 1995.

[13] S. Hidaka, Z. Hu, K. Inaba, H. Kato, K. Matsuda, and K. Nakano. Bidirectionalizing graph transformations. In *ACM SIGPLAN International Conference on Functional Programming*, pages 205–216. ACM, 2010.

[14] K. Inaba, S. Hidaka, Z. Hu, H. Kato, and K. Nakano. Sound and complete validation of graph transformations. Technical Report GRACE-TR-2010-04, GRACE Center, NII, 2010.

[15] N. Klarlund and M. I. Schwartzbach. Graph types. In *POPL '93: Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 196–205, New York, NY, USA, 1993. ACM.

[16] B. König. A general framework for types in graph rewriting. *Acta Inf.*, 42:349–388, December 2005.

[17] K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and M. Takeichi. Bidirectionalization transformation based on automatic derivation of view complement functions. In *12th ACM SIGPLAN International Conference on Functional Programming (ICFP 2007)*, pages 47–58. ACM Press, Oct. 2007.

[18] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[19] A. Ohori. A polymorphic record calculus and its compilation. *ACM Trans. Program. Lang. Syst.*, 17(6):844–895, 1995.

[20] OMG. Metaobject facility (MOF) specification. `http://www.omg.org/docs/formal/02-04-03.pdf`, 2002.

[21] D. Rémy. Type inference for records in natural extension of ML. In *Theoretical aspects of object-oriented programming: types, semantics, and language design*, pages 67–95. MIT Press, Cambridge, MA, USA, 1994.

[22] G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation: volume I. foundations*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.

[23] D. Suciu. Typechecking for semistructured data. In *Revised Papers from the 8th International Workshop on Database Programming Languages*, DBPL '01, pages 1–20, London, UK, 2002. Springer-Verlag.