A Logic-Based Incremental Approach to Graph Repair

Sven Schneider[1(✉)], Leen Lambers[1], and Fernando Orejas[2]

[1] Hasso Plattner Institut, University of Potsdam, Potsdam, Germany
Sven.Schneider@HPI.de
[2] Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract. Graph repair, restoring consistency of a graph, plays a prominent role in several areas of computer science and beyond: For example, in model-driven engineering, the abstract syntax of models is usually encoded using graphs. Flexible edit operations temporarily create inconsistent graphs not representing a valid model, thus requiring graph repair. Similarly, in graph databases—managing the storage and manipulation of graph data—updates may cause that a given database does not satisfy some integrity constraints, requiring also graph repair.

We present a logic-based incremental approach to graph repair, generating a sound and complete (upon termination) overview of least-changing repairs. In our context, we formalize consistency by so-called graph conditions being equivalent to first-order logic on graphs. We present two kind of repair algorithms: State-based repair restores consistency independent of the graph update history, whereas delta-based (or incremental) repair takes this history explicitly into account. Technically, our algorithms rely on an existing model generation algorithm for graph conditions implemented in AutoGraph. Moreover, the delta-based approach uses the new concept of satisfaction (ST) trees for encoding if and how a graph satisfies a graph condition. We then demonstrate how to manipulate these STs incrementally with respect to a graph update.

1 Introduction

Graph repair, restoring consistency of a graph, plays a prominent role in several areas of computer science and beyond. For example, in model-driven engineering, models are typically represented using graphs and the use of flexible edit operations may temporarily create inconsistent graphs not representing a valid model, thus requiring graph repair. This includes the situation where different views of an artifact are represented by a different model, i.e., the artifact is described by a multi-model, see, e.g. [6], and updates in some models may cause a global inconsistency in the multimodel. Similarly, in graph databases—managing the storage

F. Orejas has been supported by the Salvador de Madariaga grant PRX18/00308 and by funds from the Spanish Research Agency (AEI) and the European Union (FEDER funds) under grant GRAMM (ref. TIN2017-86727-C2-1-R).

R. Hähnle and W. van der Aalst (Eds.): FASE 2019, LNCS 11424, pp. 151–167, 2019.
https://doi.org/10.1007/978-3-030-16722-6_9

and manipulation of graph data—updates may cause that a given database does not satisfy some integrity constraints [1], requiring also graph repair.

Numerous approaches on model inconsistency and repair (see [12] for an excellent recent survey) operate in varying frameworks with diverse assumptions. In our framework, we consider a typed directed graph (cf. [7]) to be inconsistent if it does not satisfy a given finite set of constraints, which are expressed by graph conditions [8], a formalism with the expressive power of first-order logic on graphs. A graph repair is, then, a description of an update that, if applied to the given graph, makes it consistent. Our algorithms do not just provide one repair, but a set of them from which the user must select the right repair to be applied. Moreover, we derive only least changing repairs, which do not include other smaller viable repairs. Our approach uses techniques (and the tool AutoGraph) [17] designed for model generation of graph conditions.

We consider two scenarios: In the first one, the aim is to repair a given graph (state-based repair). In the second one, a consistent graph is given together with an update that may make it inconsistent. In this case, the aim is to repair the graph in an incremental way (delta-based repair).

The main contributions of the paper are the following ones:

– A precise definition of what an update is, together with the definition of some properties, like e.g. least changing, that a repair update may satisfy.
– Two kind of graph repair algorithms: state-based and incremental (for the delta-based case). Moreover, we demonstrate for all algorithms *soundness* (the repair result provided by the algorithms is consistent) and *completeness* (upon termination, our algorithms will find all possible desired repairs)[1].

Summarizing, most repair techniques do not provide guarantees for the functional semantics of the repair and suffer from lack of information for the deployment of the techniques (see conclusion of the survey [12]). With our logic-based graph repair approach we aim at alleviating this weakness by presenting formally its functional semantics and describing the details of the underlying algorithms.

The paper is organized as follows: After introducing preliminaries in Sect. 2, we proceed in Sect. 3 with defining graph updates and repairs. In Sect. 4, we present the state-based scenario. We continue with introducing satisfaction trees in Sect. 5 that are needed for the delta-based scenario in Sect. 6. We close with a comparison with related work in Sect. 7 and conclusion with outlook in Sect. 8. For proofs of theorems and example details we refer to our technical report [18].

2 Preliminaries on Graph Conditions

We recall graph conditions (GCs), defined here over typed directed graphs, used for representing properties on such graphs. In our running example[2], we employ

[1] Note that completeness implies totality (if the given set of constraints is satisfiable by a finite graph, then the algorithms will find a repair for any inconsistent graph).

[2] We refer to Sect. 1 with pointers to related work including diverse use cases in Software Engineering for graph repair with more complex and motivating examples.

$$\begin{tikzcd} \boxed{:A} \arrow[loop left, ":E_1"] \arrow[r, ":E_2"] & \boxed{:B} \end{tikzcd} \qquad \neg\exists(\,a, \neg(\exists(a \xrightarrow{e} b, \mathit{true}) \wedge \neg\exists(a \circlearrowleft e, \mathit{true})))$$

Fig. 1. The type graph TG (left) and the GC ψ (right) for our running example

the type graph TG from Fig. 1 and we use nodes with names a_i and b_i to indicate that they are of type $:A$ and $:B$, respectively.

GCs state facts about the existence of graph patterns in a given graph, called a host graph. For example, in the syntax used in our running example, the GC $\exists(a, \mathit{true})$ means that the host graph must include a node of type $:A$. Also, $\exists(a \longrightarrow b, \mathit{true})$ means that the host graph must include a node of type $:A$, another node of type $:B$, and an edge from the $:A$-node to the $:B$-node.

In general, in the syntax that we use in our running example, an atomic GC is of the form $\exists(H, \phi)$ (or $\neg\exists(H, \phi)$) where H is a graph that must be (or must not be) included in the host graph and where ϕ is a condition expressing more restrictions on how this graph is found (or not found) in the host graph. For instance, $\exists(\,a, \neg\exists(a \xrightarrow{e} b, \mathit{true}))$ states that the host graph must include an $:A$-node such that it has no outgoing edge e to a $:B$-node. Moreover, we use the standard boolean operators to combine atomic GCs to form more complex ones. For instance, $\exists(\,a, \neg(\exists(a \xrightarrow{e} b, \mathit{true}) \wedge \neg\exists(a \circlearrowleft e, \mathit{true})))$ states that the host graph must include an $:A$-node, such that it does not hold that there is an outgoing edge e to a $:B$-node and node a has no loop. In addition, as an abbreviation for readability, we may use the universal quantifier with the meaning $\forall(H, \phi) = \neg\exists(H, \neg\phi)$. In this sense, the condition ϕ from Fig. 1, used in our running example, states that every node of type $:A$ must have an outgoing edge to a node of type $:B$ and that such an $:A$-node must have no loop.

Formally, the syntax of GCs [8], expressively equivalent to first-order logic on graphs [5], is given subsequently. This logic encodes properties of graph extensions, which must be explicitly mentioned as graph inclusions. For instance, the GC $\exists(\,a, \neg\exists(a \xrightarrow{e} b, \mathit{true}))$ in simplified notation is formally given in the syntax of GCs as $\exists(\mathrm{i}_H, \neg\exists(a \hookrightarrow (a \xrightarrow{e} b), \mathit{true}))$, where i_H denotes the inclusion $\emptyset \hookrightarrow H$ with H the graph consisting of node a. This is because it expresses a property of the extension i_H. Moreover, therein the GC $\neg\exists(a \hookrightarrow (a \xrightarrow{e} b), \mathit{true})$ is actually a property of the extension $a \hookrightarrow (a \xrightarrow{e} b)$.

Definition 1 (Graph Conditions (GCs) [8]). *The class of* graph conditions Φ_H^{GC} *for the graph H is defined inductively:*

– $\wedge S \in \Phi_H^{\mathrm{GC}}$ *if* $S \subseteq_{\mathrm{fin}} \Phi_H^{\mathrm{GC}}$.
– $\neg\phi \in \Phi_H^{\mathrm{GC}}$ *if* $\phi \in \Phi_H^{\mathrm{GC}}$.
– $\exists(a : H \hookrightarrow H', \phi) \in \Phi_H^{\mathrm{GC}}$ *if* $\phi \in \Phi_{H'}^{\mathrm{GC}}$.

In addition true, false, $\vee S$, $\phi_1 \Rightarrow \phi_2$, and $\forall(a, \phi)$ can be used as abbreviations, with their obvious replacement.

A mono $m : H \hookrightarrow G$ satisfies *a* GC *$\psi \in \Phi_H^{\mathrm{GC}}$, written $m \models_{\mathrm{GC}} \psi$, if one of the following cases applies.*

– $\psi = \wedge S$ *and* $m \models_{\mathrm{GC}} \phi$ *for each* $\phi \in S$.
– $\psi = \neg\phi$ *and not* $m \models_{\mathrm{GC}} \phi$.
– $\psi = \exists(a : H \hookrightarrow H', \phi)$ *and* $\exists q : H' \hookrightarrow G.\ q \circ a = m \wedge q \models_{\mathrm{GC}} \phi$.

A graph G *satisfies a* GC $\psi \in \Phi_{\emptyset}^{\mathrm{GC}}$*, written* $G \models_{\mathrm{GC}} \psi$ *or* $G \in [\![\psi]\!]$*, if* $\mathrm{i}_G \models_{\mathrm{GC}} \psi$.

3 Graph Updates and Repairs

In this section, we define graph updates to formalize arbitrary modifications of graphs, graph repairs as the desired graph updates resulting in repaired graphs, as well as further desireable properties of graph updates.

In particular, it is well known that a modification or update of G_1 resulting in a graph G_2 can be represented by two inclusions or, in general two monos, which we denote by $(l : I \hookrightarrow G_1, r : I \hookrightarrow G_2)$, where I represents the part of G_1 that is preserved by this update. Intuitively, $l : I \hookrightarrow G_1$ describes the deletion of elements from G_1 (i.e., all elements in $G_1 \setminus l(I)$ are deleted) and $r : I \hookrightarrow G_2$ describes the addition of elements to I to obtain G_2 (i.e., all elements in $G_2 \setminus r(I)$ are added).

Definition 2 (Graph Update). *A* (graph) update u *is a pair* $(l : I \hookrightarrow G_1, r : I \hookrightarrow G_2)$ *of monos. The class of all updates is denoted by* $\mathcal{U}$.

Graph updates such as $(\mathrm{i}_G : \emptyset \hookrightarrow G, \mathrm{i}_G : \emptyset \hookrightarrow G)$ where G is not the empty graph delete all the elements in G that are added by r afterwards. To rule out such updates, we define an update $(l : I \hookrightarrow G_1, r : I \hookrightarrow G_2)$ to be *canonical* when the graph I is as large as possible, i.e. intuitively $I = G_1 \cap G_2$. Formally:

Definition 3 (Canonical Graph Update). *If* $(l : I \hookrightarrow G_1, r : I \hookrightarrow G_2) \in \mathcal{U}$ *and every* $(l' : I' \hookrightarrow G_1, r' : I' \hookrightarrow G_2) \in \mathcal{U}$ *and mono* $i : I \hookrightarrow I'$ *with* $l' \circ i = l$ *and* $r' \circ i = r$ *satisfies that* i *is an isomorphism then* (l, r) *is* canonical*, written* $(l, r) \in \mathcal{U}_{\mathrm{can}}$.

$$\begin{tikzcd}
G_1 & I \arrow[l, hook', "l"'] \arrow[r, hook, "r"] \arrow[d, hook, "i"'] & G_2 \\
 & I' \arrow[ul, hook', bend left, "l'"] \arrow[ur, hook, bend right, "r'"'] &
\end{tikzcd}$$

An update u_1 is a sub-update (see [14]) of u whenever the modifications defined by u_1 are fully contained in the modifications defined by u. Intuitively, this is the case when u_1 can be composed with another update u_2 such that (a) the resulting update has the same effect as u and (b) u_2 does not delete any element that was added before by u_1. This is stated, informally speaking, by requiring that I is the intersection (pullback) of I_1 and I_2 and that G_2 is its union (pushout).

Definition 4 (Sub-update [14]). *If* $u = (l : I \hookrightarrow G_1, r : I \hookrightarrow G_2) \in \mathcal{U}$, $u_1 = (l_1 : I_1 \hookrightarrow G_1, r_1 : I_1 \hookrightarrow G_3) \in \mathcal{U}$, $u_2 = (l_2 : I_2 \hookrightarrow G_3, r_2 : I_2 \hookrightarrow G_2) \in \mathcal{U}$,

$(r_1' : I \hookrightarrow I_1, l_2' : I \hookrightarrow I_2)$ is the pullback of (r_1, l_2), and (r_1, l_2) is the pushout of (r_1', l_2') then u_1 is a sub-update *of u, written $u_1 \leq^{u_2} u$ or simply $u_1 \leq u$.*

$$\begin{tikzcd}
G_1 & I_1 \arrow[l, "l_1"'] \arrow[r, hook, "r_1"] & G_2 & I_2 \arrow[l, "l_2"'] \arrow[r, hook, "r_2"] & G_3 \\
& & I \arrow[ull, bend left, "l"] \arrow[ul, bend left, "r_1'"'] \arrow[ur, bend right, "l_2'"] \arrow[urr, bend right, "r"'] & &
\end{tikzcd}$$

Moreover, we write $u_1 <^{u_2} u$ or $u_1 < u$ when $u_1 \leq^{u_2} u$ and not $u \leq u_1$.

We now define graph repairs as graph updates where the result graph satisfies the given consistency constraint ψ.

Definition 5 (Graph Repair). *If $u = (l : I \hookrightarrow G_1, r : I \hookrightarrow G_2) \in \mathcal{U}$, $\psi \in \Phi_\emptyset^{\mathrm{GC}}$, and $G_2 \models_{\mathrm{GC}} \psi$ then u is a* graph repair *or simply* repair *of G_1 with respect to ψ, written $u \in \mathcal{U}(G_1, \psi)$.*

To define a finite set of desirable repairs, we introduce the notion of least changing repairs that are repairs for which no sub-updates exist that are also repairs.

Definition 6 (Least Changing Graph Repair). *If $\psi \in \Phi_\emptyset^{\mathrm{GC}}$, $u = (l : I \hookrightarrow G_1, r : I \hookrightarrow G_2) \in \mathcal{U}(G_1, \psi)$, and there is no $u' \in \mathcal{U}(G_1, \psi)$ such that $u' < u$ then u is a* least changing graph repair *of G_1 with respect to ψ, written $u \in \mathcal{U}_{\mathrm{lc}}(G_1, \psi)$.*

Note that every least changing repair is canonical according to this definition. Moreover, the notion of least changing repairs is unrelated to other notions of repairs such as the set of all repairs that require a smallest amount of atomic modifications of the graph at hand to result in a graph satisfying the consistency constraint. For instance, a repair u_1 adding two nodes of type $:A$ may be a least changing repair even if there is a repair u_2 adding only one node of type $:B$.

A graph repair algorithm is *stable* [12], if the repair procedure returns the identity update $(\mathrm{id}_G : G \hookrightarrow G, \mathrm{id}_G : G \hookrightarrow G)$ when graph G is already consistent. Obviously, a graph repair algorithm that only returns least changing repairs is stable, since the identity update is a sub-update of any other repair.

4 State-Based Repair

In this section, we introduce two state-based graph repair algorithms (see [18] for additional technical detail), which compute a set of graph repairs restoring consistency for a given graph.

Definition 7 (State-Based Graph Repair Algorithm). *A state-based graph repair algorithm takes a graph G and a* GC *$\psi \in \Phi_\emptyset^{\mathrm{GC}}$ as inputs and returns a set of graph repairs in $\mathcal{U}(G, \psi)$.*

Note that the tool AutoGraph [17] can be used to verify this condition as follows: It determines the operation $\mathcal{A}$ that constructs a finite set of all minimal graphs satisfying a given GC ψ. Formally, $\mathcal{A}(\psi) = \cap\{S \subseteq [\![\psi]\!] \mid \forall G' \in [\![\psi]\!].\, \exists G \in$

$S. \exists m : G \hookrightarrow G'.true\}$. While AutoGraph may not terminate when computing this operation due to the inherent expressiveness of GCs, it is known that AutoGraph terminates whenever ψ is not satisfied by any graph.

The state-based algorithm $\mathcal{R}\text{epair}_{\text{sb},1}$ uses $\mathcal{A}$ to obtain repairs. $\mathcal{R}\text{epair}_{\text{sb},1}$ computes the set $\mathcal{A}(\psi \wedge \exists(\mathrm{i}_G, \mathit{true}))$ that contains all minimal graphs that (a) satisfy ψ and (b) include a copy of G. All these extensions of G correspond to a graph repair. For our running example, we do not obtain any repair for graph $\mathbf{G'_u}$ from Fig. 2 and GC $\boldsymbol{\psi}$ from Fig. 1 because the loop on node a_2 would invalidate any graph including $\mathbf{G'_u}$. We state that $\mathcal{R}\text{epair}_{\text{sb},1}$ indeed computes the non-deleting least changing graph repairs.

Theorem 1 (Functional Semantics of $\mathcal{R}\text{epair}_{\text{sb},1}$). *$\mathcal{R}\text{epair}_{\text{sb},1}$ is* sound*, i.e., $\mathcal{R}\text{epair}_{\text{sb},1}(G, \psi) \subseteq \mathcal{U}_{\text{lc}}(G, \psi)$, and* complete (upon termination) *with respect to non-deleting repairs in $\mathcal{U}_{\text{lc}}(G, \psi)$.*

The second state-based algorithm $\mathcal{R}\text{epair}_{\text{sb},2}$ computes *all* least changing graph repairs. In this algorithm we use the approach of $\mathcal{R}\text{epair}_{\text{sb},1}$ but compute $\mathcal{A}(\psi \wedge \exists(\mathrm{i}_{G_c}, \mathit{true}))$ whenever an inclusion $l : G_c \hookrightarrow G$ describes how G can be restricted to one of its subgraphs G_c. Every graph G' obtained from the application of $\mathcal{A}$ for one of these graphs G_c then results in one graph repair returned by $\mathcal{R}\text{epair}_{\text{sb},2}$ except for those that are not least changing.

To this extent we introduce the notion of a restriction tree (see example in Fig. 2) having all subgraphs G_c of a given graph G as nodes as long as they include the graph G_{min}, which is the empty graph in the state-based algorithm $\mathcal{R}\text{epair}_{\text{sb},2}$ but not in the algorithm $\mathcal{R}\text{epair}_{\text{db}}$ in Sect. 6, and where edges are given in this tree by inclusions that add precisely one node or edge.

Definition 8 (Restriction Tree RT). *If G and G_{min} are graphs and $S = \{l : G_c \hookrightarrow G_p \mid G_{min} \subseteq G_c \subset G_p \subseteq G, l \text{ is an inclusion}\}$, S' is the least subset of S such that the closure of S' under $\circ$ equals S then a* restriction tree $\mathrm{RT}(G, G_{min})$ *is a least subset of S' such that for all two inclusions $l_1 : G \hookrightarrow G_1 \in S'$ and $l_2 : G \hookrightarrow G_2 \in S'$ one of them is in $\mathrm{RT}(G, G_{min})$.*

Considering our running example, the restriction tree in Fig. 2 is traversed entirely except for the four graphs without a border, which are not traversed as they have the supergraph marked 9 satisfying $\boldsymbol{\psi}$ and therefore traversing those would generate repairs that are not least changing. The resulting graph repairs for the condition $\boldsymbol{\psi}$ are given by the graphs marked by 3–6.

Our second state-based graph repair algorithm is indeed sound and complete whenever the calls to AutoGraph using $\mathcal{A}$ terminate.

Theorem 2 (Functional Semantics of $\mathcal{R}\text{epair}_{\text{sb},2}$). *$\mathcal{R}\text{epair}_{\text{sb},2}$ is* sound*, i.e., $\mathcal{R}\text{epair}_{\text{sb},2}(G, \psi) \subseteq \mathcal{U}_{\text{lc}}(G, \psi)$, and* complete*, i.e., $\mathcal{U}_{\text{lc}}(G, \psi) \subseteq \mathcal{R}\text{epair}_{\text{sb},2}(G, \psi)$, upon termination.*

5 Satisfaction Trees

The state-based algorithms introduced in the previous section are inefficient when used in a scenario where a graph needs repair after a sequence of updates

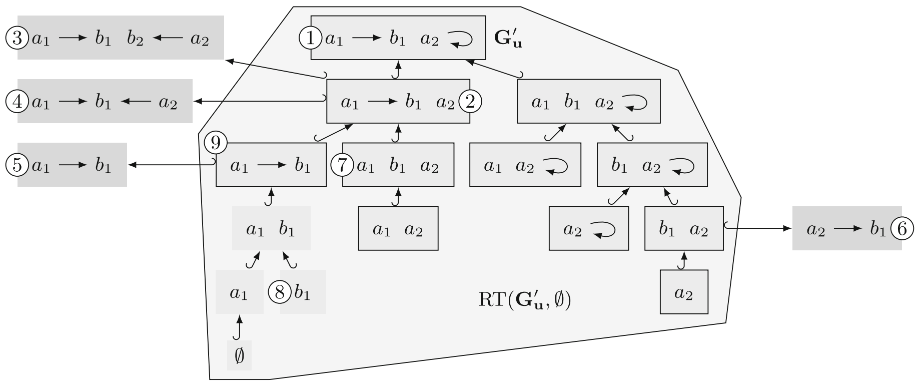


Fig. 2. The restriction tree $\mathrm{RT}(\mathbf{G}'_\mathbf{u}, \emptyset)$ (enclosed by the polygon) and four graph repairs (marked 3–6) generated using $\mathcal{R}\mathrm{epair}_{\mathrm{sb},2}$

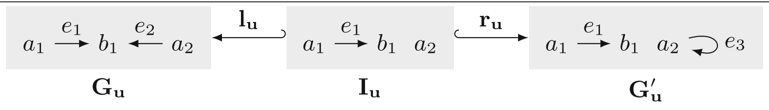
that all need repair. We thus present in Sect. 6 an incremental algorithm reducing the computational cost for a repair when an update is provided. This algorithm uses an additional data structure, called *satisfaction tree* or ST, which stores information on if and how a graph G satisfies a GC ψ (according to Definition 1). In this section, given ψ and G, we define how such an ST γ is constructed and how it is updated once the graph G is updated.

If ψ is a conjunction of conditions, its associated ST γ is a conjunction of STs and if ψ is a negation of a conditions, its associated γ is a negation of an ST. In the case when ψ is a $\exists(a : H \hookrightarrow H', \phi)$, recall that a match $m : H \hookrightarrow G$ satisfies ψ if there exists a $q : H' \hookrightarrow G$ such that $m = q \circ a$ and $q \models_{\mathrm{GC}} \phi$. For this case, we keep in ST each q satisfying these two conditions and also each q that satisfies the first condition, but not the second. More precisely, for the case of existential quantification, the corresponding ST is of the form $\exists(a : H \hookrightarrow H', \phi, m_t, m_f)$, where m_t and m_f are partial mappings (we use $\sup(f)$ to denoted the elements actually mapped by a partial map f) that map matches $q : H' \hookrightarrow G$ that satisfy $m = q \circ a$ (for a previously known $m : H \hookrightarrow G$) to an ST for the subcondition ϕ. The difference between both partial functions is that m_t maps matches q to STs for which $q \models_{\mathrm{GC}} \phi$ while m_f maps matches q to STs for which $q \not\models_{\mathrm{GC}} \phi$. Consider Fig. 3b for an example of an ST $\gamma_\mathbf{u}$.

The following definition describes the syntax of STs. The STs are defined over matches into a graph G to allow for the basic well-formedness condition that every mapped match q satisfies $q \circ a = m$.

Definition 9 (Satisfaction Trees (STs)). *The class of all* Satisfaction Trees Γ_m^{ST} *for a mono* $m : H \hookrightarrow G$ *contains* γ *if one of the following cases applies.*

- $\gamma = \wedge S$ *and* $S \subseteq_{\mathrm{fin}} \Gamma_m^{\mathrm{ST}}$.
- $\gamma = \neg\chi$ *and* $\chi \in \Gamma_m^{\mathrm{ST}}$.
- $\gamma = \exists(a, \phi, m_t, m_f)$, $a : H \hookrightarrow H'$, $\phi \in \Phi_{H'}^{\mathrm{GC}}$, $m_t, m_f \subseteq_{\mathrm{fin}} \{(q : H' \hookrightarrow G, \bar{\gamma}) \mid q \circ a = m, \bar{\gamma} \in \Gamma_q^{\mathrm{ST}}\}$, *and* m_t, m_f *are partial maps.*



$a_1 \xrightarrow{e_1} b_1 \xleftarrow{e_2} a_2$
$\mathbf{l_u}$
$a_1 \xrightarrow{e_1} b_1 \quad a_2$
$\mathbf{r_u}$
$a_1 \xrightarrow{e_1} b_1 \quad a_2 \circlearrowright e_3$
$\mathbf{G_u}$
$\mathbf{I_u}$
$\mathbf{G'_u}$

(a) A graph update $\mathbf{u} = (\mathbf{l_u} : \mathbf{I_u} \hookrightarrow \mathbf{G_u}, \mathbf{r_u} : \mathbf{I_u} \hookrightarrow \mathbf{G'_u})$

$$\gamma_{\mathbf{u}} = \neg\exists(a, \underline{\neg(\exists(a \xrightarrow{e} b, \mathit{true}) \land \neg\exists(a \circlearrowright e, \mathit{true}))}, \emptyset, \{a_2 \mapsto \gamma_{\mathbf{u,1}}, a_1 \mapsto \gamma_{\mathbf{u,2}}\})$$
$$\gamma_{\mathbf{u,1}} = \neg(\exists(a \xrightarrow{e} b, \underline{\mathit{true}}, \{a_2 \xrightarrow{e_2} b_1 \mapsto \mathit{true}\}, \emptyset) \land \neg\exists(a \circlearrowright e, \underline{\mathit{true}}, \emptyset, \emptyset))$$
$$\gamma_{\mathbf{u,2}} = \neg(\exists(a \xrightarrow{e} b, \underline{\mathit{true}}, \{a_1 \xrightarrow{e_1} b_1 \mapsto \mathit{true}\}, \emptyset) \land \neg\exists(a \circlearrowright e, \underline{\mathit{true}}, \emptyset, \emptyset))$$

(b) The ST $\gamma_{\mathbf{u}}$ for $\mathbf{G_u}$ (see Fig. 3a) and ψ (see Fig. 1).

$$\gamma^{\mathbf{I}}_{\mathbf{u}} = \neg\exists(a, \underline{\neg(\exists(a \xrightarrow{e} b, \mathit{true}) \land \neg\exists(a \circlearrowright e, \mathit{true}))}, \{a_2 \mapsto \gamma^{\mathbf{I}}_{\mathbf{u,1}}\}, \{a_1 \mapsto \gamma^{\mathbf{I}}_{\mathbf{u,2}}\})$$
$$\gamma^{\mathbf{I}}_{\mathbf{u,1}} = \neg(\exists(a \xrightarrow{e} b, \underline{\mathit{true}}, \emptyset, \emptyset) \land \neg\exists(a \circlearrowright e, \underline{\mathit{true}}, \{a_2 \circlearrowright e_3 \mapsto \mathit{true}\}, \emptyset))$$
$$\gamma^{\mathbf{I}}_{\mathbf{u,2}} = \neg(\exists(a \xrightarrow{e} b, \underline{\mathit{true}}, \{a_1 \xrightarrow{e_1} b_1 \mapsto \mathit{true}\}, \emptyset) \land \neg\exists(a \circlearrowright e, \underline{\mathit{true}}, \emptyset, \emptyset))$$

(c) The ST $\gamma^{\mathbf{I}}_{\mathbf{u}}$ for $\mathbf{I_u}$ (see Fig. 3a) and ψ (see Fig. 1) that is obtained as the backward propagation ppgB$(\gamma_{\mathbf{u}}, \mathbf{l_u})$ from $\gamma_{\mathbf{u}}$ (see Fig. 3b) and $\mathbf{l_u}$ (see Fig. 3a)

$$\gamma'_{\mathbf{u}} = \neg\exists(a, \underline{\neg(\exists(a \xrightarrow{e} b, \mathit{true}) \land \neg\exists(a \circlearrowright e, \mathit{true}))}, \{a_2 \overset{(\mathsf{R1})}{\mapsto} \gamma'_{\mathbf{u,1}}\}, \{a_1 \mapsto \gamma'_{\mathbf{u,2}}\})$$
$$\gamma'_{\mathbf{u,1}} = \neg(\exists(a \xrightarrow{e} b, \underline{\mathit{true}}, \emptyset_{(\mathsf{R2})}, \emptyset) \land \neg\exists(a \circlearrowright e, \underline{\mathit{true}}, \{a_2 \circlearrowright e_3 \overset{(\mathsf{R3})}{\mapsto} \mathit{true}\}, \emptyset))$$
$$\gamma'_{\mathbf{u,2}} = \neg(\exists(a \xrightarrow{e} b, \underline{\mathit{true}}, \{a_1 \xrightarrow{e_1} b_1 \mapsto \mathit{true}\}, \emptyset) \land \neg\exists(a \circlearrowright e, \underline{\mathit{true}}, \emptyset, \emptyset))$$

(d) The ST $\gamma'_{\mathbf{u}}$ for $\mathbf{G'_u}$ (see Fig. 3a) and ψ (see Fig. 1) that is obtained as the forward propagation ppgF$(\gamma^{\mathbf{I}}_{\mathbf{u}}, \mathbf{r_u})$ from $\gamma^{\mathbf{I}}_{\mathbf{u}}$ (see Fig. 3b) and $\mathbf{r_u}$ (see Fig. 3a). Also $\gamma'_{\mathbf{u}}$ is the result of ppgU$(\gamma_{\mathbf{u}}, \mathbf{u})$ that applies backward and forward propagation. The viable points for the delta-based repair discussed in Sec. 6 are indicated by (R1)–(R3).

Fig. 3. A graph update and an ST with its propagation over the graph update where GCs are underlined in STs for readability

The following satisfaction predicate $\models_{\mathrm{GC}}$ for STs defines when an ST γ for a mono m states that the contained GC ψ is satisfied by the morphism m.

Definition 10 (ST Satisfaction). *An* ST $\gamma \in \Gamma^{\mathrm{ST}}_{m:H \hookrightarrow G}$ *is* satisfied*, written* $\models_{\mathrm{ST}} \gamma$*, if one of the following cases applies.*

- $\gamma = \land S$ *and* $\models_{\mathrm{ST}} \chi$ *(for each* $\chi \in S$*)*
- $\gamma = \neg\chi$ *and* $\not\models_{\mathrm{ST}} \chi$*.*
- $\gamma = \exists(a, \phi, m_t, m_f)$ *and* $m_t \neq \emptyset$*.*

The following recursive operation constructs an ST γ for a graph G and a condition ψ so that γ represents how G satisfies (or not satisfies) ψ. Note that the match m in the definition of STs above and the construction of an ST below corresponds to the match $m : H \hookrightarrow G$ from Definition 1 that we operationalize in the following definition. For conjunction and negation, we construct the STs from the STs for the subconditions. For the case of existential quantification, we consider all morphisms $q : H' \hookrightarrow G$ for which the triangle $q \circ a = m$ commutes and construct the STs for the subcondition ϕ under this extended match q. The resulting STs are inserted into m_t and m_f according to whether they are satisfied.

Definition 11 (Construct ST (cst**)).** *Given* $m : H \hookrightarrow G$ *and* $\psi \in \Phi^{\mathrm{GC}}_H$*, we define* $\mathrm{cst}(\psi, m) = \gamma$*, with* $\gamma \in \Gamma^{\mathrm{ST}}_m$ *as follows.*

– *If* $\psi = \wedge S$ *then* $\gamma = \wedge\{\mathrm{cst}(\phi, m) \mid \phi \in S\}$.
– *If* $\psi = \neg\phi$ *then* $\gamma = \neg\, \mathrm{cst}(\phi, m)$.
– *If* $\psi = \exists(a : H \hookrightarrow H', \phi)$, $m_{all} = \{(q : H' \hookrightarrow G, \chi) \mid q \circ a = m, \mathrm{cst}(\phi, q) = \chi\}$, $m_t = \{(q, \chi) \in m_{all} \mid \models_{\mathrm{ST}} \chi\}$, $m_f = m_{all} \setminus m_t$, *then* $\gamma = \exists(a, \phi, m_t, m_f)$.

If G *is a graph and* $\psi \in \Phi^{\mathrm{GC}}_{\emptyset}$*, then* $\mathrm{cst}(\psi, G) = \mathrm{cst}(\psi, \mathrm{i}_G)$.

This construction of STs then ensures that $\models_{\mathrm{ST}} \gamma$ if and only if $G \models_{\mathrm{GC}} \psi$. Note that $\models_{\mathrm{ST}} \boldsymbol{\gamma_u}$ holds for the ST $\boldsymbol{\gamma_u}$ from Fig. 3b, the GC $\boldsymbol{\psi}$ from Fig. 1, and the graph $\mathbf{G_u}$ from Fig. 3.

Theorem 3 (Sound Construction of STs). *Given* $m : H \hookrightarrow G$, $\psi \in \Phi^{\mathrm{GC}}_H$*, and* $\mathrm{cst}(\psi, m) = \gamma$ *then* $\models_{\mathrm{ST}} \gamma$ *iff* $m \models_{\mathrm{GC}} \psi$.

Subsequently, we define a propagation operation ppgU of an ST γ for a graph update $u = (l : I \hookrightarrow G, r : I \hookrightarrow G')$ to obtain an ST γ' such that $\gamma' = \mathrm{cst}(\psi, G')$ whenever $\gamma = \mathrm{cst}(\psi, G)$. This overall propagation is performed by a backward propagation of γ for l using the operation ppgB followed by a forward propagation of the resulting ST for r using the operation ppgF.

For backward propagation, we describe how the deletion of elements in G by $l : I \hookrightarrow G$ affect its associated ST γ. To this end, we preserve those matches $q : H \hookrightarrow G$ for which no matched elements are deleted. This is formalized by requiring a mono $q' : H \hookrightarrow I$ such that $l \circ q' = q$. The matches q with deleted matched elements can not be preserved and are therefore removed.

Definition 12 (Propagate Match (ppgMatch**)).** *If* $q : H \hookrightarrow G$ *and* $l : I \hookrightarrow G$ *are monos, then* $\mathrm{ppgMatch}(q, l)$ *is the unique* $q' : H \hookrightarrow I$ *such that* $l \circ q' = q$ *if it exists and* $\perp$ *otherwise.*

The following recursive backward propagation defines how deletions affect the maps m_t and m_f of the given ST. That is, when $\gamma = \exists(a, \phi, m_t, m_f)$, we (a) entirely remove a mapping (m, χ) from m_t or m_f if $\mathrm{ppgMatch}(q, l) = \perp$ and (b) construct for a mapping (m, χ) from m_t or m_f the pair $(\mathrm{ppgMatch}(q, l), \chi')$ where χ' is obtained from recursively applying the backward propagation on χ when $\mathrm{ppgMatch}(q, l) \neq \perp$. The updated pair $(\mathrm{ppgMatch}(q, l), \chi')$ must be rechecked to decide to which partial map this pair must be added to ensure that the resulting ST corresponds to the ST that would be constructed for G' directly.

Definition 13 (Backward Propagation (ppgB)). *If $m : H \hookrightarrow G$, $\gamma \in \Gamma^{\mathrm{ST}}_{m}$, $l : I \hookrightarrow G$, $\mathrm{ppgMatch}(m, l) = m' : H \hookrightarrow I$, and $\gamma' \in \Gamma^{\mathrm{ST}}_{m'}$ then $\mathrm{ppgB}(\gamma, l) = \gamma'$ if one of the following cases applies.*

- $\gamma = \wedge S$ *and* $\gamma' = \wedge\{\mathrm{ppgB}(\chi, l) \mid \chi \in S\}$.
- $\gamma = \neg\chi$ *and* $\gamma' = \neg\,\mathrm{ppgB}(\chi, l)$.
- $\gamma = \exists(a, \phi, m_t, m_f)$, $m_{all} = \{(q', \chi') \mid (q, \chi) \in m_t \cup m_f \wedge \mathrm{ppgMatch}(q, l) = q' \neq \bot \wedge \mathrm{ppgB}(\chi, l) = \chi'\}$, $m'_t = \{(q, \chi) \in m_{all} \mid\models_{\mathrm{ST}} \chi\}$, $m'_f = m_{all} \setminus m'_t$, *and* $\gamma' = \exists(a, \phi, m'_t, m'_f)$.

Note that $\mathrm{ppgMatch}(\mathrm{i}_G, l) = \mathrm{i}_G$ and, hence, the operation ppgB is applicable for all ST $\gamma \in \Gamma^{\mathrm{ST}}_{\mathrm{i}_G}$, which is sufficient as we define consistency constraints using GCs over the empty graph as well.

In the case of forward propagation where additions are given by $r : I \hookrightarrow G'$ we can preserve all matches using an adaptation. But the addition of further elements may result in additional matches as well that may satisfy the conditions to be included in the corresponding m_t and m_f from the ST at hand.

Definition 14 (Forward Propagation (ppgF)). *If $\gamma \in \Gamma^{\mathrm{ST}}_{m:H\hookrightarrow I}$, $r : I \hookrightarrow G'$, and $\gamma' \in \Gamma^{\mathrm{ST}}_{r\circ m}$ then $\mathrm{ppgF}(\gamma, r) = \gamma'$ if one of the following cases applies.*

- $\gamma = \wedge S$ *and* $\gamma' = \wedge\{\mathrm{ppgF}(\chi, r) \mid \chi \in S\}$.
- $\gamma = \neg\chi$ *and* $\gamma' = \neg\,\mathrm{ppgF}(\chi, r)$.
- $\gamma = \exists(a, \phi, m_t, m_f)$, $m_{all} = \{(r \circ q, \gamma') \mid (q, \chi) \in m_t \cup m_f \wedge \mathrm{ppgF}(\chi, r) = \gamma'\} \cup \{(q, \gamma_q) \mid q \circ a = r \circ m, (\nexists q' \in \sup(m_t) \cup \sup(m_f).\ r \circ q' = q), \mathrm{cst}(\phi, q) = \gamma_q\}$, $m'_t = \{(q, \chi) \in m_{all} \mid\models_{\mathrm{ST}} \chi\}$, $m'_f = m_{all} \setminus m'_t$, *and* $\gamma' = \exists(a, \phi, m'_t, m'_f)$.

We now define the composition of both propagations to obtain the operation ppgU that updates an ST for an entire graph update.

Definition 15 (Update Propagation (ppgU)). *If $m : H \hookrightarrow G$, $\gamma \in \Gamma^{\mathrm{ST}}_{m}$, $l : I \hookrightarrow G$, $\mathrm{ppgMatch}(m, l) = m' : H \hookrightarrow G'$, and $r : I \hookrightarrow G'$ then $\mathrm{ppgU}(\gamma, (l, r)) = \mathrm{ppgF}(\mathrm{ppgB}(\gamma, l), r) \in \Gamma^{\mathrm{ST}}_{m'}$.*

The overall propagation given by this operation is *incremental*, in the sense that the operation cst is only used in the forward propagation on parts of the graph G', where the addition of graph elements by r from the graph update results in additional matches q according to the satisfaction relation for GCs. Finally, we state that ppgU incrementally computes the ST obtained using cst. The proof of this theorem relies on the fact that this property also holds for ppgB and ppgF.

Theorem 4 (ppgU is Compatible with cst). *If G is a graph, $\psi \in \Phi^{\mathrm{GC}}_{\emptyset}$, $l : I \hookrightarrow G$, and $r : I \hookrightarrow G'$ then $\mathrm{ppgU}(\mathrm{cst}(\psi, G), (l, r)) = \mathrm{cst}(\psi, G')$.*

6 Delta-Based Repair

The local states of delta-based graph repair algorithms may contain, besides the current graph as in state-based graph repair algorithms, an additional value. In our delta-based graph repair algorithm this will be an ST.

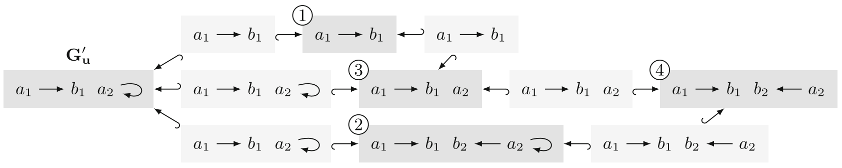


Fig. 4. An example for delta-based graph repair using $\mathcal{R}\mathrm{epair}_{\mathrm{db}}$

Definition 16 (Delta-Based Graph Repair Algorithm). *Delta-based graph repair algorithms take a graph G, a* GC *$\psi \in \Phi^{\mathrm{GC}}_{\emptyset}$, and a value q as inputs and return a set of pairs (u, q') where $u \in \mathcal{U}(G, \psi)$ is a graph repair and q' is a value.*

Our delta-based graph repair algorithm $\mathcal{R}\mathrm{epair}_{\mathrm{db}}$ will be based on the single step operation $\mathcal{R}\mathrm{epair}_{\mathrm{db1}}$. Given a graph G, a GC $\psi \in \Phi^{\mathrm{GC}}_{\emptyset}$, the ST γ that equals $\mathrm{cst}(\psi, G)$, and a graph update $u = (l : I \hookrightarrow G, r : I \hookrightarrow G')$, the single step operation $\mathcal{R}\mathrm{epair}_{\mathrm{db}}$ first updates γ using ppgU for the graph update u and then determines using $\mathcal{R}\mathrm{epair}_{\mathrm{db1}}$, if necessary, graph repairs for the resulting ST γ' according to the repair rules described in the following. The algorithm $\mathcal{R}\mathrm{epair}_{\mathrm{db}}$ then uses $\mathcal{R}\mathrm{epair}_{\mathrm{db1}}$ in a breadth first manner to obtain multi-step repairs.

For our example from Fig. 3a, such a multi-step repair of $\mathbf{G'_u}$ is given in Fig. 4 where the graph updates are obtained resulting in the graphs marked 1–3, of which only the graph marked 1 satisfies $\boldsymbol{\psi}$. The algorithm $\mathcal{R}\mathrm{epair}_{\mathrm{db}}$ then computes further graph updates resulting in the graph marked 4 also satisfying $\boldsymbol{\psi}$.

The operation $\mathcal{R}\mathrm{epair}_{\mathrm{db1}}$ for deriving single-step repairs depends on two local modifications. Firstly, a GC $\exists(a : H \hookrightarrow H', \phi)$ occurring as a subcondition in the consistency constraint ψ may be violated because, for the match $m : H \hookrightarrow G$ that locates a copy of H in the graph G under repair, no suitable match $q : H' \hookrightarrow G$ can be found for which $q \circ a = m$ and $q \models_{\mathrm{GC}} \phi$ are satisfied. The operation $\mathcal{R}\mathrm{epair}_{\mathrm{add}}$ resolves this violation by (a) using AutoGraph to construct a suitable graph H_s and by (b) integrating this graph H_s into G resulting in G' such that a suitable match $q : H' \hookrightarrow G'$ can be found.

Definition 17 (Local Addition Operation $\mathcal{R}\mathrm{epair}_{\mathrm{add}}$). *If $a : H \hookrightarrow H'$, $\phi \in \Phi^{\mathrm{GC}}_{H'}$, $m : H \hookrightarrow G$, $H_s \in \mathcal{A}(\exists(\mathrm{i}_H, \exists(a, \phi)))$, $k : H \hookrightarrow H_s$, and $(\bar{m} : H_s \hookrightarrow G', r : G \hookrightarrow G')$ is the pushout of (m, k) then $r \in \mathcal{R}\mathrm{epair}_{\mathrm{add}}(a, \phi, m)$.*

$$\begin{tikzcd}
H' & H \arrow[l, hook', "a"'] \arrow[r, hook, "k"] \arrow[d, hook, "m"'] & H_s \arrow[d, hook, "\bar{m}"] \\
 & G \arrow[r, hook, "r"] & G'
\end{tikzcd}$$

In our running example, $\mathcal{R}\mathrm{epair}_{\mathrm{add}}$ determines a graph repair resulting in the graph marked 2 in Fig. 4. For this repair, we considered the sub-ST marked by (R2) in Fig. 3d, where the morphism m matches the node a from $\boldsymbol{\psi}$ to the node a_2 in $\mathbf{G'_u}$, but where no extension of m can also match a node $:B$ and an edge between these two nodes. The repair performed then uses $a \xrightarrow{e} b$ for the graph H_s, resulting in the addition of the node b_2 and the edge from a_2 to b_2.

Secondly, a GC $\exists(a : H \hookrightarrow H', \phi)$ occurring as a subcondition in the consistency constraint ψ may be satisfied even though it should not when occurring underneath some negation. Such a violation is determined, again for a given match $m : H \hookrightarrow G$, by some match $q : H' \hookrightarrow G$ satisfying $q \circ a = m$ and $q \models_{\mathrm{GC}} \phi$. The local repair operation $\mathcal{R}\mathrm{epair}_{\mathrm{del}}$ repairs such an undesired satisfaction by selecting a graph H_p such that $H \subseteq H_p \subset H'$ using a restriction tree (see Definition 8) and deleting $G_{del} = q(H') \setminus q(H_p)$ from G. Technically, we can not use the pushout complement of a' and q as it does not exists when edges from $G \setminus G_{del}$ are attached to nodes in G_{del}. Hence, we determine the pushout complement of a'' and k', which must be constructed for this purpose suitably.

Definition 18 (Local Deletion Operation $\mathcal{R}\mathrm{epair}_{\mathrm{del}}$). *If $a : H \hookrightarrow H'$, $q : H' \hookrightarrow G$, $a' : H_p \hookrightarrow H' \in \mathrm{RT}(H', H)$, $m_1 : H' \hookrightarrow X_2$ where X_2 is obtained from $q(H')$ by adding all edges (with their nodes) that are connected to nodes in $q(H') \setminus q(a'(H_p))$, $k' : X_2 \hookrightarrow G$ is obtained such that $k' \circ m_1 = q$, $m_2 : H_p \hookrightarrow X_1$ where X_1 is obtained from H_p by adding all nodes in $X_2 \setminus q(H')$, $a'' : X_1 \hookrightarrow X_2$ is obtained such that $a'' \circ m_2 = m_1 \circ a'$, and $(l : G' \hookrightarrow G, m' : X_1 \hookrightarrow G')$ is the pushout complement of (a'', k') then $l \in \mathcal{R}\mathrm{epair}_{\mathrm{del}}(a, q)$.*

$$\begin{tikzcd}
H \arrow[r, hook, "a"] \arrow[ddrr, bend right, "m"'] & H' \arrow[dr, hook, "m_1"] \arrow[ddr, hook, "q"'] & & H_p \arrow[ll, hook', "a'"'] \arrow[d, hook, "m_2"] \\
& & X_2 \arrow[d, hook, "k'"] & X_1 \arrow[l, hook', "a''"'] \arrow[d, hook, "m'"] \\
& & G & G' \arrow[l, hook', "l"']
\end{tikzcd}$$

In our example, $\mathcal{R}\mathrm{epair}_{\mathrm{del}}$ determines a repair resulting in the graph marked 1 in Fig. 4. For this repair, we considered the sub-ST marked by (R1) in Fig. 3d where the mono m matches the node a from $\boldsymbol{\psi}$ to the node a_2 in $\mathbf{G'_u}$. The repair performed then uses $H_p = \emptyset$ for the removal of the node a_2 along with its adjacent loop (for which the technical handling in $\mathcal{R}\mathrm{epair}_{\mathrm{del}}$ is required).

The recursive operation $\mathcal{R}\mathrm{epair}_{\mathrm{db1}}$ below derives updates from an ST γ that corresponds to the current graph G (for our running example, these are $\gamma'_{\mathbf{u}}$ and $\mathbf{G'_u}$ from Fig. 3d). In the algorithm $\mathcal{R}\mathrm{epair}_{\mathrm{db}}$, we apply $\mathcal{R}\mathrm{epair}_{\mathrm{db1}}$ for the initial match i_G, γ, and *true* where this boolean indicates that we want γ to be satisfied. This boolean is changed in Rule 3 whenever the recursion is applied to an ST $\neg\gamma'$ because we expect that γ' is not to be satisfied iff we expect that $\neg\gamma'$ is to be satisfied. For conjunction, we either attempt to repair a sub-ST for $b = \mathit{true}$ in Rule 1 or we attempt to break one sub-ST for $b = \mathit{false}$. For existential quantification and $b = \mathit{true}$, we use $\mathcal{R}\mathrm{epair}_{\mathrm{add}}$ as discussed before in Rule 4 or we attempt to repair one existing match contained in m_f in Rule 5. Also, for existential quantification and $b = \mathit{false}$, we use $\mathcal{R}\mathrm{epair}_{\mathrm{del}}$ as discussed before in Rule 6 or we attempt to break one existing match contained in m_t in Rule 7.

Definition 19 (Single-Step Delta-Based Repair Algorithm $\mathcal{R}\mathrm{epair}_{\mathrm{db1}}$). *If $m : H \hookrightarrow G$, $\gamma \in \Gamma^{\mathrm{ST}}_m$, and $b \in \mathbf{B}$ then $(l : I \hookrightarrow G, r : I \hookrightarrow G') \in \mathcal{R}\mathrm{epair}_{\mathrm{db1}}(m, \gamma, b)$ if one of the following cases applies.*

- Rule 1 (repair one subcondition of a conjunction):
  $b = \mathit{true}, \gamma = \wedge S,\ \chi \in S,\ \not\models_{\mathrm{ST}} \chi,\ (l, r) \in \mathcal{R}\mathrm{epair}_{\mathrm{db1}}(m, \chi, b)$.
- Rule 2 (break one subcondition of a conjunction):
  $b = \mathit{false}, \gamma = \wedge S,\ \chi \in S,\ \models_{\mathrm{ST}} \chi,\ (l, r) \in \mathcal{R}\mathrm{epair}_{\mathrm{db1}}(m, \chi, b)$.
- Rule 3 (repair/break the subcondition of a negation):
  $\gamma = \neg\chi,\ (l, r) \in \mathcal{R}\mathrm{epair}_{\mathrm{db1}}(m, \chi, \neg b)$.
- Rule 4 (repair an existential quantification by local extension):
  $b = \mathit{true}, \gamma = \exists(a, \phi, m_t, m_f),\ m_t = \emptyset,\ r \in \mathcal{R}\mathrm{epair}_{\mathrm{add}}(a, \phi, m),\ l = \mathrm{id}_G$.
- Rule 5 (repair an existential quantification recursively):
  $b = \mathit{true}, \gamma = \exists(a, \phi, m_t, m_f),\ m_t = \emptyset,\ m_f(k) = \chi,\ (l, r) \in \mathcal{R}\mathrm{epair}_{\mathrm{db1}}(k, \chi, b)$.
- Rule 6 (break an existential quantification by local removal):
  $b = \mathit{false}, \gamma = \exists(a, \phi, m_t, m_f),\ m_t(k) \neq \bot,\ l \in \mathcal{R}\mathrm{epair}_{\mathrm{del}}(a, k),\ r = \mathrm{id}_{G'}$.
- Rule 7 (break an existential quantification recursively):
  $b = \mathit{false}, \gamma = \exists(a, \phi, m_t, m_f),\ m_t(k) = \chi,\ (l, r) \in \mathcal{R}\mathrm{epair}_{\mathrm{db1}}(k, \chi, b)$.

We define the recursive algorithm $\mathcal{R}\mathrm{epair}_{\mathrm{db}}$ to apply $\mathcal{R}\mathrm{epair}_{\mathrm{db1}}$ to obtain repairs as iterated applications of single-step repairs computed by $\mathcal{R}\mathrm{epair}_{\mathrm{db1}}$.

Definition 20 (Delta-Based Repair Algorithm $\mathcal{R}\mathrm{epair}_{\mathrm{db}}$). *If* $u = (l : I \hookrightarrow G, r : I \hookrightarrow G') \in \mathcal{U}$, $\gamma \in \Gamma^{\mathrm{ST}}_{\mathrm{i}_G}$, *and* $\gamma' = \mathrm{ppgU}(\gamma, u)$ *then* $\mathcal{R}\mathrm{epair}_{\mathrm{db}}(u, \gamma) = S$ *if one of the following cases applies.*

- $\models_{\mathrm{ST}} \gamma'$ *and* $S = \{((\mathrm{id}_{G'}, \mathrm{id}_{G'}), \gamma')\}$.
- $\not\models_{\mathrm{ST}} \gamma'$, $S' = \{(u', \mathrm{ppgU}(\gamma', u')) \mid u' \in \mathcal{R}\mathrm{epair}_{\mathrm{db1}}(\mathrm{i}_G, \gamma', \mathit{true})\}$, *and* $S = \{(u', \gamma') \in S' \mid \models_{\mathrm{ST}} \gamma'\} \cup \bigcup\{(u'' \circ u', \gamma'') \mid (u', \gamma') \in S', \not\models_{\mathrm{ST}} \gamma', (u'', \gamma'') \in \mathcal{R}\mathrm{epair}_{\mathrm{db}}(u', \gamma'), u'' \circ u' \neq \bot\}$.[3]

This computation does not terminate when repairs trigger each other ad infinitum. However, a breadth-first-computation of $\mathcal{R}\mathrm{epair}_{\mathrm{db}}$ gradually computes a set of sound repairs. Obviously, GCs that trigger such nonterminating computations should be avoided but machinery for detecting such GCs is called for.

Note that the algorithm $\mathcal{R}\mathrm{epair}_{\mathrm{db}}$ computes fewer graph repairs compared to $\mathcal{R}\mathrm{epair}_{\mathrm{sb},2}$ because repairs are applied locally in the scope defined by the GC ψ. For example, no repair would be constructed resulting in the graph marked 4 in Fig. 2. In general, explicitly also using bigger contexts in ψ results in the additional computation of less–local graph repairs. For example, the condition $\boldsymbol{\psi}$ may be rephrased into $\boldsymbol{\psi}' = \boldsymbol{\psi} \wedge \neg\exists(a\ \ b, \neg\exists(a \xrightarrow{e} b, \mathit{true}))$ to also obtain the graph repair marked 4 in Fig. 2. We now define the updates, which we expect to be computed by $\mathcal{R}\mathrm{epair}_{\mathrm{db1}}$, as those that repair a single violation of the GC ψ by defining a local update to be embeddable into the resulting update via a double pushout diagram as in the DPO approach to graph transformation [16].

Definition 21 (Locally Least Changing Graph Update). *If* G_1 *is a graph,* $\psi \in \Phi^{\mathrm{GC}}_{\emptyset}$, $G_1 \not\models_{\mathrm{GC}} \psi$, $(l : I \hookrightarrow G_1, r : I \hookrightarrow G_2) \in \mathcal{U}_{\mathrm{lc}}(G_1, \psi)$, $G_2 \models_{\mathrm{GC}} \psi$, X_1 *is a minimal subgraph of* G_1 *with a violation of* ψ *that is also a violation of* ψ *in*

[3] If u_1 and u_2 are updates then $u_1 \circ u_2 = u$ if $u_1 \leq^{u_2} u$ or $u = \bot$ otherwise (see Definition 4).

G, and the diagram below exists and the right part of it is a DPO diagram then (l, r) is a locally least changing graph update.

$$\begin{tikzcd}
X_1 \arrow[d, hook] & I' \arrow[l, hook'] \arrow[r, hook] \arrow[d, hook] & X_2 \arrow[d, hook] \\
G_1 & I \arrow[l, hook', "l"'] \arrow[r, hook, "r"] & G_2
\end{tikzcd}$$

$\mathcal{R}$epair$_{\text{db1}}$ indeed generates such locally least changing graph updates because the graph X_1 in this definition corresponds to the H_1 and the H_2 from an ST $\exists(a : H_1 \hookrightarrow H_2, \phi, m_t, m_f)$ that is subject to $\mathcal{R}$epair$_{\text{add}}$ and $\mathcal{R}$epair$_{\text{del}}$, respectively. For example, for $\mathcal{R}$epair$_{\text{add}}$, the graph H_1 in the ST determines a subgraph in G_1 that is a violation of the overall consistency condition given by a GC ψ as its match can not be extended to the graph H_2.

We now define the locally least changing graph repairs (which are to be computed by $\mathcal{R}$epair$_{\text{db}}$ such as for example the graphs marked 1 and 4 in Fig. 4) as the composition of a sequence of locally least changing updates where precisely the last graph update results in a graph satisfying the GC ψ.

Definition 22 (Locally Least Changing Graph Repair). *If G_1 is a graph, $\psi \in \Phi_{\emptyset}^{\text{GC}}$, $\pi = (l_1 : I_1 \hookrightarrow G_1, r_1 : I_1 \hookrightarrow G_2) \dots (l_n : I_n \hookrightarrow G_n, r_n : I_n \hookrightarrow G_{n+1})$ is a sequence of locally least changing graph updates, $G_1 \in [\![\psi]\!]$ implies $n = 0$ and $l_1 = r_1 = \text{id}_{G_1}$, $G_i \notin [\![\psi]\!]$ (for each $2 \le i \le n$), $G_{n+1} \in [\![\psi]\!]$, (l, r) is the iterated composition of the updates in π, and $(l, r) \in \mathcal{U}(G_1, \psi)$ is a least changing graph repair then (l, r) is a* locally least changing graph repair.

We now state that our delta-based graph repair algorithm $\mathcal{R}$epair$_{\text{db}}$ returns all desired locally least changing graph repairs upon termination.

Theorem 5 (Functional Semantics of $\mathcal{R}$epair$_{\text{db}}$). *$\mathcal{R}$epair$_{\text{db}}$ is sound (i.e., it generates only locally least changing graph repairs) and complete (upon termination) with respect to locally least changing graph repairs.*

The state-based algorithms $\mathcal{R}$epair$_{\text{sb,1}}$ and $\mathcal{R}$epair$_{\text{sb,2}}$ are inappropriate in environments where numerous updates that may invalidate consistency are applied to a large graph because the procedure of AutoGraph has exponential cost. The incremental delta-based algorithm $\mathcal{R}$epair$_{\text{db}}$ is a viable alternative when additional memory requirements for storing the ST are acceptable. The AutoGraph applications for this algorithm have negligible costs because they may be performed a priori and must only be performed for subconditions of the consistency constraint, which can be assumed to feature reasonably small graphs only.

Finally, a classification of locally least changing repairs is useful for user-based repair selection. Delta preserving repairs defined below represent such a basic class, containing only those repairs that preserve the update resulting in a graph not satisfying GC ψ, i.e., it may be desirable to avoid repairs that revert additions or deletions of this update. In our example, the repair related to the graph marked 4 in Fig. 4 is not delta preserving w.r.t. **u** from Fig. 3a.

Definition 23 (Delta Preserving Graph Repair). *If $\psi \in \Phi_{\emptyset}^{\text{GC}}$, $u_2 = (l_2 : I_2 \hookrightarrow G_2, r_2 : I_2 \hookrightarrow G_3) \in \mathcal{U}(G_2, \psi)$ is a graph repair, $u_1 = (l_1 : I_1 \hookrightarrow G_1, r_1 :$*

$I_1 \hookrightarrow G_2$) is a graph update, and there exists a graph update u such that $u_1 <^{u_2} u$ then u_2 is a delta preserving graph repair *with respect to u_1.*

7 Related Work

According to the recent survey on *model repair* [12], and the corresponding exhaustive classification of primary studies selected in the literature review, published online [11], we can see that the amount and wide variety of existing approaches makes a detailed comparison with all of them infeasible.

We consider our approach to be innovative, not only because of the proposed solutions, but because it addresses the issues of *completeness* and *least changing* for incremental graph repair in a precise and formal way. From the survey [11,12] we can see that only two other approaches [10,19] address completeness and least changing, relying also on constraint-solving technology. The main difference with our approach is that they are not incremental. In particular, the work of Schoenboeck et al. [19] proposes a logic programming approach allowing the exploration of model repair solutions ranked according to some quality criteria, re-establishing conformance of a model with its metamodel. Soundness and completeness of these repair actions is not formally proven. Moreover, the least changing bidirectional model transformation approach of Macedo et al. [10] has only a bounded search for repairs, relying on a bounded constraint solver.

Some *recent work* on rule-based *graph repair* [9] (not covered by the survey) addresses the least-changing principle by developing so-called maximally preserving (items are preserved whenever possible) repair programs. This state-based approach considers a subset of consistency constraints (up to nesting depth 2) handled by our approach, and is not complete, since it produces repairs including only a minimal amount of deletions. Some other recent rule-based graph repair approach [13,20] (also not covered by the survey) proposes so-called change preserving repairs (similar to what we define as delta-preserving). The main difference with our work is that we do not require the user to specify consistency-preserving operations from which repairs are generated, since we derive repairs using constraint solving techniques directly from the consistency constraints.

Finally, there is a variety of work on *incremental evaluation of graph queries* (see e.g. [2,4]), developed with the aim of efficiently re-evaluating a graph query after an update has been performed. Although not employed with the specific aim of complete and least changing graph repair, this work is related to our newly introduced concept of satisfaction trees, also using specific data structures to record with some detail the set of answers to a given query (as described for graph conditions, for example, also in [3]). It is part of ongoing work to evaluate how STs can be employed similarly in this field of incremental query evaluation.

8 Conclusion and Future Work

We presented a logic-based incremental approach to graph repair. It is the first approach to graph repair returning a sound and complete overview of least changing repairs with respect to graph conditions equivalent to first-order logic on graphs. Technically, it relies on an existing model generation procedure for graph conditions together with the newly introduced concept of satisfaction trees, encoding if and how a graph satisfies a graph condition.

As future work, we aim at supporting partial consistency and gradually improving it. We are confident that we can extend our work to support attributes, since our underlying model generation procedure supports it. Ongoing work is the support of more expressive consistency constraints, allowing path-related properties. Moreover, we are in the process of implementing the algorithms presented here and evaluating them on a variety of case studies. The evaluation also pertains to the overall efficiency (for which we employ techniques for localized pattern matching) and includes a comparison with other approaches for graph repair. Finally, we aim at presenting new and refined properties distinguishing between all possible repairs supporting the implementation of interactive repair selection procedures.

References

1. Angles, R., Gutiérrez, C.: Survey of graph database models. ACM Comput. Surv. **40**(1), 1:1–1:39 (2008). https://doi.org/10.1145/1322432.1322433
2. Bergmann, G., Ökrös, A., Ráth, I., Varró, D., Varró, G.: Incremental pattern matching in the viatra model transformation system. In: GRaMoT, pp. 25–32. ACM (2008). https://doi.org/10.1145/1402947.1402953
3. Beyhl, T., Blouin, D., Giese, H., Lambers, L.: On the operationalization of graph queries with generalized discrimination networks. In: Echahed, R., Minas, M. (eds.) ICGT 2016. LNCS, vol. 9761, pp. 170–186. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40530-8_11
4. Beyhl, T., Giese, H.: Incremental view maintenance for deductive graph databases using generalized discrimination networks. In: GaM@ETAPS, EPTCS, vol. 231, pp. 57–71 (2016). https://doi.org/10.4204/EPTCS.231.5
5. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Rozenberg [16], pp. 313–400
6. Diskin, Z., König, H., Lawford, M.: Multiple model synchronization with multiary delta lenses. In: Russo, A., Schürr, A. (eds.) FASE 2018. LNCS, vol. 10802, pp. 21–37. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89363-1_2
7. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-31188-2
8. Habel, A., Pennemann, K.: Correctness of high-level transformation systems relative to nested conditions. MSCS **19**(2), 245–296 (2009). https://doi.org/10.1017/S0960129508007202
9. Habel, A., Sandmann, C.: Graph repair by graph programs. In: Mazzara, M., Ober, I., Salaün, G. (eds.) STAF 2018. LNCS, vol. 11176, pp. 431–446. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04771-9_31
10. Macedo, N., Cunha, A.: Least-change bidirectional model transformation with QVT-R and ATL. Softw. Syst. Model. **15**(3), 783–810 (2016). https://doi.org/10.1007/s10270-014-0437-x

11. Macedo, N., Tiago, J., Cunha, A.: Systematic literature review of model repair approaches. http://tinyurl.com/hv7eh6h. Accessed 14 Nov 2018
12. Macedo, N., Tiago, J., Cunha, A.: A feature-based classification of model repair approaches. IEEE Trans. Softw. Eng. **43**(7), 615–640 (2017). https://doi.org/10.1109/TSE.2016.2620145
13. Ohrndorf, M., Pietsch, C., Kelter, U., Kehrer, T.: Revision: a tool for history-based model repair recommendations. In: ICSE, pp. 105–108. ACM (2018). https://doi.org/10.1145/3183440.3183498
14. Orejas, F., Boronat, A., Ehrig, H., Hermann, F., Schölzel, H.: On propagation-based concurrent model synchronization. ECEASST **57** (2013). http://journal.ub.tu-berlin.de/eceasst/article/view/871
15. Rensink, A.: Representing first-order logic using graphs. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 319–335. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30203-2_23
16. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations. World Scientific (1997)
17. Schneider, S., Lambers, L., Orejas, F.: Automated reasoning for attributed graph properties. STTT **20**(6), 705–737 (2018). https://doi.org/10.1007/s10009-018-0496-3
18. Schneider, S., Lambers, L., Orejas, F.: A logic-based incremental approach to graph repair. Technical report, 126, Hasso Plattner Institute at the University of Potsdam, Potsdam, Germany (2019)
19. Schoenboeck, J., et al.: CARE - A constraint-based approach for re-establishing conformance-relationships. In: APCCM 2014, vol. 154, pp. 19–28. Australian Computer Society (2014). http://crpit.com/abstracts/CRPITV154Schoenboeck.html
20. Taentzer, G., Ohrndorf, M., Lamo, Y., Rutle, A.: Change-preserving model repair. In: Huisman, M., Rubin, J. (eds.) FASE 2017. LNCS, vol. 10202, pp. 283–299. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54494-5_16