Loose Graph Simulations

Alessio Mansutti[1(✉)], Marino Miculan[1], and Marco Peressotti[2]

[1] Department of Mathematics, Computer Science and Physics, University of Udine, Udine, Italy
alessio.mansutti@lsv.fr, marino.miculan@uniud.it
[2] Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark
peressotti@imada.sdu.dk

Abstract. We introduce *loose graph simulations* (LGS), a new notion about labelled graphs which subsumes in an intuitive and natural way *subgraph isomorphism* (SGI), *regular language pattern matching* (RLPM) and *graph simulation* (GS). Being a unification of all these notions, LGS allows us to express directly also problems which are "mixed" instances of previous ones, and hence which would not fit easily in any of them. After the definition and some examples, we show that the problem of finding loose graph simulations is NP-complete, we provide formal translation of SGI, RLPM, and GS into LGSs, and we give the representation of a problem which extends both SGI and RLPM. Finally, we identify a subclass of the LGS problem that is polynomial.

1 Introduction

Graph pattern matching is the problem of finding patterns satisfying a specific property, inside a given graph. This problem arises naturally in many research fields: for instance, in computer science it is used in automatic system verification, network analysis and data mining [5,15,25,28]; in computational biology it is applied to protein sequencing [24]; in cheminformatics it is used to study molecular systems and predict their evolution [1,4]. As a consequence, many definitions of patterns have been proposed; for instance, these patterns can be specified by another graph, by a formal language, by a logical predicate, etc. This situation has led to different notions of graph pattern matching, such as *subgraph isomorphism* (SGI), *regular language pattern matching* (RLPM) and *graph simulation* (GS). Each of these notions has been studied in depth, yielding similar but different theories, algorithms and tools.

A drawback of this situation is that it is difficult to deal with matching problems which do not fit directly in any of these variants. In fact, often we need

M. Miculan—Partially supported by PRID 2017 *ENCASE* of the University of Udine.

M. Peressotti—Partially supported by the Open Data Framework project at the University of Southern Denmark, and by the Independent Research Fund Denmark, Natural Sciences, grant no. DFF-7014-00041.

M. Seidl and S. Zschaler (Eds.): STAF 2017 Workshops, LNCS 10748, pp. 109–126, 2018.
https://doi.org/10.1007/978-3-319-74730-9_9

to search for patterns that can be expressed as compositions of several graph pattern matching notions. An example is when we have to find a pattern which has to satisfy multiple notions of graph pattern matching at once; due to the lack of proper tools, these notions can only be checked one by one with a worsening of the performances. Another example can be found in [9], where extensions of RLPM and their application in network analysis and graph databases are discussed. A mixed problem between SGI and RLPM is presented in [2].

This situation would benefit from a more general notion of graph pattern matching, able to subsume naturally the more specific ones find in literature. This general notion would be a common ground to study specific problems and their relationships, as well as to develop common techniques for them. Moreover, a more general pattern matching notion would pave the way for more general algorithms, which would deal more efficiently with "mixed" problems.

To this end, in this paper we propose a new notion about labelled graphs, called *loose graph simulation* (LGS, Sect. 2). The semantics of its pattern queries allow us to check properties from different classical notions of pattern matching, at once and without cumbersome encodings. LGS queries have a natural graphical representation that simplifies the understanding of their semantic; moreover, they can be composed using a sound and complete algebra (Sect. 3). Various notions of graph pattern matching can be naturally reduced to LGSs, as we will formally prove in Sects. 4, 5 and 6; in particular, the encoding of subgraph isomorphism allows us to prove that computing LGSs is an NP-complete problem. Moreover, "mixed" matching problems can be easily represented as LGS queries; in fact, these problems can be obtained compositionally from simpler ones by means of the query algebra, as we will show in Sect. 7 where we solve a simplified version of the problem in [2]. Lastly (Sect. 8), we study a polynomial-time fragment of LGS that can still be used to compute various notions of graph pattern matching. Final conclusions and directions for further work (such as a distributed algorithm for computing LGSs) are in Sect. 9.

2 Hosts, Guests and Loose Graph Simulations

Loose graph simulations are a generalization of pattern matching for certain labelled graphs. As often proposed in the literature, the structures that need to be checked for properties are called *hosts*, whereas the structures that represent said properties are called *guests*.

Definition 1. *A* host graph *(herein also simply called* graph*) is a triple* (Σ, V, E) *consisting of a finite set of symbols* Σ *(also called* alphabet*), a finite set* V *of nodes and a set* $E \subseteq V \times \Sigma \times V$ *of edges. For an edge* $e = (v, l, v')$ *write* $s(e)$*,* $\sigma(e)$*, and* $t(e)$ *for its* source node v*,* label l*, and* target node v'*, respectively. For a vertex* v *write* $\mathrm{in}(v)$ *and* $\mathrm{out}(v)$ *for the sets* $\{e \mid t(e) = v\}$ *and* $\{e \mid s(e) = v\}$ *of its incoming and outgoing edges.*

Definition 2. *A* guest $G = (\Sigma, V, E, \mathcal{M}, \mathcal{U}, \mathcal{E}, \mathcal{C})$ *is a (host) graph* (Σ, V, E) *additionally equipped with:*

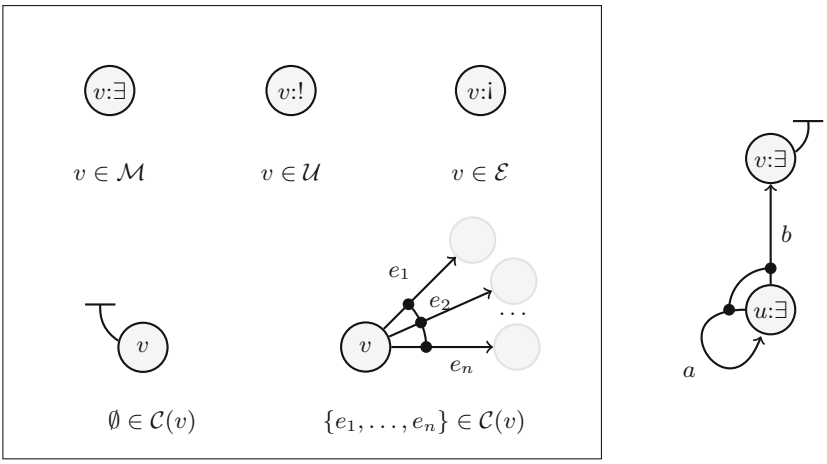

Fig. 1. The guest graphic notation (left) and an example (right).

– *three sets* $\mathcal{M},\mathcal{U},\mathcal{E} \subseteq V$*, called respectively* must*,* unique *and* exclusive *set.*
– *a* choice *function* $\mathcal{C} : V \to \mathcal{P}(\mathcal{P}(E))$*, s.t.* $\bigcup \mathcal{C}(v) = \text{out}(v)$ *for each* $v \in V$*.*

Roughly speaking, a guest is graph whose:

– nodes are decorated with usage constraints telling whether they must appear in the host, if their occurrence should be unique, and whether their occurrences can also be occurrences of other nodes or are exclusive;
– edges are grouped into possible "choices of sets of ongoing edges" for any given source node to be considered by a simulation.

The semantics of the three sets $\mathcal{M}$, $\mathcal{U}$, $\mathcal{E}$ and the choice function $\mathcal{C}$ will be presented formally in the definition of loose graph simulations (Definition 5).

Guests can be conveniently represented using the graphical notation shown in Fig. 1 (a formal algebra is discussed in Sect. 3). A node belonging to the must, unique or exclusive set is decorated with the symbols ∃, ! and ¡, respectively. Choice sets are represented by arcs with dots placed on the intersection with each edge that belongs to the given choice set. The empty choice set ($\emptyset \in \mathcal{C}(v)$) is represented by the "corked edge" (⍀).

Example 1. Figure 1 shows the graphical representation of a guest with two nodes u and v. The must set is $\{u, v\}$, the unique and exclusive sets are both empty, and the choice function takes u to $\{\{(u, a, u), (u, b, v)\}\}$ and v to $\{\emptyset\}$.

Before we formalise the notion of loose graph simulation, we need some auxiliary definitions. The following one fix the notation for paths in a graph.

Definition 3. *For* $M = (\Sigma, V, E)$*, define* $\mathbb{P}_M$ *as the set of all paths in* M*, i.e.* $\bigcup_{n\in\mathbb{N}}\{(e_0, \ldots, e_n) \in E^n \mid \forall i \in \{1, \ldots, n\}\ s(e_i) = t(e_{i-1})\}$*. Source* ($s\colon \mathbb{P}_M \to V$)*,*

target ($t\colon \mathbb{P}_M \to V$)*, and label* ($\sigma\colon \mathbb{P}_M \to \Sigma^+$) *functions are extended accordingly:* $s((e_0,\dots,e_n)) \triangleq s(e_0)$, $t((e_0,\dots,e_n)) \triangleq t(e_n)$, *and* $\sigma((e_0,\dots,e_n)) \triangleq \sigma(e_0)\dots\sigma(e_n)$. *Lastly, for any* $v, v' \in V$, *define* $\mathbb{P}_M(v,v')$ *as the set of all paths from* v *to* v', *formally* $\mathbb{P}_M(v,v') \triangleq \{\rho \in \mathbb{P}_M \mid s(\rho) = v \wedge t(\rho) = v'\}$.

Akin to graph simulations (Definition 11), LGSs are subgraphs of the product of guest and host that are coherent with the additional data prescribing node and edge usage.

Definition 4. *Let* $M_1 = (\Sigma_1, V_1, E_1)$ *and* $M_2 = (\Sigma_2, V_2, E_2)$ *be two graphs. The* tensor product graph $M_1 \times M_2$ *is the graph* $(\Sigma_1 \cap \Sigma_2,\ V_1 \times V_2,\ E^\times)$ *where* $E^\times \triangleq \{((u,u'),a,(v,v')) \mid (u,a,v) \in E_1 \wedge (u',a,v') \in E_2\}$.

When clear from the context, we denote host graphs and their components as H and as (Σ_H, V_H, E_H) (and variations thereof). We adopt the convention of denoting guests as G (and variations thereof) and writing $(\Sigma_G, V_G, E_G, \mathcal{M}, \mathcal{U}, \mathcal{E}, \mathcal{C})$ for the components of the guest G. We are now ready to define the notion of loose graph simulation.

Definition 5. *A* loose graph simulation *(LGS for short) of* G *in* H *is a subgraph* $(\Sigma_G \cap \Sigma_H, V^{G\to H}, E^{G\to H})$ *of* $G \times H$ *subject to the following conditions:*

(LGS1) *vertices of* G *in the must set occur in* $V^{G\to H}$, *i.e. for each* $u \in \mathcal{M}$ *there exists* $u' \in V_H$ *such that* $(u,u') \in V^{G\to H}$;

(LGS2) *vertices in the unique set are assigned to at most one vertex of* H, *i.e. for each* $u \in \mathcal{U}$ *and all* $u', v' \in V_H$, *if* $(u,u') \in V^{G\to H}$ *and* $(u,v') \in V^{G\to H}$ *then* $u' = v'$;

(LGS3) *vertices of* H *assigned to a vertex in the exclusive set cannot be assigned to other vertices, i.e. for each* $u \in \mathcal{E}$, $v \in V_G$ *and* $u' \in V_H$, *if* $(u,u') \in V^{G\to H}$ *and* $(v,u') \in V^{G\to H}$ *then* $u = v$;

(LGS4) *for* $(u,u') \in V^{G\to H}$, *there is a set in* $\mathcal{C}(u)$ *s.t. each of its elements is related to an edge with source* u' *and only such edges occur in* $E^{G\to H}$. *Formally,*

- *for each* $(u,u') \in V^{G\to H}$ *there exists* $\gamma \in \mathcal{C}(u)$ *such that for all* $(u,a,v) \in \gamma$ *it holds that* $((u,u'),a,(v,v')) \in E^{G\to H}$ *for some* $v' \in V_H$;
- *for each* $((u,u'),a,(v,v')) \in E^{G\to H}$ *there exists* $\gamma \in \mathcal{C}(u)$ *s.t.* $(u,a,v) \in \gamma$ *and for each* $(u,b,w) \in \gamma$ *it holds that* $((u,u'),b,(w,w')) \in E^{G\to H}$ *for some* $w' \in V_H$.

(LGS5) *the simulation preserves the connectivity w.r.t. nodes marked as must: for each* $(u,u') \in V^{G\to H}$ *and* $v \in \mathcal{M}$ *if* $\mathbb{P}_G(u,v) \neq \emptyset$ *then there exists* $v' \in V_H$ *such that* $\mathbb{P}_{(\Sigma_G\cap\Sigma_H, V^{G\to H}, E^{G\to H})}((u,u'),(v,v')) \neq \emptyset$.

The domain of all LGSs for G *and* H *is denoted as* $\mathbb{S}^{G\to H}$.

As already mentioned at the end of Definition 2, the definition of LGS attributes a semantics for the must, unique, exclusive sets and the choice function. Regarding the *unique set*, Condition LGS2 requires that every vertex of the guest in this set to be mapped by at most one element of the host. Similarly,

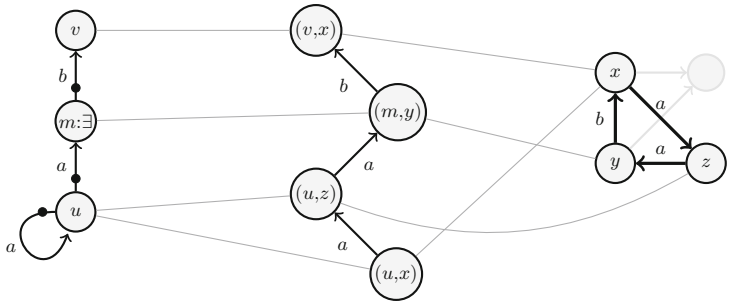

Fig. 2. An LGS (center) between a guest (left) and a host (right).

Condition LGS3 requires the vertices of the host paired in the LGS with a node of the *exclusive set* to be only paired with that node. Condition LGS4 defines the semantics of the choice function: given a pair of vertices $(u, u') \in V^{G \to H}$, it requires to select at least one set from $\mathcal{C}(u)$. The edges of these selected sets (and only these edges, as stated by the second part of the condition) must be paired in the LGS to edges in H with source u'. This condition can be seen as a generalization of the second condition of *graph simulations* (Definition 11) that requires all outgoing edges from u to be in relation with outgoing edges of u'.

Condition LGS1 and LGS5 formalise the constraints attached to must nodes: the first condition imposes that every vertex in this set must appear in the LGS, while the second condition requires that, for each $(u, u') \in V^{G \to H}$, each vertex in the must set reachable in the guest from u is also reachable in the LGS, with a path starting from (u, u').

Example 2. Figure 2 shows a guest and its loose graph simulation over a host. In this example $\mathcal{M} = \{m\}$ and $\mathcal{U} = \mathcal{E} = \emptyset$. Moreover, the choice function is *linear*, i.e. for each vertex u, $\mathcal{C}(u)$ contains a set $\{e\}$ for each edge in $\text{out}(u)$ and $\emptyset$ whenever $\text{out}(u) = \emptyset$, formally $\mathcal{C} = \lambda x.\{\{e\} \mid e \in \text{out}(x)\} \cup \{\emptyset \mid \text{out}(x) = \emptyset\}$. LGSs of this guest represents paths $(e_0, e_1, \ldots, e_n)$ of arbitrary length in the host such that $\forall i < n\ \sigma(e_i) = a$ and $\sigma(e_n) = b$. The guest is therefore similar to the regular language $a^\star\, b$ and a LGS identifies paths in the host labelled with words in this language.

Proposition 1. *Let G be a guest with choice function $\mathcal{C}$ defined as $\lambda x.\{\text{out}(x)\}$, let H be a host and let $S = (\Sigma_G \cap \Sigma_H, V^{G \to H}, E^{G \to H})$ be a subgraph of $G \times H$. If S satisfies Condition LGS4 then it also satisfies Condition LGS5.*

Proof. Let $\mathcal{C}(v) = \{\text{out}(v)\}$ for all $v \in V_G$. If $(u, u') \in V^{G \to H}$ then Condition LGS4 requires that for all $(u, a, v) \in \text{out}(u)$ there exists v' such that $(v, v') \in V^{G \to H}$ and $((u, u'), a, (v, v')) \in E^{G \to H}$. Coinductively, since the same will hold for every of those pair (v, v'), it follows that whenever there is a path in G from u to a node $m \in \mathcal{M}$ in the must set, then there must be a path in S from (u, u') to a pair of vertices (m, w), where $w \in V_H$. Hence, Condition LGS5 holds. □

3 An Algebra for Guests

Guests are used to specify the patterns to look for inside a host; hence they should be easy to construct and to understand. To this end, besides the graphical notation described in Sect. 2, in this section we introduce an algebra for guests which allows us to construct them in a compositional way.

Definition 6. *A guest is* empty *whenever it has no vertexes. A guest with only one vertex and no edges is a* unary guest *and is denoted a*

$$p_{\mathcal{A}} \triangleq (\emptyset, \{p\}, \emptyset, \{p \mid \exists \in \mathcal{A}\}, \{p \mid ! \in \mathcal{A}\}, \{p \mid \mathsf{i} \in \mathcal{A}\}, \{p \to \{\emptyset \mid \varnothing \in \mathcal{A}\}\})$$

where p is the only vertex and $\mathcal{A} \subseteq \{\exists, !, \mathsf{i}, \varnothing\}$ state if p is respectively in $\mathcal{M}$, $\mathcal{U}$, $\mathcal{E}$ or if $\emptyset \in \mathcal{C}(p)$. For α a name, P and Q unary guests, the arrow operator *from P to Q α is defined as*

$$P \xrightarrow{\alpha} Q \triangleq (\{\alpha\}, \{p, q\}, \{(p, \alpha, q)\}, \mathcal{M}_P \cup \mathcal{M}_Q, \mathcal{U}_P \cup \mathcal{U}_Q, \mathcal{E}_P \cup \mathcal{E}_Q, \mathcal{C}^{\to})$$

$$\mathcal{C}^{\to} \triangleq \lambda x. \begin{cases} c_P \cup \{\{(p, \alpha, q)\}\} \cup c_Q & \text{if } p = q \wedge x = p \\ c_P \cup \{\{(p, \alpha, q)\}\} & \text{if } p \neq q \wedge x = p \\ c_Q & \text{if } p \neq q \wedge x = q \end{cases}$$

A guest is called elementary *whenever it is empty, unary, or the result of the arrow operator.*

For example, a node p with only a self loop labelled α can be expressed with the term $p \xrightarrow{\alpha} p$. Besides the elementary guests, the algebra is completed by introducing two binary operators used to combine guests.

Definition 7. *Let G_1 and G_2 be two guests. Their* addition *is the guest:*

$$G_1 \oplus G_2 \triangleq (\Sigma_1 \cup \Sigma_2, V_1 \cup V_2, E_1 \cup E_2, \mathcal{M}_1 \cup \mathcal{M}_2, \mathcal{U}_1 \cup \mathcal{U}_2, \mathcal{E}_1 \cup \mathcal{E}_2, \mathcal{C}^{\oplus})$$

where the choice function $\mathcal{C}^{\oplus}$ is defined as

$$\mathcal{C}^{\oplus} \triangleq \lambda x. \begin{cases} \mathcal{C}_1(x) \cup \mathcal{C}_2(x) & \text{if } x \in V_1 \wedge x \in V_2 \\ \mathcal{C}_1(x) & \text{if } x \in V_1 \\ \mathcal{C}_2(x) & \text{if } x \in V_2 \end{cases}$$

The multiplication *of G_1 and G_2 is the guest:*

$$G_1 \otimes G_2 \triangleq (\Sigma_1 \cup \Sigma_2, V_1 \cup V_2, E_1 \cup E_2, \mathcal{M}_1 \cup \mathcal{M}_2, \mathcal{U}_1 \cup \mathcal{U}_2, \mathcal{E}_1 \cup \mathcal{E}_2, \mathcal{C}^{\otimes})$$

where the choice function $\mathcal{C}^{\otimes}$ is defined as follows

$$\mathcal{C}^{\otimes} \triangleq \lambda x. \begin{cases} \{\gamma_1 \cup \gamma_2 \mid \gamma_1 \in \mathcal{C}_1(x) \wedge \gamma_2 \in \mathcal{C}_2(x)\} & \text{if } x \in V_1 \wedge x \in V_2 \\ \mathcal{C}_1(x) & \text{if } x \in V_1 \\ \mathcal{C}_2(x) & \text{if } x \in V_2 \end{cases}$$

Notice how addition and multiplication operators differ only by the definition of the choice function for vertices of both G_1 and G_2. In the case of addition, the resulting choice function is the union of the two choice function $\mathcal{C}_1$ and $\mathcal{C}_2$, whereas for the multiplication, given a vertex $v \in V_1 \cap V_2$, every set of $\mathcal{C}^{\otimes}(v)$ is the union of a set in $\mathcal{C}_1(v)$ and one in $\mathcal{C}_2(v)$.

Proposition 2. *The operations $\oplus$ and $\otimes$ form an idempotent commutative semiring structure over the set of all guests.*

The algebra offers a clean and modular representation of guests. Modularity, in particular, allows us to combine queries as illustrated in the second part of this work. Furthermore, guests admit normal forms.

Definition 8. *A term G in the algebra of guests is in* normal form *whenever $G = \bigoplus_{i \in I} \bigotimes_{j \in J_i} G_{i,j}$ where each $G_{i,j}$ is an elementary guest.*

Example 3. Consider the guest $(\{a,b\}, \{p,q\}, \{(p,a,p),(p,b,q)\}, \{p,q\}, \emptyset, \emptyset, \{p \mapsto \{\{(p,a,p),(p,b,q)\}\}, q \mapsto \{\emptyset\}\})$ shown in Fig. 1 on the right. This guest is represented by the term $q_{\{\exists,\varnothing\}} \oplus (p_{\{\exists\}} \xrightarrow{a} p \otimes p \xrightarrow{b} q)$ which is in normal form.

Every guest admits a normal form.

Proposition 3. *For $G = (\Sigma, V, E, \mathcal{M}, \mathcal{U}, \mathcal{E}, \mathcal{C})$ a guest, its normal form is:*

$$\bigoplus_{v \in V} v_{\{\exists \mid v \in \mathcal{M}\} \cup \{! \mid v \in \mathcal{U}\} \cup \{\mathrm{i} \mid v \in \mathcal{E}\} \cup \{\varnothing \mid \emptyset \in \mathcal{C}(v)\}} \oplus \bigoplus_{\substack{v \in V \\ \gamma \in \mathcal{C}(v)}} \left(\bigotimes_{e \in \gamma} \left(s(e) \xrightarrow{\sigma(e)} t(e) \right) \right)$$

For $G = (\Sigma, V, E, \mathcal{M}, \mathcal{U}, \mathcal{E}, \mathcal{C})$ a guest, we write $G[p/q]$ for the guest obtained renaming $p \in V$ as $q \notin V$. In particular, the set of edges and choice function are:

$$E[p/q] = \left\{ (u,a,v) \;\middle|\; \begin{array}{l} (u',a,v') \in E \\ (u' \neq p \implies u = u') \wedge (u' = p \implies u = q) \\ (v' \neq p \implies v = v') \wedge (v' = p \implies v = q) \end{array} \right\}$$

$$\mathcal{C}[p/q] = \lambda x. \begin{cases} \{S[p/q] \mid S \in \mathcal{C}(x)\} & \text{if } x \neq p \wedge x \neq q \\ \{S[p/q] \mid S \in \mathcal{C}(p)\} & \text{if } x = q \end{cases}$$

4 The LGS Problem is NP-complete

In this section we analyse the complexity of computing LGSs by studying their emptiness problem. Without loss of generality, we will now consider only guests and hosts with the same Σ. In the following, let $G = (\Sigma_G, V_G, E_G, \mathcal{M}, \mathcal{U}, \mathcal{E}, \mathcal{C})$ and $H = (\Sigma_H, V_H, E_H)$ be a guest and a host respectively.

Definition 9. *The* emptiness problem for LGSs *between G and H consists in checking whether $\mathbb{S}^{G \to H} = \emptyset$.*

Proposition 4. *Computing LGSs, as well as their emptiness problem, is in NP.*

Proof. Let $S = (\Sigma, V^{G\to H}, E^{G\to H})$ be a subgraph of $G \times H$. We will now prove that there exists a polynomial algorithm w.r.t. the size of G and H that checks whether S satisfies all the conditions of Definition 5. The satisfiability checking of Condition LGS1 is in $\mathcal{O}(\mathcal{M} \times V^{G\to H})$ since it is sufficient for every vertex in the must set $\mathcal{M}$ to check whether there is a vertex of the host paired with it. For similar reasons, Conditions LGS2 and LGS3 can also be checked in polynomial time. Moreover, to check Conditions LGS4 it is sufficient to check, for each $(u, v) \in V^{G\to H}$, whether there is $\gamma \in \mathcal{C}(v)$ s.t. $\gamma \subseteq \pi_1 \circ \mathrm{out}((u, v))$ and if for all $u' \in \pi_1 \circ \mathrm{out}((u, v))$ there exists $\gamma \in \mathcal{C}(v)$ s.t. $u' \in \gamma \subseteq \pi_1 \circ \mathrm{out}((u, v))$. This can be done by a naive algorithm in $\mathcal{O}(V_H \times E_G \times (V_G \times E_H + \mathcal{C} \times E_G^2))$. Lastly, checking whether S satisfies Condition LGS5 requires the evaluation of the reachability relation of G and S and therefore can be computed in $\mathcal{O}(V_G^3 \times V_H^3)$ using the Floyd-Warshall Algorithm [11]. Since every condition can be checked in polynomial time we can conclude that the LGS problem is in NP. □

4.1 NP-Hardness: Subgraph Isomorphisms via LGSs

We will now show the NP-hardness of the emptiness problem for LGSs by reducing the emptiness problem for subgraph isomorphism to it. The subgraph isomorphism problem requires to check whether a subgraph of a graph (host) and isomorphic to a second graph (query) exists. Application of this problem can be found in network analysis [15], bioinformatics and chemoinformatics [1,4].

Definition 10. *Let $H = (\Sigma, V_H, E_H)$ and $Q = (\Sigma, V_Q, E_Q)$ be two graphs called* host *and* query *respectively. There exists a subgraph of H isomorphic to Q whenever there exists a pair of injections $\phi : V_Q \hookrightarrow V_H$ and $\eta : E_Q \hookrightarrow E_H$ s.t. $\sigma(e) = \sigma \circ \eta(e)$, $\phi \circ s(e) = s \circ \eta(e)$, and $\phi \circ t(e) = t \circ \eta(e)$ for each $e \in E_Q$.*

The subgraph isomorphism problem, as well as the emptiness problem associated to it, is shown to be NP-complete by Cook [6]. Its complexity and its importance makes it one of the most studied problem and multiple algorithmic solutions where derived for it [4,7,27]. We will now show that the emptiness problem for subgraph isomorphism can be solved using LGSs.

Proposition 5. *Let $H = (\Sigma, V_H, E_H)$ and $Q = (\Sigma, V_Q, E_Q)$ be a host and a query for subgraph isomorphism respectively. Moreover, let*

$$G = \bigoplus_{v \in V_Q} v_{\{\exists!\mathrm{i}\} \cup \{\varnothing \mid \mathrm{out}(v) = \emptyset\}} \oplus \left(\bigotimes_{e \in E_Q} \left(s(e) \xrightarrow{\sigma(e)} t(e) \right) \right)$$

Then, there exists a subgraph of H isomorphic to Q iff there exists a LGS of G in H, i.e. $\mathbb{S}^{G\to H} \neq \emptyset$.

Proof. From the definition of G, its must, unique and exclusive sets, as well as its choice function, are $\mathcal{M} = \mathcal{U} = \mathcal{E} = V_Q$ and $\mathcal{C} = \lambda x.\{\text{out}(x)\}$ respectively. Suppose $\phi : V_Q \hookrightarrow V_H$ and $\eta : E_Q \hookrightarrow E_H$ be two injections as in Definition 10. Then the graph $S = (\Sigma, V^{G\to H}, E^{G\to H})$ where $V^{G\to H} \triangleq \{(u, u') \mid u' = \phi(u)\}$ and $E^{G\to H} \triangleq \{((u, u'), a, (v, v')) \mid (u', a, v') = \eta((u, a, v))\}$ form a LGS for G. Indeed, it satisfy Conditions LGS1 to LGS3, since ϕ is an injection. Moreover, since $\eta : E_Q \hookrightarrow E_H$ is also an injection and for each edge $e \in E_Q$ it holds that $\sigma(e) = \sigma \circ \eta(e)$, $\phi \circ s(e) = s \circ \eta(e)$ and $\phi \circ t(e) = t \circ \eta(e)$, S must be such that for each $(u, u') \in V^{G\to H}$ and for each $(u, a, v) \in \text{out}(u)$ there exists v' such that $(v, v') \in V^{G\to H}$ and $((u, u'), a, (v, v')) \in E^{G\to H}$. It follows that S is a subgraph of $G \times H$ and Condition LGS4 is satisfied, since $\mathcal{C}(u) = \{\text{out}(u)\}$. Moreover the satisfaction of Condition LGS5 follows from Proposition 1. S is therefore a LGS of G in H. Conversely, suppose that there is a LGS $S = (\Sigma, V^{G\to H}, E^{G\to H})$. Let ϕ s.t. $\phi(u) = u' \iff (u, u') \in V^{G\to H}$ and η s.t. $\eta((u, a, v)) = (u', a, v') \iff ((u, u'), a, (v, v')) \in E^{G\to H}$. Since $\mathcal{M} = \mathcal{U} = \mathcal{E} = V_Q$ and S is a LGS, it holds that ϕ is an injection defined on the domain V_Q. Moreover η is also an injection, since $\mathcal{C} = \lambda x.\{\text{out}(x)\}$ and S satisfies Condition LGS4, and together with the hypothesis that S is a subgraph of $G \times H$ it must also hold that for each edge $e \in E_Q$ $\sigma(e) = \sigma \circ \eta(e)$, $\phi \circ s(e) = s \circ \eta(e)$ and $\phi \circ t(e) = t \circ \eta(e)$. There exists therefore a subgraph of H isomorphic to Q. □

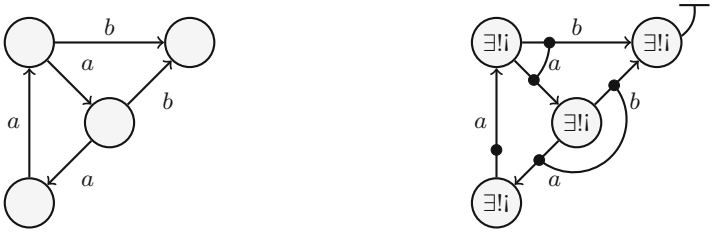

Fig. 3. A possible query for subgraph isomorphism (on the left) and its translation to a guest for LGSs (on the right).

Note how the translation from subgraph isomorphism's queries to guest for LGSs defined in Proposition 5 is *structure-preserving*. Indeed, an example of this can be seen in Fig. 3. This property is important since it makes defining LGSs' guests to solve the subgraph isomorphism problem as intuitive as the respective queries for it. This is also the case for other notions commonly used in the graphs' pattern matching community. Moreover, since the translated guest is as intuitive as the original query, this property strengthens the idea of using guests and LGSs to represent and compute hybrid queries w.r.t. these notions.

From Propositions 4 and 5 it follows that:

Theorem 1. *The emptiness problem for LGSs is NP-complete.*

5 Graph Simulations Are Loose Graph Simulations

Graph simulations are particular relations between graphs that are extensively applied in several fields [8,10]. The *graph simulation problem* requires to check whether a portion of a graph (host) *simulates* another graph (query).

Definition 11. *A* graph simulation *of* $Q = (\Sigma, V_Q, E_Q)$ *(herein* query*) in* $H = (\Sigma, V_H, E_H)$ *(herein* host*) is a relation* $\mathcal{R} \subseteq V_Q \times V_H$ *such that:*

- *for each node* $u \in V_Q$ *there exists a node* $v \in V_H$ *such that* $(u, v) \in \mathcal{R}$*;*
- *for each pair* $(u, v) \in \mathcal{R}$ *and for each edge* $e \in \mathrm{out}(u)$ *there exists an edge* $e' \in \mathrm{out}(v)$ *such that* $\sigma(e) = \sigma(e')$ *and* $(t(e), t(e')) \in \mathcal{R}$*.*

Graph simulation existence can be decided in polynomial time [3,13]. Their emptiness problem can be reduced to the emptiness problem for loose ones.

Proposition 6. *Let* $H = (\Sigma, V_H, E_H)$ *and* $Q = (\Sigma, V_Q, E_Q)$ *be a host and a query for graph simulation respectively. Moreover, let*

$$G = \bigoplus_{v \in V_Q} v_{\{\exists\} \cup \{\varnothing \mid \mathrm{out}(v) = \emptyset\}} \oplus \bigotimes_{e \in E_Q} s(e) \xrightarrow{\sigma(e)} t(e)$$

Then, there is a graph simulation of Q *in* H *iff* $\mathbb{S}^{G \to H} \neq \emptyset$*.*

Proof. From definition of G, its must, unique, exclusive sets and its choice function are $\mathcal{M} = V_Q$, $\mathcal{U} = \mathcal{E} = \emptyset$ and $\mathcal{C} = \lambda x.\{\mathrm{out}(x)\}$ respectively. Let $\mathcal{R}$ be a graph simulations. The graph $S = (\Sigma, V^{G \to H}, E^{G \to H})$ where $V^{G \to H} = \mathcal{R}$ and $E^{G \to H} = \{((u, u'), a, (v, v')) \mid (u, u'), (v, v') \in R, (u, a, v) \in E_Q, (u', a, v') \in E_H\}$ is a loose graph simulations for G. $\mathcal{U} = \mathcal{E} = \emptyset$ makes Conditions LGS2 and LGS3 always true, whereas the first condition of Definition 11, that requires all vertices of V_Q to appear in the first projection of $\mathcal{R}$, makes Conditions LGS1 satisfied. The second condition of Definition 11 requires that, given a pair $(u, v) \in R$, every edge of $\mathrm{out}(u)$ is associated with one edge of $\mathrm{out}(v)$ with the same label and with targets paired in $\mathcal{R}$. Condition LGS4 is therefore satisfied. Lastly, the satisfaction of Condition LGS5 follows from Proposition 1. S is therefore a loose graph simulation of G in H. Conversely, suppose there exists a LGS $S = (\Sigma, V^{G \to H}, E^{G \to H})$. Then $V^{G \to H}$ is a graph simulation. The definition of must set $\mathcal{M} = V_Q$ ensures that each vertex of V_Q must appear in the first projection of $V^{G \to H}$: the first condition of Definition 11 is satisfied. Moreover, the definition of the choice function $\mathcal{C} = \lambda x.\{\mathrm{out}(x)\}$ and Condition LGS4 implies that for each $(u, u') \in V^{G \to H}$ and for all $(u, a, v) \in \mathrm{out}(u)$ there exists v' such that $((u, u'), a, (v, v')) \in E^{G \to H}$ and, since S is a subgraph of $G \times H$, $(v, v') \in V^{G \to H}$. Thus, the second condition of Definition 11 holds and $V^{G \to H}$ is a graph simulation. □

Example 4. Figure 4 shows a query for graph simulations and the equivalent guest for loose graph simulations. As already seen in Sect. 4.1, the translation preserve the structure of the graph.

Fig. 4. A possible query for *graph simulation* (on the left) and its translation in a guest for loose graph simulations (on the right).

6 Regular Languages Pattern Matching

Regular languages defines finite sequences of characters (called *words* or *strings*) from a finite alphabet Σ [14]. Although widely used in text pattern matching, they are also used in graph pattern matching [2,20]. In this section we will restrict ourselves to ϵ-free regular languages, i.e. regular languages without the empty word ϵ [29]. This restriction is quite common, since the empty word is matched by any text or graph and therefore it does not represent a meaningful pattern.

Definition 12. *Let Σ be an alphabet. $\emptyset$ is a* ϵ-free regular language. *For each $a \in \Sigma$, $\{a\}$ is a* ϵ-free regular language. *If A and B are* ϵ-free regular language, *so are $A \cdot B \triangleq \{vw \mid v \in P \wedge w \in Q\}$, $A \mid B \triangleq A \cup B$, and $A^+ \triangleq \bigcup_{n \in \mathbb{N}} A^{n+1}$.*

In [29] it is shown that every regular language without the *empty* string ϵ can be expressed with the operations defined for *ϵ-free regular languages*. We will now introduce the pattern matching problem for non-empty ϵ-free regular languages. In the following let $H = (\Sigma, V_H, E_H)$ and $\mathcal{L}$ be respectively a host and a ϵ-free regular language such that $\mathcal{L} \neq \emptyset$.

Definition 13. *The emptiness problem for regular language pattern matching (RLPM) consist in checking if there is a path $\rho \in \mathbb{P}_H$ such that $\sigma(\rho) \in \mathcal{L}$.*

To solve this problem using LGSs we will use the equivalence between regular languages and non-deterministic finite automata [26].

Definition 14. *An NFA is a tuple, $N = (\Sigma, Q, \Delta, q_0, F)$ consisting of an alphabet Σ, a finite set of states Q, an* initial *state q_0, a set of* accepting *(or* final*) states $F \subseteq Q$ and a* transition *function $\Delta\colon Q \times \Sigma \to \mathcal{P}(Q)$. Let $w = a_0, a_1, \ldots, a_n$ be a word in Σ^*. The NFA N accepts w if there is a sequence of states $r_0, r_1, \ldots, r_{n+1}$ in Q such that $r_0 = q_0$, $r_{i+1} \in \Delta(r_i, a_i)$ for $i = 0, \ldots, n$, and $r_{n+1} \in F$. With $\mathcal{L}(N)$ we denote the set of words accepted by N, i.e. its accepted language.*

Remark 1. Any non-empty regular language without ϵ can be translated to a non-deterministic finite automaton (NFA) with one initial state (say q_0'), one final state (say f) and s.t. $\text{in}(q_0') = \emptyset$ and $\text{out}(f) = \emptyset$. Indeed, for $N = (\Sigma, Q, \Delta, q_0, F)$ any NFA s.t. $\mathcal{L}(N) \neq \emptyset$ and $\varepsilon \notin \mathcal{L}(N)$ define $N' = (\Sigma, Q \cup \{q_0', f\}, \Delta', q_0', \{f\})$ where:

– for all $a \in \Sigma$, $\Delta'(q'_0, a) \triangleq \Delta(q_0, a)$ and $\Delta'(f, a) = \emptyset$;
– for all $q \in Q$ and $a \in \Sigma$, $\Delta'(q, a) \triangleq \Delta(q, a) \cup \{f \mid F \cap \Delta(q, a) \neq \emptyset\}$.

By construction $\mathcal{L}(N) = \mathcal{L}(N')$, $\mathrm{in}(q'_0) = \emptyset$, and $\mathrm{out}(f) = \emptyset$.

Proposition 7. *Let $N = (Q, \Sigma, \Delta, q_0, \{f\})$ be a NFA where the initial state q_0 does not have any incoming transitions and the only final state f does not have any outgoing ones. Let $H = (\Sigma, V_H, E_H)$ be a host. Let*

$$G = q_{0\{\exists\}} \oplus f_{\{\exists,\varnothing\}} \oplus \bigoplus_{q \in Q,\ a \in \Sigma, q' \in \Delta(q,a)} \left(q \xrightarrow{a} q'\right)$$

Then, there exists a path $\rho \in \mathbb{P}_H$ in H s.t. $\sigma(\rho)$ is accepted by N iff there exists a loose graph simulation of G in H, i.e. $\mathbb{S}^{G \to H} \neq \emptyset$.

Proof. It follows from definition of acceptance that if there is $(e_0, \ldots, e_n) \in \mathbb{P}_H$ such that $\sigma(\rho)$ is accepted by N then, there is a sequence

$$(p_0, s(e_0)) \xrightarrow{\sigma(e_0)} (p_1, s(e_1)) \xrightarrow{\sigma(e_1)} \ldots \xrightarrow{\sigma(e_{n-1})} (p_n, s(e_n)) \xrightarrow{\sigma(e_n)} (p_{n+1}, t(e_n))$$

such that $p_0 = q_0$ and $p_{n+1} = f$; for all $i \in \{1, \ldots, n\}$ $t(e_{i-1}) = s(e_i)$; for all $i \in \{0, \ldots, n\}$ $p_{i+1} \in \Delta(p_i)$. Regard the sequence as a graph, say S, then $S \in \mathbb{S}^{G \to H}$ since S is a subgraph of $G \times H$ and G is constructed from N by preserving its transition relation Δ. Conditions LGS1 to LGS3 hold since $p_0 = q_0$, $p_n = f$ and $\mathcal{U} = \mathcal{E} = \emptyset$. Conditions LGS4 holds since $\{(p_i, \sigma(e_i), p_{i+1})\} \in \mathcal{C}(p_i)$ for any $i \in \{0, \ldots, n\}$ by construction. Conditions LGS5 holds since projecting the graph to its first component yields a path from q_0 to f. Representing G requires space polynomial in the size of N. Conversely, if there is $S \in \mathbb{S}^{G \to H}$ then LGS5 ensures that there is a path $\rho = (e_0, \ldots, e_n)$ in it such that $\pi_1 \circ s(\rho) = q_0$ and $\pi_1 \circ t(\rho) = f$. It follows from definition of E that the path ρ is coherent with Δ, i.e. $\forall i \in \{0, \ldots, n\}$ $\pi_1 \circ t(e_i) \in \Delta \circ \pi_1 \circ s(e_i)$. Thus, the sequence of labels $\sigma(\pi_2(\rho))$ in the second projection of ρ $((\pi_2 \circ s(e_0), \sigma(e_0), \pi_2 \circ t(e_0)), \ldots, (\pi_2 \circ s(e_n), \sigma(e_n), \pi_2 \circ t(e_n)))$, is such that $\sigma(\pi_2(\rho))$ is accepted by N. □

Example 5. Figure 5 shows a NFA and a guest identifying the same language. These two objects have the same structure (states/nodes and transition/edges).

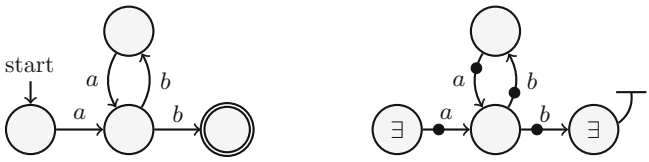

Fig. 5. A query for *regular languages* represented as an NFA (left) and as a LGS guest (on the right). The accepted language is $(ab)^+$.

7 Subgraph Isomorphism with Regular Path Expressions

Many approaches found in literature define hybrid notions of similarities, "merging" classical ones such as GS, SGI and RLPM [2,9]. These and similar merges are naturally handled by the modular definition of LGS guests. As an example, we discuss *subgraph isomorphism with regular languages* (RL-SGI) [2].

Definition 15. *Let Σ be a finite alphabet. A* graph decorated with regular languages (over Σ) *is a tuple $(\Sigma, V, E, \mathcal{L})$ consisting of a set V of nodes, a set $E \subseteq V \times V$ of edges and a labelling function $\mathcal{L} : E \to RE_\Sigma$ decorating each edge with a non empty ϵ-free regular language over Σ.*

Definition 16 (RL-SGI). *Let $H{=}(\Sigma, V_H, E_H)$ be a host and $Q{=}(\Sigma, V_Q, E_Q, \mathcal{L})$ a graph decorated with regular languages. We say that there is a* regular-language subgraph isomorphism of Q into H *iff there is a pair of injections $\phi : V_Q \hookrightarrow V_H$ and $\eta : E_Q \hookrightarrow \mathbb{P}_H$ s.t. for each $e \in E_Q$ $\phi \circ s(e) = s \circ \eta(e)$, $\phi \circ t(e) = t \circ \eta(e)$, and $\sigma \circ \eta(e) \in \mathcal{L}(e)$. Vertexes of paths in $\eta(E_Q)$ cannot appear in $\phi(V_Q)$ except for their source and target, i.e.: $\forall(e_0, \ldots, e_n) \in \eta(E_Q)\ \forall i \in \{1, \ldots, n\}\ s(e_i) \notin \phi(V_Q)$.*

RL-SGI can be seen as a hybrid notion between subgraph isomorphism and RLPM. We will now show how to solve this problem with loose graph simulations by defining a proper translation from its queries to guests.

Proposition 8. *Let $Q = (\Sigma, V_Q, E_Q, \mathcal{L})$ be a query for RL-SGI. Let*

$$G = \bigoplus_{v \in V_Q} v_{\{\exists!i\}} \oplus \bigotimes_{e \in E_Q} G_e[q_e/s(e)][f_e/t(e)]$$

such that G_e is the translation of the automaton $N_e = (\Sigma, V_e, \delta_e, q_e, \{f_e\})$ for $\mathcal{L}(e)$, as per Proposition 7 and where q_e and f_e are merged if $s(e) = t(e)$. For each host $H = (V_H, E_H)$ there exists a RL-SGI of Q into H iff $\mathbb{S}^{G \to H} \neq \emptyset$.

Proof. It follows from definition of G that: *(i)* V_Q is a subset of the vertices of V_G and $\mathcal{M} = \mathcal{U} = \mathcal{E} = V_Q$; *(ii)* for any $v \in V_Q$, any $\gamma \in \mathcal{C}(v)$, and any $e \in \text{out}(v)$ of Q, there is exactly one edge in γ that is induced by a transition in N_e Similarly to the proof of Proposition 5, Conditions LGS1 to LGS3 together with the first property ensure that each LGS over G corresponds to an injection w.r.t V_Q. It follows from the second property, Proposition 7, Conditions LGS4 and LGS5 that every LGS over G contains, for each $e \in E_Q$ a path whose labels, starting and ending nodes lie in $\mathcal{L}(e)$ and $V_Q \times V_H$, whereas all other vertices are in $(V_G \setminus V_Q) \times V_H$. Then, $\mathbb{S}^{G \to H} \neq \emptyset$ iff there are RL-SGIs of Q into H. □

Example 6. Figures 6 and 7 show a query for RL-SGI and its translation as a LGS guest. As illustrated by Proposition 8 and Fig. 7, translations are obtained modularly: following Sects. 4.1 and 6, the first step is to represent nodes and edges of a RL-SGI query in the guests for the SGI and RLPM queries, respectively; the second is to compose them via the guest algebra.

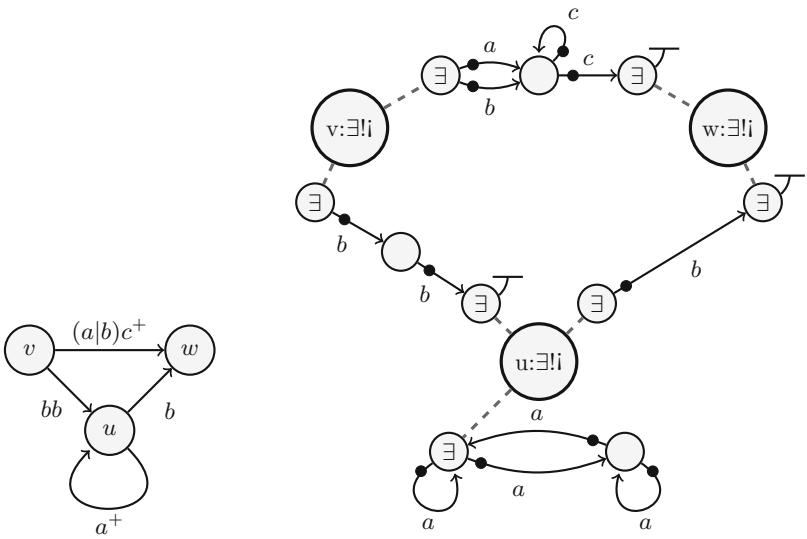

Fig. 6. A RE-SGISO query (left) and simple guests required to encode it (right). Vertices with the same name are highlighted by dashed edges between them.

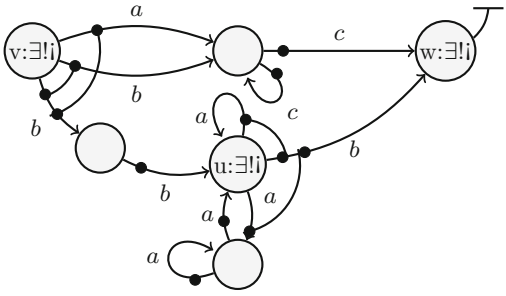

Fig. 7. A guest obtained via *multiplication* and *addition* operator from the guest in Fig. 6 (right) and equivalent to the RE-SGISO query in Fig. 6 (left).

8 A Polynomial Fragment of LGSs

RLPM and GS are two well-known problems for graph pattern matching and they both admit polynomial time algorithms. Since the emptiness problem for LGSs is NP-complete, we are interested in studying fragments of LGSs that are solvable in polynomial time yet expressive enough to capture the RLPM and GS problems. The class of simulation problems for guests whose unique and exclusive sets are empty enjoys this property.

Fix $G = (\Sigma_G, V_G, E_G, \mathcal{M}, \mathcal{U}, \mathcal{E}, \mathcal{C})$ and $H = (\Sigma_H, V_H, E_H)$. If $\mathcal{U}$ and $\mathcal{E}$ are empty then, LGSs for G and H are closes under unions hence the union $\bigcup \mathbb{S}^{G \to H}$ of all LGSs correspond to the greatest LGS. Observe that greatest LGSs may not exist in the general case.

Proposition 9. *Let G be a guest such that $\mathcal{U} = \mathcal{E} = \emptyset$. Then $\bigcup \mathbb{S}^{G \to H}$ is a LGS.*

Figure 8 shows an algorithm for computing the greatest LGS provided that $\mathcal{U}$ and $\mathcal{E}$ are empty. The algorithm runs in polynomial time and can be readily adapted to compute the greatest LGSs included in a given subgraph of $G \times H$. It follows that the emptiness problem admits a polynomial procedure.

Data: A host H and a guest G s.t. $\mathcal{U} = \mathcal{E} = \emptyset$
Result: $\bigcup \mathbb{S}^{G \to H}$ if it exists, otherwise *false*.

1 $(\Sigma, V_S, E_S) \leftarrow G \times H$;
2 **do**
3 $\quad (\Sigma, V_{S'}, E_{S'}) \leftarrow (\Sigma, V_S, E_S)$;
4 $\quad$ **foreach** $(u, v) \in V_{S'}$ **do**
5 $\quad\quad$ **foreach** $((u, v), a, (u', v')) \in \text{out}((u, v))$ **do**
6 $\quad\quad\quad$ **if** $\nexists \gamma \in \mathcal{C}(u)$ *s.t.* $(u, a, u') \in \gamma$ *and* $\forall (u, b, u'') \in \gamma$ $\exists (v, b, v'') \in \text{out}(v)$ $((u, v), b, (u'', v'')) \in \text{out}((u, v))$ **then**
7 $\quad\quad\quad\quad$ $E_{S'} \leftarrow E_{S'} \setminus \{((u, v), a, (u', v'))\}$;
8 $\quad\quad$ **if** $(\text{out}((u, v)) = \emptyset$ *and* $\emptyset \notin \mathcal{C}(u))$ *or* $(\exists m \in \mathcal{M}$ *s.t.* $\mathbb{P}_G(u, m) \neq \emptyset$ *and* $\forall v' \in V_H$ $\mathbb{P}_{(\Sigma, V_{S'}, E_{S'})}((u, v), (m, v')) = \emptyset)$ **then**
9 $\quad\quad\quad$ $E_{S'} \leftarrow E_{S'} \setminus (\text{out}((u, v)) \cup \text{in}((u, v)))$;
10 $\quad\quad\quad$ $V_{S'} \leftarrow V_{S'} \setminus \{(u, v)\}$;
11 **while** $V_S \neq V_{S'}$ *or* $E_S \neq E_{S'}$;
12 **if** $\forall m \in \mathcal{M}$ $\exists v \in V_H$ *s.t.* $(m, v) \in V_S$ **then return** (Σ, V_S, E_S) ;
13 **else return** false;

Fig. 8. Algorithm for computing the greatest loose graph simulation.

Theorem 2. *Let H be a host and G be a guest such that $\mathcal{U} = \mathcal{E} = \emptyset$. Then, the maximal LGS exists and is computed in polynomial time.*

Proof. The algorithm in Fig. 8 starts by computing $G \times H$ and saving it to (Σ, V_S, E_S) (Line 1). Afterwards, the *do-while* loop (Lines 2–11) proceeds removing nodes and edges of (Σ, V_S, E_S) that do not satisfy Conditions LGS4 and LGS5. Lastly (Lines 12–15), Condition LGS1 is checked and, if satisfied, (Σ, V_S, E_S) is returned, otherwise there is no greatest LGS and the algorithm terminates returning *false*. The algorithm runs in polynomial time, since Conditions LGS1, LGS4 and LGS5 can be checked in polynomial time (Proposition 4)

and the loop will be performed at most $|V_S| + |E_S|$ times. Conditions at Lines 6 and 8 check that edges and nodes satisfy Conditions LGS4 and LGS5. If any of these does not hold, the temporary copy of (Σ, V_S, E_S), i.e. $(\Sigma, V_{S'}, E_{S'})$, is updated removing an edge or a vertex. Thus, $V_S \neq V_{S'}$ or $E_S \neq E_{S'}$ iff (Σ, V_S, E_S) does not satisfy Conditions LGS4 and LGS5. After the *do-while* loop, (Σ, V_S, E_S) is a (possibly empty) relation that satisfies Conditions LGS4 and LGS5. Thus it remains only to check Condition LGS1 and this is done at Line 15: if the check fails there is no greatest LGSs otherwise it is the graph (Σ, V_S, E_S) returned by the algorithm. Assume otherwise that there is a LGS (Σ, V_M, E_M) s.t. $V_S \subset V_M$ or $E_S \subset E_M$. Then in (Σ, V_M, E_M) there is a node or an edge that satisfies LGS4 and LGS5 and is in $G \times H \setminus (\Sigma, V_S, E_S)$. Since it satisfies LGS4 and LGS5 it cannot be removed by the loop hence it is in (Σ, V_S, E_S) — a contradiction. □

9 Conclusions and Future Work

In this paper we have introduced *loose graph simulations*, which are relations between graphs that can be used to check structural properties of labelled hosts. LGSs' guests can be represented using a simple graphical notation, but also compositionally by means of an algebra which is sound and complete. We have shown formally that computing LGSs is an NP-complete problem, where the NP-hardness is obtained via a reduction of subgraph isomorphism to them. Moreover, we have shown that many other classical notions of graph pattern matching are naturally subsumed by LGSs. Therefore, LGSs offer a simple common ground between multiple well-known notions of graph pattern matching supporting a modular approach to these notions as well as to the development of common techniques.

An algorithm for computing LGSs in a decentralised fashion and inspired to the "distributed amalgamation" strategy is introduced in [16]. Roughly speaking, the host graph is distributed over processes; each process uses its partial view of the host to compute partial solutions to exchange with its peers. Distributed amalgamation guarantees each solution is eventually found by at least one process.

The same strategy is at the core of distributed algorithms for solving problems such as *bigraphical embeddings* and the distributed execution of bigraphical rewriting systems [17,19,22]. Bigraphs [12,21,23] have been proved to be quite effective for modelling, designing and prototyping distributed systems, such as *multi-agent systems* [18]. This similarity and the ability of LGS to subsume several graph problems suggests to investigate graph rewriting systems where redex occurrences are defined in terms of LGSs.

Another topic for further investigation is how to systematically minimise guests or combine sets of guests into single instances, while preserving the semantics of LGSs. Moreover, following what already done in Sect. 8, the complexity of various fragments of LGSs still needs to be addressed, *eg.* defining a fragment that is *fixed-parameter tractable*. Results in these directions would have a positive practical impact on applications based on LGSs.

Acknowledgements. We thank the anonymous reviewers and the participants to the GCM'17 workshop for their comments. We thank Andrea Corradini for his insightful observations on a preliminary version of this work and for proposing the name "loose graph simulations".

References

1. Apostolakis, J., Körner, R., Marialke, J.: Embedded subgraph isomorphism and its applications in cheminformatics and metabolomics. In: 1st German Conference in Chemoinformatics (2005)
2. Barceló, P., Libkin, L., Reutter, J.L.: Querying regular graph patterns. J. ACM **61**(1), 8:1–8:54 (2014)
3. Bloom, B., Paige, R.: Transformational design and implementation of a new efficient solution to the ready simulation problem. J. SCP **24**(3), 189–220 (1995)
4. Bonnici, V., Giugno, R., Pulvirenti, A., Shasha, D.E., Ferro, A.: A subgraph isomorphism algorithm and its application to biochemical data. BMC Bioinform. **14**(S-7), S13 (2013)
5. Chakrabarti, D., Faloutsos, C.: Graph mining: laws, generators, and algorithms. ACM Comput. Surv. **38**, 2 (2006)
6. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC, pp. 151–158. ACM (1971)
7. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. IEEE Trans. Pattern Anal. Mach. Intell. **26**(10), 1367–1372 (2004)
8. Fan, W.: Graph pattern matching revised for social network analysis. In: ICDT, pp. 8–21. ACM (2012)
9. Fan, W., Li, J., Ma, S., Tang, N., Wu, Y.: Adding regular expressions to graph reachability and pattern queries. Front. Comput. Sci. **6**(3), 313–338 (2012)
10. Fan, W., Wang, X., Wu, Y., Deng, D.: Distributed graph simulation: impossibility and possibility. PVLDB **7**(12), 1083–1094 (2014)
11. Floyd, R.W.: Algorithm 97: shortest path. Commun. ACM **5**(6), 345 (1962)
12. Grohmann, D., Miculan, M.: Directed bigraphs. In: Proceedings of MFPS. Electronic Notes in Theoretical Computer Science, vol. 173, pp. 121–137. Elsevier (2007)
13. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing simulations on finite and infinite graphs. In: FOCS, pp. 453–462. IEEE Computer Society (1995)
14. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation - International Edition, 2 edn. Addison-Wesley, Boston (2003)
15. Lischka, J., Karl, H.: A virtual network mapping algorithm based on subgraph isomorphism detection. In: VISA, pp. 81–88. ACM (2009)
16. Mansutti, A.: Le simulazioni lasche: definizione, applicazioni e computazione distribuita. Master's thesis, University of Udine (2016)
17. Mansutti, A., Miculan, M., Peressotti, M.: Distributed execution of bigraphical reactive systems. ECEASST **71**, 1–21 (2014)
18. Mansutti, A., Miculan, M., Peressotti, M.: Multi-agent systems design and prototyping with bigraphical reactive systems. In: Magoutis, K., Pietzuch, P. (eds.) DAIS 2014. LNCS, vol. 8460, pp. 201–208. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43352-2_16

19. Mansutti, A., Miculan, M., Peressotti, M.: Towards distributed bigraphical reactive systems. In: Echahed, R., Habel, A., Mosbah, M. (eds.) Proceedings of GCM (2014)
20. Mendelzon, A.O., Wood, P.T.: Finding regular simple paths in graph databases. SIAM J. Comput. **24**(6), 1235–1258 (1995)
21. Miculan, M., Peressotti, M.: Bigraphs reloaded: a presheaf presentation. Technical report UDMI/01/2013, University of Udine (2013)
22. Miculan, M., Peressotti, M.: A CSP implementation of the bigraph embedding problem. CoRR, abs/1412.1042 (2014)
23. Milner, R.: The Space and Motion of Communicating Agents. Cambridge University Press, Cambridge (2009)
24. Pevzner, P.: Computational Molecular Biology - an Algorithmic Approach. MIT Press, Cambridge (2000)
25. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformations. Foundations, vol. 1. World Scientific, Singapore (1997)
26. Thompson, K.: Regular expression search algorithm. Commun. ACM **11**, 419–422 (1968)
27. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM **23**, 31–42 (1976)
28. Yan, X., Han, J.: gSpan: graph-based substructure pattern mining. In: ICDM, pp. 721–724. IEEE Computer Society (2002)
29. Ziadi, D.: Regular expression for a language without empty word. Theor. Comput. Sci. **163**(1&2), 309–315 (1996)