Graph Decomposition and Local Reasoning

D. Costa, J. Brotherston, D. Pym

University College London
d.costa,j.brotherston,d.pym @ ucl.ac.uk

Abstract. In modelling complex systems, at scales from code-level, through distributed systems, to organizational structure, it is typically necessary to introduce a model of location; that is, the places at which system resources reside, together with their interconnectivity. Typically, locations are modelled using (possibly directed) graphs. Handling very large systems typically requires (i) that models be compositional and (ii) that the composition of models supports local reasoning about their properties; that is, just as in Separation Logic, the properties of the components of a model can be established independently of one another. In this paper, we provide a theory of graph composition, based on a concept of a 'pregraph', that supports local reasoning about component graphs.

1 Introduction

In the world of complex systems modelling, it is commonplace to employ model constructions in which there are concepts of location and of the connections between locations. These concepts are used to describe the underlying architecture of the system. Locations may be logical or physical, but, either way, perhaps the most common structures used to describe locations and the connections between them are (possibly directed) *graphs*.

The world of program verification, itself a form of complex systems modelling, provides a valuable insight: the desirability of models being constructed compositionally. First, large complex models should consist of assemblies of smaller, possibly independently defined, components; second, it is desirable to be able to reason about the component models *locally*. It should be possible to understand which properties of given component models are dependent on the properties of other component models. Therefore, if the essential structure of component models is described using graphs, it follows that a theory of graph (de)composition that supports local reasoning is needed.

We propose a solution in which we move from graphs to *pregraphs*. Roughly speaking, a pregraph is a graph in which we permit edges to be 'dangling' in the sense that one vertex may be absent. Ordinary and dangling edges are drawn as follows (direction omitted):

Dangling edges admit a form of composition that supports local reasoning, as will be explained in more detail in the sequel. For a sneak peak on how this works, consider Figure 1, in which the graph G has a subgraph H.

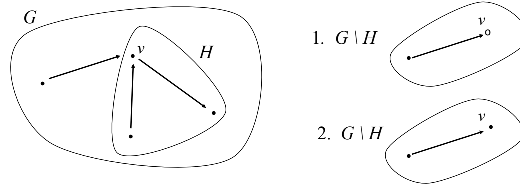

Fig. 1. Graph decomposition

The allocation of vertex v when we extract H from G is our main concern: should it be part of $G\backslash H$ and H simultaneously or only one of these? If dangling edges are permitted, then – for some form of pregraph decomposition, the choices for which will be discussed in the next section – the resulting $G\backslash H$ has no vertices in common with H – that is, it is separated from H in the sense of separation introduced in [4, 3, 5]. However, if dangling edges are not permitted, then the resulting $G\backslash H$ must share a vertex, v, with H. In the former case, an appropriate form of local reasoning is supported whereas in the latter it is not.

An appropriate form of local reasoning is described, in the spirit of [6], by the 'Frame Property' in Figure 2, where the action a may for example alter the structure of the pregraph. In short, the Frame Property holds when it is the case that if a acts locally on G_1, then the action a on $G_1 \circ G_2$ leaves G_2 unaffected.

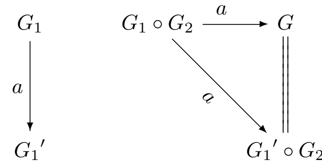

Fig. 2. Frame property.

While the use of graphs to structure complex system models is more-or-less ubiquitous, a theory of compositionality, essential to support tractable reasoning, remains outstanding. The key requirement to address this gap is to have a theory of (pre)graph (de)composition that supports effective local reasoning, which we propose in this paper.

The rest of this paper is structured as follows: In **Section 2**, we discuss the sensible choices of composition for pregraphs and identify the most appropriate choice. In **Section 3**, we define a separation-like logic which we called Separating Pregraph Logic, SPGL, for reasoning about pregraphs. We set up this logic in the style of Separation Logic, giving its semantic definition in the style of BI's Pointer Logic [4, 3]. There are several steps in this definition, which largely follow the corresponding steps in the set up of Separation Logic. We start by introducing a programming language for manipulating pregraphs. Then we introduce an assertion language to deal with properties of pregraphs. Putting these things

together, we sketch some key proof rules for SPGL's 'Hoare Triples' of the form $\{\varphi\}\mathtt{C}\{\psi\}$, where C is a command in the programming language for pregraphs, φ is precondition for C, and ψ is a postcondition for C. Perhaps the most important of these rules is the Frame Rule,

$$\frac{\{\varphi\}\mathtt{C}\{\psi\}}{\{\chi * \varphi\}\mathtt{C}\{\chi * \psi\}}$$

where χ does not include any free variables modified by C. This rule will allow us to reason about properties of pregraphs compositionally, a property of critical importance when handling large, complex models. In **Section 4**, we verify the correctness of the specification of a small toy program we built, before we move on to the verification of the well-known Breadth-first search algorithm. Finally, in **Section 5**, we summarize our contribution and consider some directions for further research.

2 Composition of pregraphs

In the introduction, we have argued for the value of considering the notion of *pregraphs*, which we now formally introduce. Critical to the value of considering pregraphs is their composition, so, having defined pregraphs, we proceed to reflect on how they might be composed, considering and comparing three possible definitions.

Definition 1 (Pregraph) *Let $\mathbb{V}$ be a countably infinite domain whose elements are called vertices and are usually denoted by lower case letters v, u, w, etc. A* pregraph, G, *is a pair* (V, E) *of finite sets where* $\mathrm{V} \subset \mathbb{V}$ *is the set of vertices of the pregraph and* $\mathrm{E} \subset (\mathrm{V} \times \mathbb{V}) \cup (\mathbb{V} \times \mathrm{V})$ *is the set of edges of the pregraph.*

A vertex $v \in \mathrm{V}$ is *owned by* G; a vertex $u \in \mathbb{V}\backslash\mathrm{V}$, such that either $(v, u) \in \mathrm{E}$ or $(u, v) \in \mathrm{E}$ for some owned vertex v, is called a *dangling vertex*. Edges in $\mathrm{V} \times \mathrm{V}$ are called *full edges*; otherwise they are called *half* or *dangling edges*. We may sometimes distinguish half edges (v, u) between incoming (if $u \in \mathrm{V}$) and outgoing (if $v \in \mathrm{V}$). Note that a graph (in the usual sense) is a pregraph, but a pregraph is not always a graph.

Example 1 *Consider the pregraph* $\mathsf{G} = (\{v, u, w\}, \{(v, u), (x, v), (x, w), (u, y)\})$. *A visual representation of* G *is as in Figure 3. The vertices v, u and w are owned by* G*;* (v, u) *is a full edge and all the others are half edges. In particular,* (x, v) *and* (x, w) *are incoming edges and* (u, y) *is an outgoing edge.*

We will now briefly explore some possible notions of composition for pregraphs. Before going any further, let us make it a rule that the composition of two pregraphs, when defined, is a pregraph that takes the union of vertices and the union of edges of the two initial pregraphs:

$$\mathsf{G}_1 \circ \mathsf{G}_2 \downarrow \text{ implies } \mathsf{G} = \mathsf{G}_1 \circ \mathsf{G}_2 = (\mathrm{V}_1 \cup \mathrm{V}_2, \mathrm{E}_1 \cup \mathrm{E}_2).$$

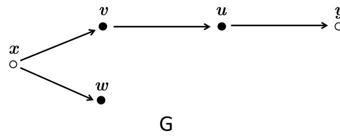

Fig. 3. A visual representation of the pregraph G from Example 1.

Our first requirement for the definedness of the composition of two pregraphs is aligned with the notion of separation in [6]: it is that the pregraphs must not share vertices. We call such pregraphs *independent*, formally defined as follows:

Definition 2 (Independent pregraphs) *The pregraphs* $\mathsf{G}_1 = (\mathrm{V}_1, \mathrm{E}_1)$ *and* $\mathsf{G}_2 = (\mathrm{V}_2, \mathrm{E}_2)$ *are said to be* independent*, denoted* $\mathsf{G}_1 \# \mathsf{G}_2$*, if* $\mathrm{V}_1 \cap \mathrm{V}_2 = \varnothing$*.*

Observe that if two pregraphs are independent, they do not share full edges. Therefore, the restrictions on the composition of pregraphs that we may further add must concern half edges. The simplest case we can consider is when there are no restrictions at all on half edges. Unfortunately, this composition, $\circ_1$ is not cancellative, as the following example illustrates:

Example 2 *Consider the pregraphs* $\mathsf{G}_1 = (\{v, w\}, \{(x, v), (x, w)\})$*,* $\mathsf{G}_2 = (\{u\}, \{(v, u), (u, y)\})$ *and* $\mathsf{G}_3 = (\{v, w\}, \{(x, v), (x, w), (v, u)\})$*. We can see that* $\mathsf{G}_1 \# \mathsf{G}_2$ *and* $\mathsf{G}_1 \# \mathsf{G}_3$ *and therefore, given there are no other restrictions, composing* G_1 *with* G_2 *and with* G_3 *is possible. The resulting pregraph in both cases is* $\mathsf{G} = (\{v, u, w\}, \{(v, u), (x, v), (x, w), (u, y)\})$*, so the composition is not cancellative.*

Thus, given pregraphs G and G_1 and knowing that $\mathsf{G} = \mathsf{G}_1 \circ_1 \mathsf{G}_2$, it is not always the case that there is a unique G_2 for which the equality hold. Hence, this is not the ideal notion of composition.

On the other hand, if in addition to consider independent pregraphs, we also demand that they do not share any edges, we will obtain a composition $\circ_2$ that is cancellative. Furthermore, it is commutative and associative too.

However, the decomposition of a pregraph $\mathsf{G} = (\mathrm{V}, \mathrm{E})$ into pregraphs G_1 and G_2, for fixed V_1 and V_2 is not always unique, as the following example shows:

Example 3 *Take the pregraph* $\mathsf{G} = (\{u, v\}, \{(u, v)\})$ *that we would like to decompose into two pregraphs* $\mathsf{G}_1 = (\{u\}, \mathrm{E}_1)$ *and* $\mathsf{G}_2 = (\{v\}, \mathrm{E}_2)$*. The sets of vertices* E_1 *and* E_2 *are not uniquely defined, as both the combinations* $\mathrm{E}_1 = \varnothing$*,* $\mathrm{E}_2 = \{(u, v)\}$ *and* $\mathrm{E}_1 = \{(u, v)\}, \mathrm{E}_2 = \varnothing$ *work.*

In the middle of the scale lies a third composition, $\circ_3$, that is defined when pregraphs are independent and share half edges in a special way. Namely, if a pregraph G_1 has an incoming (respectively outgoing) edge (u, v) and the dangling vertex u (respectively v) is owned by G_2, then the composition of G_1 and G_2 is defined only if (u, v) is an outgoing (respectively incoming) edge of G_2. The overall idea is that two independent pregraphs must agree on the half edges that have dangling vertices owned by one another. Formally,

$$\mathsf{G}_1 \circ_3 \mathsf{G}_2 \downarrow \text{ if and only if } \begin{array}{l} \mathsf{G}_1 \# \mathsf{G}_2 \text{ and} \\ \mathrm{E}_1 \cap (\mathrm{V}_2 \times \mathrm{V}_1) = \mathrm{E}_2 \cap (\mathrm{V}_2 \times \mathrm{V}_1) \text{ and} \\ \mathrm{E}_1 \cap (\mathrm{V}_1 \times \mathrm{V}_2) = \mathrm{E}_2 \cap (\mathrm{V}_1 \times \mathrm{V}_2). \end{array}$$

This composition ticks all the boxes for what we believe is a *good* notion of composition: $\circ_3$ is commutative, associative, cancellative, its unit is the empty pregraph $\mathsf{e} = (\varnothing, \varnothing)$, *i.e.* for every pregraph G, $\mathsf{G} \circ_3 \mathsf{e} = \mathsf{e} \circ_3 \mathsf{G} = \mathsf{G}$, and it provides unique decompositions of a pregraph when given a partition of its set of vertices. The following example shows the third composition at work:

Example 4 *Consider the pregraphs* $\mathsf{G}_1 = (\{u\}, \{(u, v)\})$ *and* $\mathsf{G}_2 = (\{v, w\}, \{(u, v), (v, w), (v, x)\})$. *First, observe that* $\mathsf{G}_1 \# \mathsf{G}_2$. *Then note that* $\mathrm{V}_2 \times \mathrm{V}_1 = \{(v, u), (w, u)\}$, *so* $\mathrm{E}_1 \cap (\mathrm{V}_2 \times \mathrm{V}_1) = \varnothing = \mathrm{E}_2 \cap (V_2 \times \mathrm{V}_1)$. *At last,* $\mathrm{V}_1 \times \mathrm{V}_2 = \{(u, v), (u, w)\}$ *implies that* $\mathrm{E}_1 \cap (\mathrm{V}_1 \times \mathrm{V}_2) = \{(u, v)\} = \mathrm{E}_2 \cap (V_1 \times \mathrm{V}_2)$. *Therefore,* $\mathsf{G}_1 \circ_3 \mathsf{G}_2 \downarrow$ *and so* $\mathsf{G}_1 \circ_3 \mathsf{G}_2 = (\{u, v, w\}, \{(u, v), (v, w), (v, x)\})$, *as represented in Figure 4.*

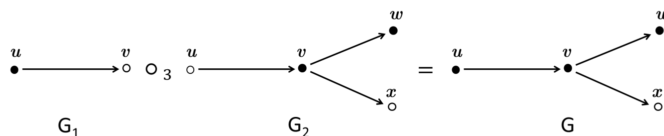

Fig. 4. A particular case of composition $\circ_3$ at work in Example 4.

As hinted before, given a pregraph $\mathsf{G} = (\mathrm{V}, \mathrm{E})$, its decomposition into pregraphs $\mathsf{G}_1 = (\mathrm{V}_1, \mathrm{E}_1)$ and $\mathsf{G}_2 = (\mathrm{V}_2, \mathrm{E}_2)$, for fixed $\mathrm{V}_1, \mathrm{V}_2$ such that $\mathrm{V}_1 \cup \mathrm{V}_2 = \mathrm{V}$ is now a precise computation. Each set of edges E_i $(i = 1, 2)$ is constructed by taking for every vertex v in V_i, all edges in E where v appears. Formally,

$$\mathrm{E}_i = \bigcup_{v \in \mathrm{V}_i} \{(u, v), (v, u) \in \mathrm{E}\}$$

The following example illustrates the decomposition process:

Example 5 *Take the pregraph* $\mathsf{G} = (\{u, v\}, \{(u, v)\})$ *as in Example 3. We would like to determine* G_1 *and* G_2 *such that* $\mathsf{G} = \mathsf{G}_1 \circ_3 \mathsf{G}_2$, *where* $\mathrm{V}_1 = \{u\}$ *and* $\mathrm{V}_2 = \{v\}$. *According to the method above,* $\mathrm{E}_1 = \mathrm{E}_2 = \{(u, v)\}$. *The decomposition is represented in Figure 5.*

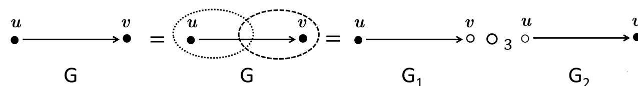

Fig. 5. Given a partition of the original set of vertices, $\circ_3$ produces unique decompositions.

Given its properties, from now on we will work with this last composition. Its formal definition comes as follows: (note that we drop the subscript)

Definition 3 (Composition of pregraphs) *The composition of pregraphs is a partial function* $\circ : \mathsf{PreGs} \times \mathsf{PreGs} \hookrightarrow \mathsf{PreGs}$. $\circ(\mathsf{G}_1, \mathsf{G}_2)$ *is defined if and only if* $\mathsf{G}_1 \# \mathsf{G}_2$, $\mathrm{E}_1 \cap (\mathrm{V}_2 \times \mathrm{V}_1) = \mathrm{E}_2 \cap (\mathrm{V}_2 \times \mathrm{V}_1)$ *and* $\mathrm{E}_1 \cap (\mathrm{V}_1 \times \mathrm{V}_2) = \mathrm{E}_2 \cap (\mathrm{V}_1 \times \mathrm{V}_2)$. *Whenever defined,* $\circ(\mathsf{G}_1, \mathsf{G}_2) = (\mathrm{V}_1 \cup \mathrm{V}_2, \mathrm{E}_1 \cup \mathrm{E}_2)$.

Notation-wise, we will for the rest of the paper use $\mathsf{G}_1 \circ \mathsf{G}_2$ rather than $\circ(\mathsf{G}_1, \mathsf{G}_2)$. Furthermore, the set of all pregraphs with composition and the empty pregraph constitutes a BBI-model, as shown next:

Proposition 1 *Let* PreGs *be the set of all pregraphs over vertex names* $\mathbb{V}$, *with the composition* $\circ$ *as defined in Definition 3. Then* $\langle \mathsf{PreGs}, \circ, \mathsf{e} \rangle$ *is a* BBI*-model.*

Additionally, it satisfies the following separation theory properties:

1. *Cancellativity:* $\mathsf{G} \circ \mathsf{H}_1 = \mathsf{G} \circ \mathsf{H}_2 \Rightarrow \mathsf{H}_1 = \mathsf{H}_2$;
2. *Indivisible Unit:* $\mathsf{G} \circ \mathsf{H} = \mathsf{e} \Rightarrow \mathsf{G} = \mathsf{H} = \mathsf{e}$;
3. *Disjointness:* $\mathsf{G} \circ \mathsf{G} = \mathsf{H} \Rightarrow \mathsf{G} = \mathsf{H} = \mathsf{e}$;
4. *Cross-split:* $\mathsf{G} \circ \mathsf{H} = \mathrm{I} \circ \mathrm{J} \Rightarrow \exists \mathsf{G}_1, \mathsf{G}_2, \mathsf{H}_1, \mathsf{H}_2.\ \mathsf{G} = \mathsf{G}_1 \circ \mathsf{G}_2, \mathsf{H} = \mathsf{H}_1 \circ \mathsf{H}_2$, $\mathrm{I} = \mathsf{G}_1 \circ \mathsf{H}_1$, $\mathrm{J} = \mathsf{G}_2 \circ \mathsf{H}_2$.

It is sometimes useful to associate data with vertices in a pregraph. A formal definition is as follows:

Definition 4 (Extended pregraph with data) *Given a pregraph* $\mathsf{G} = (\mathrm{V}, \mathrm{E})$ *in* PreGs, *an extension of* G *with data from a set* D *is a pregraph* $\mathsf{G}_D = (\mathrm{V}_D, \mathrm{E})$ *such that* $v \in \mathrm{V}$ *if and only if there is one tuple* $\langle v, d \rangle \in \mathrm{V}_D$ *for some data* $d \in D$, *also* $|\mathrm{V}| = |\mathrm{V}_D|$.

We will refer to extended pregraphs simply as pregraphs, omitting the mention to the data set D. Therefore, elements of sets of vertices V are tuples of the form $\langle v, d \rangle$. Whenever we want to refer solely to a vertex v we say that $v \in \overline{\mathrm{V}}$ which is taken to be the set underlying V.

Two extended pregraphs $\mathsf{G}_1, \mathsf{G}_2$ are independent if and only if $\overline{\mathrm{V}_1} \cap \overline{\mathrm{V}_2} = \varnothing$. The use of extended pregraphs rather than just pregraphs has no effect on their composition: simply replace in the formal definition V_1 and V_2 by $\overline{\mathrm{V}_1}$ and $\overline{\mathrm{V}_2}$.

3 SPGL: a separating logic over pregraphs

In this section, we introduce SPGL ('Separating PreGraph Logic') a close analogue of standard Separation Logic [3, 5] based on an underlying model of (extended) pregraphs and their associated composition (defined in the previous section), rather than heap memories and their composition (namely the union of disjoint heaps). First, we introduce a relatively abstract language of `while` programs that directly operate on pregraphs; then we introduce our logical language of *assertions* about program states in this language; and finally we introduce a Hoare logic proof system for this language, which in particular obeys separation logic's well known *frame rule* [6].

3.1 An abstract programming language on pregraphs

Here, we adapt a standard `while` programming language to operate directly on pregraphs. We assume countably infinite sets Var of *variables* and LVar of *list variables*. *List expressions* LExp, *branching conditions* B and *commands* C in our language are then given by the following grammar:

$$
\begin{aligned}
\mathsf{LExp} &::= [\,] \mid \mathsf{L} \mid x;\mathsf{LExp} \mid \mathsf{LExp};x \\
\mathsf{B} &::= x = y \mid x \neq y \mid \mathsf{LExp} = \mathsf{LExp} \mid \mathsf{LExp} \neq \mathsf{LExp} \mid x.\mathsf{data} = d \\
\mathsf{C} &::= \mathtt{skip} \mid \mathsf{C};\mathsf{C} \mid \mathtt{if}\,\mathsf{B}\,\mathtt{then}\,\mathsf{C}\,\mathtt{else}\,\mathsf{C} \mid \mathtt{while}\,\mathsf{B}\,\mathtt{do}\,\mathsf{C} && \text{(i)} \\
&\quad \mid x := \mathtt{add_vertex}() \mid \mathtt{add_edge}(x,y) \mid \mathtt{del_vertex}(x) \\
&\quad \mid \mathtt{del_edge}(x,y) && \text{(ii)} \\
&\quad \mid \mathtt{enqueue}(\mathsf{L},x) \mid x := \mathtt{dequeue}(\mathsf{L}) \mid \mathsf{L} := \mathsf{LExp} \\
&\quad \mid x.\mathsf{data} := d && \text{(iii)}
\end{aligned}
$$

where $\mathsf{L} \in \mathsf{LVar}$, x, y range over Var and d is data from a set D.

The first group of commands, (i), corresponds to the usual skip, composition, if-then-else, and while programs. The second group of commands, (ii), involves pregraph-changing programs at the level of the pregraph structure, either by adding or deleting vertices or edges. The third group of commands, (iii), includes the addition/removal of an element to/from a queue, the assignment of specific data to a variable, and the assignment of a list to a variable.

Our semantic model is based on the usual stack-and-heap model of standard separation logic (cf. [6]), except that we replace heaps by (extended) pregraphs. A *(program) state* is given by a pair (s, G) where G (in PreGs) is a pregraph with data and $s : \mathsf{Var} \cup \mathsf{LVar} \to (\mathbb{V} \times D) \cup \mathsf{ls}(\mathbb{V} \times D)$ is called a *stack* representing the local store, where $s(x) \in (\mathbb{V} \times D)$, $s(\mathsf{L}) \in \mathsf{ls}(\mathbb{V} \times D)$, and $\mathsf{ls}(\mathsf{X})$ denotes the set of lists with elements in X. Observe that we represent pairs in $\mathbb{V} \times D$ using angle brackets: $\langle u, d\rangle$. We extend s to list expressions by making $s([\,]) = [\,]$, $s(x;\mathsf{LExp}) = s(x);s(\mathsf{LExp})$ and $s(\mathsf{LExp};x) = s(\mathsf{LExp});s(x)$. It is sometimes useful to refer to the vertex assigned to a variable, for which we use the following notation: $\overline{s}(x) = u \Leftrightarrow s(x) = \langle u, d\rangle$. for some $d \in D$. We define a function to evaluate branching conditions, dependent on the stack, as follows $[\![.]\!]_s : \mathsf{B} \to \{\text{true}, \text{false}\}$ such that $[\![\mathsf{X} \bullet \mathsf{Y}]\!]_s = \text{true}$ if and only if $s(\mathsf{X}) \bullet s(\mathsf{Y})$, $\bullet \in \{=, \neq\}$ and $[\![x.data = d]\!]_s = \text{true}$ if and only if $s(x) = \langle u, d\rangle$, for some $u \in \mathbb{V}$. A *(program) configuration* is either a triple $\langle \mathsf{C}, s, \mathsf{G}\rangle$ where C is a command and (s, G) a program state, or the special configuration fault, used to denote memory faults.

The small-step operational semantics of programs is given by a binary relation $\rightsquigarrow$ on program configurations, shown in Tables 1, 2 and 3, where $\langle \mathsf{C}, s, \mathsf{G}\rangle \rightsquigarrow \langle \mathsf{C}', s', \mathsf{G}'\rangle$ holds if the execution of the command C in the state (s, G) can result in the configuration $\langle \mathsf{C}', s', \mathsf{G}'\rangle$. We write $\rightsquigarrow^*$ for the reflexive-transitive closure of $\rightsquigarrow$. A configuration $\langle \mathsf{C}, s, \mathsf{G}\rangle$ is called *safe* if $\langle \mathsf{C}, s, \mathsf{G}\rangle \not\rightsquigarrow^* \mathsf{fault}$, i.e. if no memory error can result from it.

$$\frac{}{\langle \mathtt{skip};\ \mathsf{C}, s, \mathsf{G}\rangle \rightsquigarrow \langle \mathsf{C}, s, \mathsf{G}\rangle} \qquad \frac{\langle \mathsf{C}_1, s, \mathsf{G}\rangle \rightsquigarrow \langle \mathsf{C}_1', s', \mathsf{G}'\rangle}{\langle \mathsf{C}_1; \mathsf{C}_2, s, \mathsf{G}\rangle \rightsquigarrow \langle \mathsf{C}_1'; \mathsf{C}_2, s', \mathsf{G}'\rangle}$$

$$\frac{[\![\mathsf{B}]\!]_s = \text{true}}{\langle \mathtt{if}\ \mathsf{B}\ \mathtt{then}\ \mathsf{C}\ \mathtt{else}\ \mathsf{C}', s, \mathsf{G}\rangle \rightsquigarrow \langle \mathsf{C}, s, \mathsf{G}\rangle} \qquad \frac{[\![\mathsf{B}]\!]_s = \text{false}}{\langle \mathtt{if}\ \mathsf{B}\ \mathtt{then}\ \mathsf{C}\ \mathtt{else}\ \mathsf{C}', s, \mathsf{G}\rangle \rightsquigarrow \langle \mathsf{C}', s, \mathsf{G}\rangle}$$

$$\frac{[\![\mathsf{B}]\!]_s = \text{true}}{\langle \mathtt{while}\ \mathsf{B}\ \mathtt{do}\ \mathsf{C}, s, \mathsf{G}\rangle \rightsquigarrow \langle \mathsf{C}; \mathtt{while}\ \mathsf{B}\ \mathtt{do}\ \mathsf{C}, s, \mathsf{G}\rangle} \qquad \frac{[\![\mathsf{B}]\!]_s = \text{false}}{\langle \mathtt{while}\ \mathsf{B}\ \mathtt{do}\ \mathsf{C}, s, \mathsf{G}\rangle \rightsquigarrow \langle \mathtt{skip}, s, \mathsf{G}\rangle}$$

Table 1. Small-step operational semantics of commands in group (i).

$$\frac{s' \overset{x}{\sim} s \qquad s'(x) = \langle u, d\rangle : u \notin \overline{\mathrm{V}} \quad \mathrm{V}' = \mathrm{V} \cup \{s'(x)\}}{\langle x := \mathtt{add_vertex}(d), s, (\mathrm{V}, \mathrm{E})\rangle \rightsquigarrow \langle \mathtt{skip}, s', (\mathrm{V}', \mathrm{E})\rangle}$$

$$\frac{s(x), s(y) \in \mathrm{V} \quad \mathrm{E}' = \mathrm{E} \cup \{(\overline{s}(x), \overline{s}(y))\}}{\langle \mathtt{add_edge}(x, y), s, (\mathrm{V}, \mathrm{E})\rangle \rightsquigarrow \langle \mathtt{skip}, s, (\mathrm{V}, \mathrm{E}')\rangle} \qquad \frac{s(x) \notin \mathrm{V} \text{ or } s(y) \notin \mathrm{V}}{\langle \mathtt{add_edge}(x, y), s, (\mathrm{V}, \mathrm{E})\rangle \rightsquigarrow \mathsf{fault}}$$

$$\frac{s(x) \in \mathrm{V} \quad \mathrm{V}' = \mathrm{V} \backslash \{s(x)\} \quad \mathrm{E}' = \mathrm{E} \cap ((\overline{\mathrm{V}'} \times \mathbb{V}) \cup (\mathbb{V} \times \overline{\mathrm{V}'}))}{\langle \mathtt{del_vertex}(x), s, (\mathrm{V}, \mathrm{E})\rangle \rightsquigarrow \langle \mathtt{skip}, s, (\mathrm{V}', \mathrm{E}')\rangle}$$

$$\frac{s(x) \notin \mathrm{V}}{\langle \mathtt{del_vertex}(x), s, (\mathrm{V}, \mathrm{E})\rangle \rightsquigarrow \mathsf{fault}}$$

$$\frac{s(x), s(y) \in \mathrm{V} \quad \mathrm{E}' = \mathrm{E} \backslash \{(\overline{s}(x), \overline{s}(y))\}}{\langle \mathtt{del_edge}(x, y), s, (\mathrm{V}, \mathrm{E})\rangle \rightsquigarrow \langle \mathtt{skip}, s, (\mathrm{V}, \mathrm{E}')\rangle} \qquad \frac{s(x) \notin \mathrm{V} \text{ or } s(y) \notin \mathrm{V}}{\langle \mathtt{del_edge}(x, y), s, (\mathrm{V}, \mathrm{E})\rangle \rightsquigarrow \mathsf{fault}}$$

Table 2. Small-step operational semantics of commands in group (ii).

$$\frac{s' \overset{\mathsf{L}}{\sim} s \quad s'(\mathsf{L}) = s(\mathsf{L}; x)}{\langle \mathsf{enqueue}(\mathsf{L}, x), s, \mathsf{G}\rangle \rightsquigarrow \langle \mathtt{skip}, s', \mathsf{G}\rangle}$$

$$\frac{s(\mathsf{L}) = s(y; \mathsf{LExp}) \quad s' \overset{x,\mathsf{L}}{\sim} s \quad s'(x) = s(y) \quad s'(\mathsf{L}) = s(\mathsf{LExp})}{\langle x := \mathtt{dequeue}(\mathsf{L}), s, \mathsf{G}\rangle \rightsquigarrow \langle \mathtt{skip}, s', \mathsf{G}\rangle}$$

$$\frac{s(\mathsf{L}) = []}{\langle x := \mathtt{dequeue}(\mathsf{L}), s, \mathsf{G}\rangle \rightsquigarrow \mathsf{fault}} \qquad \frac{s' \overset{\mathsf{L}}{\sim} s \quad s'(\mathsf{L}) = s(\mathsf{LExp})}{\langle \mathsf{L} := \mathsf{LExp}, s, \mathsf{G}\rangle \rightsquigarrow \langle \mathtt{skip}, s', \mathsf{G}\rangle}$$

$$\frac{s(x) \in \mathrm{V} \quad s' \overset{x}{\sim} s \quad (s(x) = \langle u, d'\rangle \text{ implies } s'(x) = \langle u, d\rangle) \quad \mathrm{V}' = (\mathrm{V} \backslash \{s(x)\}) \cup \{s'(x)\}}{\langle x.\mathsf{data} := d, s, (\mathrm{V}, \mathrm{E})\rangle \rightsquigarrow \langle \mathtt{skip}, s', (\mathrm{V}', \mathrm{E})\rangle}$$

$$\frac{s(x) \notin \mathrm{V}}{\langle x.\mathsf{data} := d, s, (\mathrm{V}, \mathrm{E})\rangle \rightsquigarrow \mathsf{fault}}$$

Table 3. Small-step operational semantics of commands in group (iii).

There are two key operational facts about our programming language, which collectively underpin the soundness of separation logic's *frame rule*. Those results are stated below, cf. [6].

Lemma 1 (Safety monotonicity) *If* $\langle \mathsf{C}, s, \mathsf{G}\rangle$ *is safe, and* $\mathsf{G} \circ \mathsf{G}'$ *is defined, then* $\langle \mathsf{C}, s, \mathsf{G} \circ \mathsf{G}'\rangle$ *is also safe.*

Lemma 2 (Frame property) *If* $\langle \mathsf{C}, s, \mathsf{G}_1 \circ \mathsf{G}_2\rangle \rightsquigarrow^* \langle s', \mathsf{G}'\rangle$ *and* $\langle \mathsf{C}, s, \mathsf{G}_1\rangle$ *is safe, then there exists* s'', G_1' *such that* $\langle \mathsf{C}, s, \mathsf{G}_1\rangle \rightsquigarrow^* \langle s'', \mathsf{G}_1'\rangle$, $s'' \equiv s'$ *and* $\mathsf{G}' = \mathsf{G}_1' \circ \mathsf{G}_2$.

3.2 Assertion language: syntax and semantics

Here, we describe a language of assertions suitable for describing properties of pregraphs and their composition. Following the general philosophy of separation logic, where the atomic formulas describe individual memory cells ($x \mapsto y$), here our atomic formulas will describe regions whose vertices share the same data.

We assume infinite, countable sets Var of *variables* and LVar of *list variables*. We also take in the construction of list expressions LExp from the previous subsection. The connectives of the logic are imported directly from the assertion language of separation logic, which in turn can be seen as an instance of the bunched logic BBI [3, 1].

Definition 5 (Formulas) Formulas *of* SPGL *are given by the following grammar:*

$$\begin{aligned}\varphi, \psi := {} & x = y \mid r = t \mid x.\mathsf{data} = d \mid \mathsf{region}(d, \mathsf{S}, \mathsf{T}) \\ & \mid \bot \mid \top \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \varphi \rightarrow \psi \mid \mathrm{I} \mid \varphi * \psi \mid \varphi -\!\!* \psi\end{aligned}$$

where $x, y \in \mathsf{Var}$*,* $r, t \in \mathsf{LExp}$*,* $\mathsf{S} \subseteq \mathsf{Var}$*,* $\mathsf{T} \subseteq \mathsf{Var}^2$*, and* $d \in D$*.*

In order to deal with the sets that appear in our formulas, we update the stack s such that for a set of variables S/pairs of variables T, the stack returns a set of vertices/pairs of vertices, respectively. Namely: $s(\mathsf{S}) = \{s(x) \mid x \in \mathsf{S}\}$ and $s(\mathsf{T}) = \{(\overline{s}(x), \overline{s}(y)) \mid (x, y) \in \mathsf{T}\}$. We define an extension to deal with a couple new set operations, as follows: $s(\mathsf{X} \Cup \mathsf{Y}) = s(\mathsf{X}) \cup s(\mathsf{Y})$, $s(\mathsf{X} \Cap \mathsf{Y}) = s(\mathsf{X}) \cap s(\mathsf{Y})$ and $s(\mathsf{X} \mathbin{\backslash\!\backslash} \mathsf{Y}) = s(\mathsf{X}) \backslash s(\mathsf{Y})$ where X, Y are both sets of variables or sets of pairs of variables. And finally, to deal with restricted sets of pairs: $s(\mathsf{T}\|_{\mathsf{S}}) = \{(\overline{s}(x), \overline{s}(y)) \mid s(x) \in s(\mathsf{S}) \text{ or } s(y) \in s(\mathsf{S})\}$. Observe that $s((\mathsf{X} \mathbin{\backslash\!\backslash} \mathsf{Y}) \Cup \mathsf{Y}) = s(\mathsf{X})$ and $s(\mathsf{T}\|_{\mathsf{S} \backslash\!\backslash \mathsf{S}'} \Cup \mathsf{T}\|_{\mathsf{S}'}) = s(\mathsf{T}\|_{\mathsf{S}})$.

Now we introduce the semantics. First, we write PreGs for the set of all pregraphs with vertices from $\mathbb{V}$ and data in D. We define a local satisfaction relation as follows:

Definition 6 (Satisfaction) *The satisfaction relation* $s, \mathsf{G} \models \varphi$ *where* s *is a stack,* G *is a pregraph and* φ *is a formula, is defined by structural induction on* φ *in Figure 6.*

$$\begin{array}{rcl}
s, \mathsf{G} \models x = y & \Leftrightarrow & s(x) = s(y) \\
s, \mathsf{G} \models r = t & \Leftrightarrow & s(r) = s(t) \\
s, \mathsf{G} \models x.\mathsf{data} = d & \Leftrightarrow & s(x) = \langle u, d \rangle \text{ for some } u \in \mathbb{V} \\
s, \mathsf{G} \models \mathsf{region}(d, \mathsf{S}, \mathsf{T}) & \Leftrightarrow & \text{for all } u,\ \text{if } u \in \overline{\mathrm{V}_{\mathsf{G}}} \text{ then } \langle u, d \rangle \in \mathrm{V}_{\mathsf{G}}, \\
 & & \mathrm{V}_{\mathsf{G}} = s(\mathsf{S}) \text{ and } \mathrm{E}_{\mathsf{G}} = s(\mathsf{T}) \\
s, \mathsf{G} \models \mathrm{I} & \Leftrightarrow & \mathsf{G} = \mathsf{e} \\
s, \mathsf{G} \models \varphi * \psi & \Leftrightarrow & \text{there exist } \mathsf{G}_1, \mathsf{G}_2 \text{ s.t. } \mathsf{G} = \mathsf{G}_1 \circ \mathsf{G}_2 \\
 & & \text{and } s, \mathsf{G}_1 \models \varphi \text{ and } s, \mathsf{G}_2 \models \psi \\
s, \mathsf{G} \models \varphi -\!\!* \psi & \Leftrightarrow & \text{for all } \mathsf{G}', \text{ if } \mathsf{G} \circ \mathsf{G}' \text{ is defined and } \mathsf{G}', s \models \varphi, \\
 & & \text{then } \mathsf{G} \circ \mathsf{G}', s \models \psi
\end{array}$$

Semantics for constants $\top, \bot$ and Boolean connectives is defined as usual.

Fig. 6. Definition of the satisfaction relation $s, \mathsf{G} \models \varphi$ for SPGL.

A formula φ is said to be valid, *denoted* $\models \varphi$, *if* $s, \mathsf{G} \models \varphi$ *for all* $\mathsf{G} \in \mathsf{PreGs}$. *We write the* entailment $\varphi \models \psi$, *where* φ *and* ψ *are formulas, to mean that if* $s, \mathsf{G} \models \varphi$ *then* $s, \mathsf{G} \models \psi$.

Note that every pregraph can be described by a formula of the form

$$\mathsf{region}(d_1, \{x_1\}, \mathsf{T}_1) * \ldots * \mathsf{region}(d_n, \{x_n\}, \mathsf{T}_n)$$

where n is its number of vertices.

3.3 Logical interactions in SPGL

In this section we will explore interactions between region formulas, namely how to split and combine regions into new ones. First, we show how we can split a region into two subregions. The result is stated for when the original region has size n (the number of vertices in the pregraph) and its subregions have size $n-1$ and 1. Other partitions can be obtained in conjunction with the result on merging regions that will appear after.

Lemma 3 (Split) $\mathsf{region}(d, \mathsf{S}, \mathsf{T}) \models \mathsf{region}(d, \mathsf{S}_1, \mathsf{T}_1) * \mathsf{region}(d, \mathsf{S}_2, \mathsf{T}_2)$, *where* $\mathsf{S}_1 = \mathsf{S} \mathbin{\backslash\!\!\backslash} \{x\}$, $\mathsf{S}_2 = \mathsf{S} \Cap \{x\}$, $\mathsf{T}_i = \mathsf{T}\|_{\mathsf{S}_i}$, $i = 1, 2$.

Reversely, given two regions with the same data, we can combine them into a unique region as follows:

Lemma 4 (Merge) $\mathsf{region}(d, \mathsf{S}_1, \mathsf{T}_1) * \mathsf{region}(d, \mathsf{S}_2, \mathsf{T}_2) \models \mathsf{region}(d, \mathsf{S}_1 \Cup \mathsf{S}_2, \mathsf{T}_1 \Cup \mathsf{T}_2)$.

3.4 Hoare logic proof system

It is time to introduce specifications for the programs considered in Subsection 3.1. We begin by properly defining partial correctness for Hoare triples $\{\varphi\}\mathsf{C}\{\psi\}$, which will be, in fact, analogous to the usual definition in the heaplets semantics.

Definition 7 (Partial correctness) $\{\varphi\}\mathsf{C}\{\psi\}$ *is true when, for all* s, G, *if* $s, \mathsf{G} \models \varphi$ *then* (i) $\mathsf{C}, s, \mathsf{G}$ *is safe and* (ii) *if* $\mathsf{C}, s, \mathsf{G} \rightsquigarrow^* s', \mathsf{G}'$ *then* $s', \mathsf{G}' \models \psi$.

Table 4 lists the specifications for all programs in groups (ii) and (iii) introduced in Subsection 3.1.

The Frame Rule for SPGL takes the form:

$$\frac{\{\varphi\}\mathsf{C}\{\psi\}}{\{\varphi * \chi\}\mathsf{C}\{\psi * \chi\}}, \mathsf{Mod}(\mathsf{C}) \cap [\mathsf{L}]\mathsf{Var}(\chi) = \varnothing$$

where $\mathsf{Mod}(\mathsf{C})$ denotes the set of variables/list variables updated in the command C, namely if C is of the form $x := \mathtt{add_vertex}(d)$ or $x.\mathsf{data} := d$ then $\mathsf{Mod}(\mathsf{C}) = \{x\}$, if C is of the form $\mathtt{enqueue}(\mathsf{L}, x)$ or $\mathsf{L} := \mathsf{LExp}$ then $\mathsf{Mod}(\mathsf{C}) = \{\mathsf{L}\}$, and $\mathsf{Mod}(x := \mathtt{dequeue}(\mathsf{L})) = \{x, \mathsf{L}\}$; $\mathsf{Mod}(\mathsf{C})$ is empty for the remaining cases. $[\mathsf{L}]\mathsf{Var}(R)$ denotes the set of variables and list variables that occur in R.

$\{\mathsf{e}\}$
$x := \texttt{add_vertex}(d)$
$\{\mathsf{region}(d, \{x\}, \varnothing)\}$

$\{\mathsf{region}(d, \{x\}, \mathsf{T})\}$
$\texttt{del_vertex}(x)$
$\{\mathsf{e}\}$

$\{\mathsf{region}(d_1, \{x\}, \mathsf{T}_1) * \mathsf{region}(d_2, \{y\}, \mathsf{T}_2)\}$
$\texttt{add_edge}(x, y)$
$\{\mathsf{region}(d_1, \{x\}, \mathsf{T}_1 \uplus \{(x, y)\}) * \mathsf{region}(d_2, \{y\}, \mathsf{T}_2 \uplus \{(x, y)\})\}$

$\{\mathsf{region}(d_1, \{x\}, \mathsf{T}_1) * \mathsf{region}(d_2, \{y\}, \mathsf{T}_2)\}$
$\texttt{del_edge}(x, y)$
$\{\mathsf{region}(d_1, \{x\}, \mathsf{T}_1 \setminus\setminus \{(x, y)\}) * \mathsf{region}(d_2, \{y\}, \mathsf{T}_2 \setminus\setminus \{(x, y)\})\}$

$\{\mathsf{L} = \mathsf{LExp}\}$
$\texttt{enqueue}(\mathsf{L}, x)$
$\{\mathsf{L} = \mathsf{LExp}; x\}$

$\{\mathsf{L} = y; \mathsf{LExp}\}$
$x := \texttt{dequeue}(\mathsf{L})$
$\{x = y * \mathsf{L} = \mathsf{LExp}\}$

$\{\mathsf{region}(d_2, \{x\}, \mathsf{T})\}$
$x.\mathsf{data} := d_1$
$\{\mathsf{region}(d_1, \{x\}, \mathsf{T})\}$

$\{\mathsf{e}\}$
$\mathsf{L} := \mathsf{LExp}$
$\{\mathsf{e} \wedge \mathsf{L} = \mathsf{LExp}\}$

Table 4. Specifications for commands in Subsection 3.1.

Theorem 1 (Soundness) *The Frame Rule is sound.*

Proof. The proof follows the same steps as in [6], by replacing heaps with pregraphs. As in the work mentioned, the proof relies heavily on the results on Safety Monotonicity (Lemma 1) and Frame Property (Lemma 2) previously stated.

We leave a concrete proof of completeness for the future. However, it is reasonable to assume that our system is complete. Observe that the Frame Rule is sound, and furthermore, the Rule of Consequence is sound as well. Therefore the typical "ingredients" at the base of completeness in SL are present in SPGL too.

$$\frac{\varphi' \Rightarrow \varphi \quad \{\varphi\}\mathsf{C}\{\psi\} \quad \psi \Rightarrow \psi'}{\{\varphi'\}\mathsf{C}\{\psi'\}}$$

4 (Pre)Graph algorithms examples

In this section, we demonstrate how SPGL can be used to construct verification proofs of (pre)graph algorithms. The first is a very simple example; afterwards we proceed to the verification of the Breadth-first search algorithm.

4.1 `new_client`: a (pre)graph changing program

Consider the toy example in Figure 7. The program takes a list of vertices V and considers a pregraph whose owned vertices are those in V. It adds a new vertex to the pregraph and then adds full edges from every pre-existing vertex into the new one – we call this new vertex a "*new client*". In order to make this a simple example, there is no data associated with the vertices; therefore we will keep notation lighter and omit the entries that refer to it. We will also omit the fact that *subformulas* containing only variable assignments (as $x = y$, $\mathsf{L} = \mathsf{L}'$) are satisfied in empty pregraphs ($x = y \wedge \mathsf{e}$, $\mathsf{L} = \mathsf{L}' \wedge \mathsf{e}$).

We prove that the program satisfies the specification

$\{\mathsf{region}(\{\!|\mathrm{V}|\!\}, \mathsf{T})\}$
$\quad \texttt{new_client(V)}$
$\{\mathsf{region}(\{\!|\mathrm{V}|\!\}, \mathsf{T} \uplus \{(y, x), y \in \{\!|\mathrm{V}|\!\}\}) * \mathsf{region}(\{x\}, \{(y, x), y \in \{\!|\mathrm{V}|\!\}\})\}$

```
new_client(V){
    Q:=V;
    x:=add_vertex();
    while(Q ≠ []){
        y:=dequeue(Q);
        add_edge(y, x);
    }
}
```

Fig. 7. A program to add a *new client*.

where $\{\!|\mathrm{V}|\!\}$ is the set of all elements in list V. We will use analogous notation going forward. The proof is in Figure 8 and makes use of previous results on the manipulation of region formulas from Subsection 3.3. In summary, in the proof we introduce a new list variable which is assigned the list of vertices in the input and add a new vertex according to specifications in Table 4. Inside the `while` loop, the variable `Q` is always assigned to a list with at least one element and the size of the pregraph does not change – the pregraph is constituted by the vertices in $s(\{\!|\mathrm{V}|\!\})$ plus the new vertex $s(x)$. At each iteration, the sets T_1 and T_2 are incremented with edges of the form (y, x) for y the vertex previously dequeued. In fact, at each step, $\mathsf{T}_1 = \mathsf{T} \uplus \{(y, x) \mid y \in \{\!|\mathrm{V}|\!\} \setminus\!\setminus \{\!|y; \mathsf{L}|\!\}\}$ and $\mathsf{T}_2 = \{(y, x) \mid v \in \{\!|\mathrm{V}|\!\} \setminus\!\setminus \{\!|y; \mathsf{L}|\!\}\}$. This observation provides us with the invariant for the `while` loop:

$$\begin{aligned}&\mathsf{region}(\{\!|\mathrm{V}|\!\}, \mathsf{T} \uplus \{(y, x) \mid y \in \{\!|\mathrm{V}|\!\} \setminus\!\setminus \{\!|\mathsf{L}|\!\}\})\\ * \ &\mathsf{region}(\{x\}, \{(y, x) \mid y \in \{\!|\mathrm{V}|\!\} \setminus\!\setminus \{\!|\mathsf{L}|\!\}\}) * \mathsf{Q} = \mathsf{L}\end{aligned}$$

.

Therefore, when exiting the loop: $\mathsf{region}(\{\!|\mathrm{V}|\!\}, \mathsf{T} \uplus \{(y, x) \mid v \in \{\!|\mathrm{V}|\!\}\}) * \mathsf{region}(\{x\}, \{(y, x) \mid y \in \{\!|\mathrm{V}|\!\}\}) * \mathsf{Q} = []$ holds. As we mentioned before, the variable assignment holds in an empty region, thus it ends being omitted.

```
{region({|V|}, T)}
new_client(V)
Q:=V;
{region({|V|}, T) ∗ Q = V}
x:=add_vertex();
{region({|V|}, T) ∗ region({x}, ∅) ∗ Q = V}
while(Q ≠ []){
{region({|V|}, T1) ∗ region({x}, T2) ∗ Q = w; L}
y:=dequeue(Q);
{region({|V|}, T1) ∗ region({x}, T2) ∗ y = w ∗ Q = L}
{region({|V|} \\ {y}, T1||{|V|}\\{y}) ∗ region({y}, T1||{y})
  ∗ region({x}, T2) ∗ y = w ∗ Q = L}                               by Lemma 3
add_edge(y, x);
{region({|V|} \\ {y}, T1||{|V|}\\{y}) ∗ region({y}, T1||{y} ⊎ {(y, x)})
  ∗ region({x}, T2 ⊎ {(y, x)}) ∗ y = w ∗ Q = L}
{region({|V|}, T1 ⊎ {(y, x)}) ∗ region({x}, T2 ⊎ {(y, x)}) ∗ y = w ∗ Q = L}   by Lemma 4
}
{region({|V|}, T ⊎ {(y, x) | y ∈ {|V|}}) ∗ region({x}, {(y, x), y ∈ {|V|}}) ∗ Q = []}
}
{region({|V|}, T ⊎ {(y, x) | y ∈ {|V|}}) ∗ region({x}, {(y, x), y ∈ {|V|}})}
```

Fig. 8. Verification proof of the program in 4.1.

4.2 Breadth-first search algorithm verification

In this section we use SPGL and the usual principles of SL to construct the verification proof of the Breadth-first search (BFS) algorithm [2] whose goal is

to, given an undirected graph and a source vertex, mark all reachable vertices from the source. Since we are dealing with undirected (pre)graphs, we consider the sets of edges E and the sets T in region formulas to be sets of two-sets $\{a, a'\}$ instead of pairs (a, a').

From the point of view of the algorithm, a graph is composed by three (possibly empty, but definitely disjoint) regions, which are defined by clustering vertices according to their data: in this case, one of the colours white, grey or black. Simply put, white vertices correspond to undiscovered vertices, grey vertices correspond to vertices that are about to be explored, and black vertices correspond to vertices already visited. We will use $d(\mathsf{S}, \mathsf{T})$ to abbreviate $\mathsf{region}(d, \mathsf{S}, \mathsf{T})$.

In the beginning, all vertices of the graph G are white, except for the source s which we already consider grey. The algorithm creates a queue Q to accommodate s; in fact, this queue will accommodate all the vertices that are coloured grey in the future. While the queue is not empty, meaning there are vertices left to explore, the algorithm picks (and removes) the first element from the queue, and immediately colours it black. Given that at this point nothing tells us exactly to which region does that element belong, we simply "split" it from all regions: note that since it belongs to a unique region, removing it from all others produces no changes at all in the pregraph. Here is a zoomed in window on that step for some intuition:

$$\begin{aligned}
&\mathtt{white}(X, A) * \mathtt{grey}(Y, B) * \mathtt{black}(Z, C)\\
&\quad\models \mathtt{white}(X \setminus\!\!\setminus \{x\}, A\|_{X \setminus\!\setminus \{x\}}) * \mathtt{white}(X \Cap \{x\}, A\|_{X \Cap \{x\}})\\
&\qquad * \ \mathtt{grey}(Y \setminus\!\!\setminus \{x\}, B\|_{Y \setminus\!\setminus \{x\}}) * \mathtt{grey}(Y \Cap \{x\}, B\|_{Y \Cap \{x\}})\\
&\qquad * \ \mathtt{black}(Z \setminus\!\!\setminus \{x\}, C\|_{Z \setminus\!\setminus \{x\}}) * \mathtt{black}(Z \Cap \{x\}, C\|_{Z \Cap \{x\}})
\end{aligned}$$

Given that x can only be in one of X, Y, Z, the previous entails the following: $\mathtt{white}(X \setminus\!\!\setminus \{x\}, A\|_{X \setminus\!\setminus \{x\}}) * \mathtt{grey}(Y \setminus\!\!\setminus \{x\}, B\|_{Y \setminus\!\setminus \{x\}}) * \mathtt{black}(Z \setminus\!\!\setminus \{x\}, C\|_{Z \setminus\!\setminus \{x\}}) * \ \square(\{x\}, (A \Cup B \Cup C)\|_{\{x\}})$, for $\square \in \{\mathtt{white}, \mathtt{grey}, \mathtt{black}\}$.

From this *imprecise* state, assigning a colour to the region that is constituted by x is actually possible. Thus in the proof we find $\mathtt{white}(\overline{X}, A\|_{\overline{X}}) * \mathtt{grey}(\overline{Y}, B\|_{\overline{Y}}) * \mathtt{black}(\overline{Z}, C\|_{\overline{Z}}) * \mathtt{black}(\{x\}, (A \Cup B \Cup C)\|_{\{x\}})$, where $\overline{\mathsf{S}}$ is an abbreviation for $\mathsf{S} \setminus\!\!\setminus \{x\}$.

Now comes the second peculiarity of this example: the algorithm creates a new queue to accommodate variables adjacent to x. We consider $\mathtt{Adj}(x)$ a particular type of list variable that takes direct influence from the pregraph G, namely: $\{\!|s(\mathtt{Adj}(x))|\!\} = \{y \mid \{y, x\} \in \mathrm{E}_{\mathsf{G}}\}$. While this queue is not empty, the algorithm dequeues its first variable, which in case is white is added to the queue Q. When we exit this `while`, we begin a new iteration of the main loop.

At the end of the algorithm, the set of all variables(/vertices) that are reachable from the source are coloured black and all the others remain white. We prove in Figure 9 that the algorithm satisfies the specification

$$\begin{aligned}
&\{\mathtt{white}(\mathsf{S}, \mathsf{T}) * \mathtt{grey}(\varnothing, \varnothing) * \mathtt{black}(\varnothing, \varnothing)\}\\
&\quad \mathtt{BFS(G,s)}\\
&\{\mathtt{white}(\mathsf{S} \setminus \mathsf{S}', \mathsf{T}') * \mathtt{grey}(\varnothing, \varnothing) * \mathtt{black}(\mathsf{S}', \mathsf{T}'')\}
\end{aligned}$$

$\{\mathtt{white}(\mathsf{S},\mathsf{T}) * \mathtt{grey}(\{s\},\ \mathsf{T}\|_{\{s\}}) * \mathtt{black}(\varnothing,\varnothing)\}$

`BFS(G,`s`){`

`Q:=`s`;[]`

$\{\mathtt{white}(\mathsf{S},\mathsf{T}) * \mathtt{grey}(\{s\},\ \mathsf{T}\|_{\{s\}}) * \mathtt{black}(\varnothing,\varnothing) * \mathsf{Q} = s;[]\}$

`while(Q` $\neq$ `[]){`

$\{\mathtt{white}(X,A) * \mathtt{grey}(Y,B) * \mathtt{black}(Z,C) * \mathsf{Q} = w;\mathsf{L}\}$

x`:=dequeue(Q)`

$\{\mathtt{white}(X,A) * \mathtt{grey}(Y,B) * \mathtt{black}(Z,C) * x = w * \mathsf{Q} = \mathsf{L}\}$

x`.colour:=black`

$\{\mathtt{white}(X\setminus\!\!\setminus\{x\},\ A\|_{X\setminus\!\setminus\{x\}}) * \mathtt{grey}(Y\setminus\!\!\setminus\{x\},\ B\|_{Y\setminus\!\setminus\{x\}}) * \mathtt{black}(Z\setminus\!\!\setminus\{x\},\ C\|_{Z\setminus\!\setminus\{x\}})$
$\quad * \mathtt{black}(\{x\},\ (A \Cup B \Cup C)\|_{\{x\}}) * x = w * \mathsf{Q} = \mathsf{L}\}$

Q'`:=`$Adj(x)$

$\{\mathtt{white}(\overline{X}),\ A\|_{\overline{X}}) * \mathtt{grey}(\overline{Y},\ B\|_{\overline{Y}}) * \mathtt{black}(\overline{Z},\ C\|_{\overline{Z}})$
$\quad * \mathtt{black}(\{x\},\ (A \Cup B \Cup C)\|_{\{x\}}) * x = w * \mathsf{Q} = \mathsf{L} * \mathsf{Q}' = Adj(x)\}$

`while(`$\mathsf{Q}' \neq []$`){`

$\{\mathtt{white}(X_1,A_1) * \mathtt{grey}(Y_1,B_1) * \mathtt{black}(Z_1,C_1) * x = w * \mathsf{Q} = \mathsf{L} * \mathsf{Q}' = z;\mathsf{L}_2\}$

y`:=dequeue(`Q'`)`

$\{\mathtt{white}(X_1,A_1) * \mathtt{grey}(Y_1,B_1) * \mathtt{black}(Z_1,C_1) * x = w * \mathsf{Q} = \mathsf{L} * y = z * \mathsf{Q}' = \mathsf{L}_2\}$

`if(`y`.colour=white){`

$\{\mathtt{white}(X_1\setminus\!\!\setminus\{y\},\ A_1\|_{X_1\setminus\!\setminus\{y\}}) * \mathtt{white}(\{y\},\ A_1\|_{\{y\}})$
$\quad * \mathtt{grey}(Y_1,B_1) * \mathtt{black}(Z_1,C_1) * x = w * \mathsf{Q} = \mathsf{L} * y = z * \mathsf{Q}' = \mathsf{L}_2\}$ by Lemma 3

`y.colour:=grey`

$\{\mathtt{white}(X_1\setminus\!\!\setminus\{y\},\ A_1\|_{X_1\setminus\!\setminus\{y\}}) * \mathtt{grey}(\{y\},\ A_1\|_{\{y\}})$
$\quad * \mathtt{grey}(Y_1,B_1) * \mathtt{black}(Z_1,C_1) * x = w * \mathsf{Q} = \mathsf{L} * y = z * \mathsf{Q}' = \mathsf{L}_2\}$

`enqueue(Q,`y`)`

$\{\mathtt{white}(X_1\setminus\!\!\setminus\{y\},\ A_1\|_{X_1\setminus\!\setminus\{y\}}) * \mathtt{grey}(\{y\},\ A_1\|_{\{y\}})$
$\quad * \mathtt{grey}(Y_1,B_1) * \mathtt{black}(Z_1,C_1) * x = w * \mathsf{Q} = \mathsf{L};y * y = z * \mathsf{Q}' = \mathsf{L}_2\}$

$\{\mathtt{white}(X_1\setminus\!\!\setminus\{y\},\ A_1\|_{X_1\setminus\!\setminus\{y\}}) * \mathtt{grey}(Y_1 \Cup \{y\}, B_1 \Cup A_1\|_{\{y\}})$
$\quad * \mathtt{black}(Z_1,C_1) * x = w * \mathsf{Q} = \mathsf{L};y * y = z * \mathsf{Q}' = \mathsf{L}_2\}$ by Lemma 4

}

}

$\{\mathtt{white}(\overline{X}\setminus\!\!\setminus(\{k \mid \{k,x\} \in \mathsf{T}\} \Cap \overline{X}),\ A\|_{\overline{X}\setminus\!\setminus(\{k \mid \{k,x\}\in\mathsf{T}\}\Cap\overline{X})})$
$\quad * \mathtt{grey}(\overline{Y} \Cup (\{k \mid \{k,x\} \in \mathsf{T}\} \Cap \overline{X}),\ B\|_{\overline{Y}} \Cup A\|_{\{k \mid \{k,x\}\in\mathsf{T}\}\Cap\overline{X}})$
$\quad * \mathtt{black}(\overline{Z} \Cup \{x\},\ C\|_{\overline{Z}} \Cup (A \Cup B \Cup C)\|_{\{x\}})$
$\quad * x = w * \mathsf{Q} = \mathsf{L}; \llbracket\{k \mid \{k,x\} \in \mathsf{T}\} \Cap \overline{X}\rrbracket * \mathsf{Q}' = []\}$

}

$\{\mathtt{white}(\mathsf{S}\setminus\!\!\setminus\{k \mid \exists k_1,\ldots,k_n \text{ s.t. } \{k_1,k_2\},\ldots,\{k_{n-1},k_n\} \in \mathsf{T}, k_1 = s, k_n = k, k \in \mathsf{S}\},$
$\quad \mathsf{T}\|_{\mathsf{S}\setminus\!\setminus\{k \mid \exists k_1,\ldots,k_n \text{ s.t. } \{k_1,k_2\},\ldots,\{k_{n-1},k_n\}\in\mathsf{T},k_1=s,k_n=k,k\in\mathsf{S}\}})$
$\quad * \mathtt{grey}(\varnothing,\varnothing) * \mathtt{black}(\{k \mid \exists k_1,\ldots,k_n \text{ s.t. } \{k_1,k_2\},\ldots,\{k_{n-1},k_n\} \in \mathsf{T}, k_1 = s, k_n = k, k \in \mathsf{S}\},$
$\quad \mathsf{T}\|_{\{k \mid \exists k_1,\ldots,k_n \text{ s.t. } \{k_1,k_2\},\ldots,\{k_{n-1},k_n\}\in\mathsf{T},k_1=s,k_n=k,k\in\mathsf{S}\}} * Q = [])\}$

}

Fig. 9. Verification proof of the Breadth-first search algorithm.

where S' is the set of vertices in the same connected component as s and the sets of edges $\mathsf{T}', \mathsf{T}''$ are obtained by restricting T to the set of edges where at least one of the elements in each two-set is in the region `white` or `black`, respectively.

Invariants I for the `while` loops in the algorithm can be stated as follows: at each iteration of the `while(`$\mathsf{Q}' \neq []$`)` loop, the following holds:

$$\begin{aligned} I = {} & \mathtt{white}(\overline{X}\setminus\!\!\setminus(M \Cap \overline{X}),\ A\|_{\overline{X}\setminus\!\setminus(M\Cap\overline{X})}) \\ & * \ \mathtt{grey}(\overline{Y} \Cup (M \Cap \overline{X}),\ B\|_{\overline{Y}} \Cup A\|_{M\Cap\overline{X}}) \\ & * \ \mathtt{black}(\overline{Z} \Cup \{x\},\ C\|_{\overline{Z}} \Cup (A \Cup B \Cup C)\|_{\{x\}}) \\ & * \ x = w * \mathsf{Q} = \mathsf{L}; \llbracket M \Cap \overline{X}\rrbracket * \mathsf{Q}' = Adj(x)\setminus\!\!\setminus\llbracket M\rrbracket \end{aligned}$$

where $\llbracket M \Cap \overline{X}\rrbracket$ is a list whose elements are in the set $M \Cap \overline{X}$ and $Adj(x)\setminus\!\!\setminus\llbracket M\rrbracket$ should be interpreted as the list of variables adjacent to x that are not in M.

For the main loop, $\texttt{while(Q} \neq []\texttt{)}$ the invariant takes the form:

$$I = \texttt{white}(\mathsf{S}\backslash(\{\!|\mathsf{L}|\!\} \cup J),\, \mathsf{T}|_{\mathsf{S}\backslash(\{\!|\mathsf{L}|\!\}\cup J)}) * \texttt{grey}(\{\!|\mathsf{L}|\!\},\, \mathsf{T}|_{\{\!|\mathsf{L}|\!\}})$$
$$* \texttt{black}(J,\, \mathsf{T}|_J) * \mathsf{Q} = \mathsf{L}$$

where at each iteration, if the elements in J are reachable in at most k steps, then those in $\{\!|\mathsf{L}|\!\}$ are reachable in at most $k+1$ steps. At the end of the algorithm we obtain a black region of variables reachable in n steps and no other variable is reachable in more than n steps. The white region contains all variables not reachable from the source.

5 Conclusions and future work

Being able to split graphs into smaller parts has always constituted a problem. More often than not, the remainder of a graph after a subgraph is extracted is not a graph. However, one certainly recognizes that being able to zoom in into a region of a graph, make changes and posteriorly zoom out is much desirable in several contexts. In order to overcome the issues that arise when one tries to split graphs, we introduced a new concept, that of a pregraph. Simply put, a pregraph is a kind of graph where some vertices may be missing, under the constraints that missing vertices are part of an edge and that at most one vertex per edge is missing. After a detailed study of notions of composition for pregraphs, we settled on a definition that makes the set of all pregraphs, the composition and the empty pregraph (the unit), a BBI-model.

We introduced a separation-like logic where instead of heaps we consider pregraphs, in an attempt to provide a formal way of verifying graph algorithms. We were indeed able to prove the correctness of a small toy example and of the well-known Breadth-first search algorithm. While we cannot claim that SPGL is equipped to deal with all kinds of graph algorithms, it certainly feels like it is the first step in the right direction. In the future we intend to use this new logic and by adding new extra machinery, be able to prove the correctness of more complex algorithms, such as the Ford-Fulkerson algorithm, which involves weights on edges.

One of the main ingredients of the logic is the presence of a sound frame rule, which is what enables the proofs of the examples presented. In fact, the frame rule used is analogous to the rule found in standard Separation logic. A detailed account on completeness is left as future work. An in-depth study of complexity and decidability for SPGL are also topics to address later on.

Acknowledgements. This work has been supported by the UK EPSRC project EP/R006865/1.

References

1. J. Brotherston and J. Villard. Parametric completeness for separation theories. In *Proceedings of POPL-41*, pages 453–464. ACM, 2014.

2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
3. S. Ishtiaq and P. W. O'Hearn. BI as an assertion language for mutable data structures. In *Proceedings of POPL-28*, pages 14–26. ACM, 2001.
4. P. W. O'Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 1999.
5. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of LICS-17*, pages 55–74. IEEE Computer Society, 2002.
6. H. Yang and P. O'Hearn. A semantic basis for local reasoning. In *Proceedings of FOSSACS-5*, pages 402–416. Springer, 2002.

Appendix

Proposition 1 Let PreGs be the set of all pregraphs over vertex names $\mathbb{V}$, with the composition $\circ$ as defined in Definition 3. Then $\langle \mathsf{PreGs}, \circ, \mathsf{e} \rangle$ is a BBI-model.

Additionally, it satisfies the following separation theory properties:

1. Cancellativity: $\mathsf{G} \circ \mathsf{H}_1 = \mathsf{G} \circ \mathsf{H}_2 \Rightarrow \mathsf{H}_1 = \mathsf{H}_2$;
2. Indivisible Unit: $\mathsf{G} \circ \mathsf{H} = \mathsf{e} \Rightarrow \mathsf{G} = \mathsf{H} = \mathsf{e}$;
3. Disjointness: $\mathsf{G} \circ \mathsf{G} = \mathsf{H} \Rightarrow \mathsf{G} = \mathsf{H} = \mathsf{e}$;
4. Cross-split: $\mathsf{G} \circ \mathsf{H} = \mathrm{I} \circ \mathrm{J} \Rightarrow \exists \mathsf{G}_1, \mathsf{G}_2, \mathsf{H}_1, \mathsf{H}_2.\ \mathsf{G} = \mathsf{G}_1 \circ \mathsf{G}_2, \mathsf{H} = \mathsf{H}_1 \circ \mathsf{H}_2, \mathrm{I} = \mathsf{G}_1 \circ \mathsf{H}_1, \mathrm{J} = \mathsf{G}_2 \circ \mathsf{H}_2$.

Proof. The triple $\langle \mathsf{PreGs}, \circ, \mathsf{e} \rangle$ is a BBI-model if $\circ$ is commutative, associative, and $\mathsf{G} \circ \mathsf{e} = \mathsf{G}$ for all $\mathsf{G} \in \mathsf{PreGs}$. It is trivial to prove that the first and third conditions hold. In order to prove associativity of $\circ$, assume that $\mathsf{G}_1 \circ (\mathsf{G}_2 \circ \mathsf{G}_3)$ is defined. Then, by definition, $\mathsf{G}_1 \# (\mathsf{G}_2 \circ \mathsf{G}_3)$ and G_1 and $\mathsf{G}_2 \circ \mathsf{G}_3$ agree on half edges and $\mathsf{G}_2 \# \mathsf{G}_3$ and G_2 and G_3 agree on half edges. Since $\mathrm{V}_1 \cap (\mathrm{V}_2 \cup \mathrm{V}_3) = \varnothing$ and $\mathrm{V}_2 \cap \mathrm{V}_3 = \varnothing$, it follows that $\mathrm{V}_1 \cap \mathrm{V}_2 = \varnothing$ and $(\mathrm{V}_1 \cup \mathrm{V}_2) \cap \mathrm{V}_3 = \varnothing$.

We need to prove that G_1 and G_2 agree on half edges, so that $\mathsf{G}_1 \circ \mathsf{G}_2 \downarrow$, and then prove that $\mathsf{G}_1 \circ \mathsf{G}_2$ and G_3 agree on half edges too. Take a half edge $(u, v) \in \mathrm{E}_1$ and assume, W.L.G., that $u \in \mathrm{V}_1$, $v \in \mathrm{V}_2$. Then $v \in \mathrm{V}_2 \cup \mathrm{V}_3$. Since G_1 and $\mathsf{G}_2 \circ \mathsf{G}_3$ agree on half edges, $(u, v) \in \mathrm{E}_2 \cup \mathrm{E}_3$. Moreover, because $\mathrm{V}_1 \cap \mathrm{V}_2 = \mathrm{V}_1 \cap \mathrm{V}_3 = \mathrm{V}_2 \cap \mathrm{V}_3 = \varnothing$, $u, v \notin \mathrm{V}_3$ and therefore $(u, v) \in \mathrm{E}_2$. For a dangling edge $(u, v) \in \mathrm{E}_2$ such that, W.L.G., $u \in \mathrm{V}_2, v \in \mathrm{V}_1$, observe that $(u, v) \in \mathrm{E}_2 \cup \mathrm{E}_3$ and $u \in \mathrm{V}_2 \cup \mathrm{V}_3$. This implies that $(u, v) \in \mathrm{E}_1$, given that G_1 and $\mathsf{G}_2 \circ \mathsf{G}_3$ agree on half edges. This concludes the proof that G_1 and G_2 agree on half edges.

In order to prove that $\mathsf{G}_1 \circ \mathsf{G}_2$ and G_3 agree on half edges, take a half edge $(u, v) \in \mathrm{E}_1 \cup \mathrm{E}_2$ and consider, W.L.G., that $u \in \mathrm{V}_1 \cup \mathrm{V}_2$, $v \in \mathrm{V}_3$. If $u \in \mathrm{V}_1$, $(u, v) \in \mathrm{E}_1$. Also, $v \in \mathrm{V}_2 \cup \mathrm{V}_3$. Since G_1 and $\mathsf{G}_2 \circ \mathsf{G}_3$ agree on half edges, $(u, v) \in \mathrm{E}_2 \cup \mathrm{E}_3$. Additionally, $u, v \notin \mathrm{V}_2$ implies that $(u, v) \notin \mathrm{E}_2$ and thus $(u, v) \in \mathrm{E}_3$. If, on the other hand, $u \in \mathrm{V}_2$, then $(u, v) \in \mathrm{E}_2$, which implies $(u, v) \in \mathrm{E}_3$ as G_2 and G_3 agree on half edges. For a dangling edge $(u, v) \in \mathrm{E}_3$ such that, W.L.G., $u \in \mathrm{V}_3$, $v \in \mathrm{V}_1 \cup \mathrm{V}_2$, in case $v \in \mathrm{V}_1$, observe that $u \in \mathrm{V}_2 \cup \mathrm{V}_3$ and $(u, v) \in \mathrm{E}_2 \cup \mathrm{E}_3$. Given that G_1 and $\mathsf{G}_2 \circ \mathsf{G}_3$ agree on half edges, $(u, v) \in \mathrm{E}_1$. Case $v \in \mathrm{V}_2$ it immediately follows that $(u, v) \in \mathrm{E}_2$ as G_2 and G_3 agree on half edges. Thus $(u, v) \in \mathrm{E}_1 \cup \mathrm{E}_2$. This concludes the proof that $\mathsf{G}_1 \circ \mathsf{G}_2$ and G_3 agree on half edges. Consequently $\circ$ is commutative.

We now prove the additional properties listed:

1. Cancellativity: Assume that $\mathsf{G}, \mathsf{H}_1, \mathsf{H}_2 \in \mathsf{PreGs}$ and $\mathsf{G} \circ \mathsf{H}_1 = \mathsf{G} \circ \mathsf{H}_2$. Then $(\mathrm{V}_{\mathsf{G}} \cup \mathrm{V}_{\mathsf{H}_1}, \mathrm{E}_{\mathsf{G}} \cup \mathrm{E}_{\mathsf{H}_1}) = (\mathrm{V}_{\mathsf{G}} \cup \mathrm{V}_{\mathsf{H}_2}, \mathrm{E}_{\mathsf{G}} \cup \mathrm{E}_{\mathsf{H}_2})$. Since $\mathsf{G} \circ \mathsf{H}_1$ and $\mathsf{G} \circ \mathsf{H}_2$ are defined, $\mathrm{V}_{\mathsf{G}} \cap \mathrm{V}_{\mathsf{H}_1} = \mathrm{V}_{\mathsf{G}} \cap \mathrm{V}_{\mathsf{H}_2} = \varnothing$. Therefore, $\mathrm{V}_{\mathsf{G}} \cup \mathrm{V}_{\mathsf{H}_1} = \mathrm{V}_{\mathsf{G}} \cup \mathrm{V}_{\mathsf{H}_2}$ implies $\mathrm{V}_{\mathsf{H}_1} = \mathrm{V}_{\mathsf{H}_2}$.

We prove by contradiction that $\mathrm{E}_{\mathsf{H}_1} = \mathrm{E}_{\mathsf{H}_2}$, so take $(u, v) \in \mathrm{E}_{\mathsf{H}_1}$ and $(u, v) \notin \mathrm{E}_{\mathsf{H}_2}$. From the first assumption it follows that $(u, v) \in \mathrm{E}_{\mathsf{G} \circ \mathsf{H}_1}$. Then, since $\mathsf{G} \circ \mathsf{H}_1 = \mathsf{G} \circ \mathsf{H}_2$, it implies that $(u, v) \in \mathrm{E}_{\mathsf{G} \circ \mathsf{H}_2}$. Given that we assumed that $(u, v) \notin \mathrm{E}_{\mathsf{H}_2}$, it must be the case that $(u, v) \in \mathrm{E}_{\mathsf{G}}$. So, $(u, v) \in \mathrm{E}_{\mathsf{H}_1} \cap \mathrm{E}_G$ and therefore (u, v)

is a dangling edge in both H_1 and G, which in turn implies that (u, v) is a full edge in $\mathsf{G} \circ \mathsf{H}_1$. However, (u, v) is not a full edge in $\mathsf{G} \circ \mathsf{H}_2$, since $(u, v) \notin \mathrm{E}_{\mathsf{H}_2}$. This contradicts the fact that $\mathsf{G} \circ \mathsf{H}_1 = \mathsf{G} \circ \mathsf{H}_2$; thus $\mathrm{E}_{\mathsf{H}_1} = \mathrm{E}_{\mathsf{H}_2}$.

2. Indivisible unit: Take pregraphs $\mathsf{G}, \mathsf{H} \in \mathsf{PreGs}$ such that $\mathsf{G} \circ \mathsf{H} = \mathsf{e}$. Then $(\mathrm{V}_\mathsf{G} \cup \mathrm{V}_\mathsf{H}, \mathrm{E}_\mathsf{G} \cup \mathrm{E}_\mathsf{H}) = (\varnothing, \varnothing)$, which implies that $\mathrm{V}_\mathsf{G} = \mathrm{V}_\mathsf{H} = \mathrm{E}_\mathsf{G} = \mathrm{E}_\mathsf{H} = \varnothing$. Thus $\mathsf{G} = \mathsf{H} = \mathsf{e}$.

3. Disjointness: Assume that $\mathsf{G} \circ \mathsf{G} = \mathsf{H}$ for some pregraphs $\mathsf{G}, \mathsf{H} \in \mathsf{PreGs}$. By definition, the composition is defined only if $\mathrm{V}_\mathsf{G} \cap \mathrm{V}_\mathsf{G} = \varnothing$, which implies that $\mathrm{V}_\mathsf{G} = \varnothing$. By the definition of pregraph it follows that $\mathrm{E}_\mathsf{G} = \varnothing$. Consequently $\mathsf{G} = \mathsf{H} = \mathsf{e}$.

4. Cross-split: The decomposition of pregraphs is always possible, so there exist $\mathsf{G}_1, \mathsf{G}_2$ such that $\mathsf{G} = \mathsf{G}_1 \circ \mathsf{G}_2$ for every possible choice of $\mathrm{V}_{\mathsf{G}_1}, \mathrm{V}_{\mathsf{G}_2}$ as long as $\mathrm{V}_{\mathsf{G}_1} \cap \mathrm{V}_{\mathsf{G}_2} = \varnothing$ and $\mathrm{V}_{\mathsf{G}_1} \cup \mathrm{V}_{\mathsf{G}_2} = \mathrm{V}_\mathsf{G}$. Analogously for the decomposition of H. Therefore, $\mathsf{G} \circ \mathsf{H} = (\mathsf{G}_1 \circ \mathsf{G}_2) \circ (\mathsf{H}_1 \circ \mathsf{H}_2)$. Associativity and commutativity of $\circ$ lead to $\mathsf{G} \circ \mathsf{H} = (\mathsf{G}_1 \circ \mathsf{H}_1) \circ (\mathsf{G}_2 \circ \mathsf{H}_2)$. Thus for appropriate choices of $\mathsf{G}_1, \mathsf{G}_2, \mathsf{H}_1, \mathsf{H}_2$, $\mathrm{I} = \mathsf{G}_1 \circ \mathsf{H}_1$ and $\mathrm{J} = \mathsf{G}_2 \circ \mathsf{H}_2$.

Lemma 1 (Safety Monotonicity) If $\langle \mathsf{C}, s, \mathsf{G} \rangle$ is safe, and $\mathsf{G} \circ \mathsf{G}'$ is defined, then $\langle \mathsf{C}, s, \mathsf{G} \circ \mathsf{G}' \rangle$ is also safe.

Proof. • Commands in group (i):

The commands `skip`, $\mathsf{C1}; \mathsf{C}_2$, `if` B `then` C `else` C' and `while` B `do` C are always safe.

• Commands in group (ii):

For $\mathsf{C} = x :=$ `add_vertex`(d), $\langle \mathsf{C}, s, \mathsf{G} \rangle$ is always safe, regardless of the choice of pregraph. Given that pregraphs own a finite number of vertices, and that the set $\mathbb{V}$ is countably infinite, it follows that for any given pregraph G there is always a vertex $u \in \mathbb{V} \backslash \overline{\mathrm{V}}$ such that $\langle u, d \rangle$ that can be added to V.

For $\mathsf{C} =$ `add_edge`(x, y), since $\langle \mathsf{C}, s, \mathsf{G} \rangle$ is safe, $s(x), s(y) \in \mathrm{V}$, thus the edge $(\overline{s}(x), \overline{s}(y))$ can be added to the set of edges E of the pregraph. Assuming that $\mathsf{G} \circ \mathsf{G}' \downarrow$, it immediately follows that $s(x), s(y) \in \mathrm{V} \cup \mathrm{V}'$ and thus the edge $(\overline{s}(x), \overline{s}(y))$ can be added to $\mathrm{E} \cup \mathrm{E}'$. Therefore $\langle \mathsf{C}, s, \mathsf{G} \circ \mathsf{G}' \rangle$ is safe.

For $\mathsf{C} =$ `del_vertex`(x), since $\langle \mathsf{C}, s, \mathsf{G} \rangle$ is safe, $s(x) \in \mathrm{V}$. So $s(x)$ can be removed from the set of vertices V and all dangling edges involving $\overline{s}(x)$ can be removed from the set of edges E. Assuming that $\mathsf{G} \circ \mathsf{G}' \downarrow$, it follows that $s(x) \in \mathrm{V} \cup \mathrm{V}'$. Therefore, $s(x)$ may be removed from $\mathrm{V} \cup \mathrm{V}'$ and all dangling edges involving $\overline{s}(x)$ can be removed from $\mathrm{E} \cup \mathrm{E}'$. Thus $\langle \mathsf{C}, s, \mathsf{G} \circ \mathsf{G}' \rangle$ is safe.

For $\mathsf{C} =$ `del_edge`(x, y), since $\langle \mathsf{C}, s, \mathsf{G} \rangle$ is safe, $s(x), s(y) \in \mathrm{V}$. Therefore the edge $(\overline{s}(x), \overline{s}(y))$ can be removed from the set of edges E. From the assumption that $\mathsf{G} \circ \mathsf{G}' \downarrow$, it is straightforward that $s(x), s(y) \in \mathrm{V} \cup \mathrm{V}'$, then the edge $(\overline{s}(x), \overline{s}(x))$ may be removed from the set of edges $\mathrm{E} \cup \mathrm{E}'$. In conclusion, $\langle \mathsf{C}, s, \mathsf{G} \circ \mathsf{G}' \rangle$ is safe.

• Commands in group (iii):

The safety of none of these commands depends on the pregraph, therefore if $\langle \mathsf{C}, s, \mathsf{G} \rangle$ is safe, then surely so is $\langle \mathsf{C}, s, \mathsf{G} \circ \mathsf{G}' \rangle$.

Lemma 2 (Frame property) If $\langle \mathsf{C}, s, \mathsf{G}_1 \circ \mathsf{G}_2 \rangle \rightsquigarrow^* \langle s', \mathsf{G}' \rangle$ and $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle$ is safe, then there exists s'', G'_1 such that $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle \rightsquigarrow^* \langle s'', \mathsf{G}'_1 \rangle$, $s'' \equiv s'$ and $\mathsf{G}' = \mathsf{G}'_1 \circ \mathsf{G}_2$.

Proof. Assume that for each case, $\langle \mathsf{C}, s, \mathsf{G}_1 \circ \mathsf{G}_2 \rangle \rightsquigarrow^* \langle s', \mathsf{G}' \rangle$ and $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle$ is safe.

- Commands in group (i):

Check the proof for standard Separation logic in [6].

- Commands in group (ii):

For $\mathsf{C} = x := \texttt{add_vertex}(d)$, $s' \overset{x}{\sim} s$, $\mathsf{G}' = (\mathrm{V}_1 \cup \mathrm{V}_2 \cup \{s'(x)\}, \mathrm{E}_1 \cup \mathrm{E}_2)$, where $s'(x) \notin \mathrm{V}_1 \cup \mathrm{V}_2$ is a new vertex with data d. From the assumption that $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle$ is safe, we reach the state $\langle s'', \mathsf{G}'_1 \rangle$, where $s'' \overset{x}{\sim} s$, $\mathsf{G}'_1 = (\mathrm{V}_1 \cup \{s''(x)\}, \mathrm{E}_1)$ and $s''(x) \notin \mathrm{V}_1$ is a new vertex with data d. We may consider $s'' \equiv s'$, from which it follows that $\mathsf{G}'_1 = (\mathrm{V}_1 \cup \{s'(x)\}, \mathrm{E}_1)$ and therefore $\mathsf{G}' = \mathsf{G}_1 \circ \mathsf{G}_2$.

For $\mathsf{C} = \texttt{add_edge}(x, y)$, from the first assumption follows that $s(x), s(y) \in \mathrm{V}_1 \cup \mathrm{V}_2$, $s' = s$ and $\mathsf{G}' = (\mathrm{V}_1 \cup \mathrm{V}_2, \mathrm{E}_1 \cup \mathrm{E}_2 \cup \{(\overline{s}(x), \overline{s}(y))\})$. The second assumption implies that $s(x), s(y) \in \mathrm{V}_1$. Therefore, $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle \rightsquigarrow^* \langle s, \mathsf{G}'_1 \rangle$, where $\mathsf{G}'_1 = (\mathrm{V}_1, \mathrm{E}_1 \cup \{(\overline{s}(x), \overline{s}(y))\})$. So $\mathsf{G}' = \mathsf{G}'_1 \circ \mathsf{G}_2$.

For $\mathsf{C} = \texttt{del_vertex}(x)$, the first assumption implies that $s(x) \in \mathrm{V}_1 \cup \mathrm{V}_2$. Also, $s' = s$ and $\mathsf{G}' = (\mathrm{V}', \mathrm{E}')$, where $\mathrm{V}' = (\mathrm{V}_1 \cup \mathrm{V}_2) \backslash \{s(x)\}$ and $\mathrm{E}' = (\mathrm{E}_1 \cup \mathrm{E}_2) \cap ((\overline{\mathrm{V}'} \times \mathbb{V}) \cup (\mathbb{V} \times \overline{\mathrm{V}'}))$. From the assumption that $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle$ is safe follows that $s(x) \in \mathrm{V}_1$ and $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle \rightsquigarrow^* \langle s, \mathsf{G}'_1 \rangle$ where $\mathsf{G}'_1 = (\mathrm{V}'_1, \mathrm{E}'_1)$ $\mathrm{V}'_1 = \mathrm{V}_1 \backslash \{s(x)\}$ and $\mathrm{E}'_1 = \mathrm{E}_1 \cap ((\overline{\mathrm{V}'_1} \times \mathbb{V}) \cup (\mathbb{V} \times \overline{\mathrm{V}'_1}))$. We now prove that $\mathsf{G}' = \mathsf{G}'_1 \circ \mathsf{G}_2$. In the first place, $\mathrm{V}' = \mathrm{V}'_1 \cup \mathrm{V}_2 = (\mathrm{V}_1 \backslash \{s(x)\}) \cup \mathrm{V}_2 = (\mathrm{V}_1 \cup \mathrm{V}_2) \backslash \{s(x)\}$ since $s(x) \in \mathrm{V}_1$ and $s(x) \notin \mathrm{V}_2$. As for the set of edges, E' is equal to the set E except for all dangling edges whose owned vertex is $\overline{s}(x)$; the construction of E'_1 is analogous. Given that $s(x) \notin \mathrm{V}_2$, there are no dangling edges in E_2 whose owned vertex is $\overline{s}(x)$. Therefore, $\mathrm{E}' = \mathrm{E}'_1 \cup \mathrm{E}_2$.

For $\mathsf{C} = \texttt{del_edge}(x, y)$, the first assumption implies that $s(x), s(y) \in \mathrm{V}_1 \cup \mathrm{V}_2$, $s' = s$ and $\mathsf{G}' = (\mathrm{V}', \mathrm{E}')$, where $\mathrm{V}' = \mathrm{V}_1 \cup \mathrm{V}_2$ and $\mathrm{E}' = (\mathrm{E}_1 \cup \mathrm{E}_2) \backslash \{(\overline{s}(x), \overline{s}(y))\}$. From the second assumption we obtain that $s(x), s(y) \in \mathrm{V}_1$. It follows that $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle \rightsquigarrow^* \langle s, \mathsf{G}'_1 \rangle$, where $\mathsf{G}'_1 = (\mathrm{V}_1, \mathrm{E}_1 \backslash \{(\overline{s}(x), \overline{s}(y))\})$. So $\mathrm{V}' = \mathrm{V}'_1 \cup \mathrm{V}_2$ and since $s(x), s(y) \notin \mathrm{V}_2$ and $(\overline{s}(x), \overline{s}(y)) \notin \mathrm{E}_2$, it follows that $\mathrm{E}' = \mathrm{E}'_1 \cup \mathrm{E}_2$. Thus $\mathsf{G}' = \mathsf{G}'_1 \circ \mathsf{G}_2$.

- Commands in group (iii):

For $\mathsf{C} = \texttt{enqueue}(\mathsf{L}, x)$, $s' \overset{\mathsf{L}}{\sim} s$, $s'(\mathsf{L}) = s(\mathsf{L}); s(x)$ and $\mathsf{G}' = \mathsf{G}_1 \circ \mathsf{G}_2$. From the assumption that $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle$ is safe follows that $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle \rightsquigarrow^* (s'', \mathsf{G}_1)$, where $s'' \overset{\mathsf{L}}{\sim} s$, $s''(x) = s(\mathsf{L}); s(x)$. Therefore, $s'' = s$ and the result follows.

For $\mathsf{C} = x := \texttt{dequeue}(\mathsf{L})$, $s' \overset{x,\mathsf{L}}{\sim} s$, $s'(x) = y$ and $s'(\mathsf{L}) = \mathsf{LExp}$ when $s(\mathsf{L}) = y; \mathsf{LExp}$. Also, $\mathsf{G}' = \mathsf{G}_1 \circ \mathsf{G}_2$. Given the assumption of the safety of $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle$, it is the case that $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle \rightsquigarrow^* (s', \mathsf{G}_1)$. Thus the property holds.

For $\mathsf{C} = \mathsf{L} := \mathsf{LExp}$, $s' \overset{\mathsf{L}}{\sim} s$, $s'(\mathsf{L}) = \mathsf{LExp}$, $\mathsf{G}' = \mathsf{G}_1 \circ \mathsf{G}_2$. Given the assumption of the safety of $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle$, it is the case that $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle \rightsquigarrow^* (s', \mathsf{G}_1)$. The result follows.

For $\mathsf{C} = x.\mathsf{data} := d$, $s(x) \in \mathrm{V}$, $s' \overset{x}{\sim} s$ and $\mathsf{G}' = (\mathrm{V}', \mathrm{E})$ such that if $s(x) = \langle u, d' \rangle$ then $s'(x) = \langle u, d \rangle$ and $\mathrm{V}' = (\mathrm{V} \backslash \{s(x)\}) \cup \{s'(x)\}$. Assuming

that $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle$ is safe, it follows that $s(x) \in \mathrm{V}_1$, and $\langle \mathsf{C}, s, \mathsf{G}_1 \rangle \rightsquigarrow^* (s', \mathsf{G}_1')$, where $\mathsf{G}_1' = (\mathrm{V}_1', \mathrm{E}_1)$ and $\mathrm{V}_1' = (\mathrm{V}_1 \backslash \{s(x)\}) \cup \{s'(x)\}$. The result follows.

Lemma 3 (Split) $\mathsf{region}(d, \mathsf{S}, \mathsf{T}) \models \mathsf{region}(d, \mathsf{S}_1, \mathsf{T}_1) * \mathsf{region}(d, \mathsf{S}_2, \mathsf{T}_2)$, where $\mathsf{S}_1 = \mathsf{S} \mathbin{\backslash\!\backslash} \{x\}$, $\mathsf{S}_2 = \mathsf{S} \Cap \{x\}$, $\mathsf{T}_i = \mathsf{T}\|_{\mathsf{S}_i}$, $i = 1, 2$.

Proof. Take s, G such that $s, \mathsf{G} \models \mathsf{region}(d, \mathsf{S}, \mathsf{T})$. We want to prove that $s, \mathsf{G}_1 \models \mathsf{region}(d, \mathsf{S}_1, \mathsf{T}_1)$ and $s, \mathsf{G}_2 \models \mathsf{region}(d, \mathsf{S}_2, \mathsf{T}_2)$, where $\mathsf{G}_1 \circ \mathsf{G}_2 = \mathsf{G}$.

First we show that $\mathrm{V} = \mathrm{V}_1 \cup \mathrm{V}_2$. Take an element $\langle u, d \rangle \in \mathrm{V}$. We want to prove that $\langle u, d \rangle \in \mathrm{V}_1 \cup \mathrm{V}_2$. By definition, $\langle u, d \rangle \in \mathrm{V}$ if and only if there exists $y \in \mathsf{S}$ such that $s(y) = \langle u, d \rangle$. Also, $\mathrm{V}_1 = s(\mathsf{S} \backslash \{x\}) = \{s(y) \mid y \in \mathsf{S}\} \mathbin{\backslash\!\backslash} \{s(x)\}$ and $\mathrm{V}_2 = s(\mathsf{S} \Cap \{x\}) = \{s(y) \mid y \in \mathsf{S}\} \cap \{s(x)\}$. If $s(y) = s(x)$ then $\langle u, d \rangle \in \mathrm{V}_2$; otherwise, if $s(y) \neq s(x)$ then $\langle u, d \rangle \in \mathrm{V}_1$.

Now we prove that $\overline{\mathrm{V}_1} \cap \overline{\mathrm{V}_2} = \varnothing$. Observe that from the previous paragraph we can extract that $\overline{\mathrm{V}_1}, \overline{\mathrm{V}_2} \subseteq \overline{\mathrm{V}}$. Also, note that for $x, y \in \mathsf{S}$, if $s(x) = \langle u, d \rangle$ and $s(y) = \langle u, d' \rangle$ then $d = d'$ by the definition of pregraph with data. Take an element $\langle u, d \rangle \in \mathrm{V}_1 = s(\mathsf{S}_1)$. There is no element $\langle u, d' \rangle \in \mathrm{V}$ with $d' \neq d$, so we can already conclude that $\langle u, d' \rangle \notin \mathrm{V}_2$. From our premise, it follows that $\langle u, d \rangle \in \{s(y) \mid y \in \mathsf{S}\} \backslash \{s(x)\}$. Equivalently, $\langle u, d \rangle \notin \{s(y) \mid y \in \mathsf{S}\} \cap \{s(x)\}$, thus $\langle u, d \rangle \notin s(\mathsf{S}_2) = \mathrm{V}_2$. Therefore $\overline{\mathrm{V}_1} \cap \overline{\mathrm{V}_2} = \varnothing$.

We conclude the proof showing that $\mathrm{E} = \mathrm{E}_1 \cup \mathrm{E}_2$. Take $(u, v) \in \mathrm{E} = s(\mathsf{T})$. Assume that $(u, v) \notin \mathrm{E}_1 = s(\mathsf{T}_1)$. It follows that $(u, v) \notin \{(\overline{s}(y), \overline{s}(z)) \mid s(y) \in s(\mathsf{S}_1) \text{ or } s(z) \in s(\mathsf{S}_1)\}$. Thus $(u, v) \neq (\overline{s}(y), \overline{s}(z))$ for all pairs such that either $s(y) \in s(\mathsf{S})$ and $s(y) \neq s(x)$ or $s(z) \in s(\mathsf{S})$ and $s(z) \neq s(x)$. If $(u, v) = (\overline{s}(y), \overline{s}(z))$ and $s(y) \in s(\mathsf{S})$, then $s(y) = s(x)$ and $(u, v) \in s(\mathsf{T}_2) = \mathrm{E}_2$. On the other hand if $s(y) \notin s(\mathsf{S})$, then by the definition of pregraph, $s(z) \in s(\mathsf{S})$. Therefore, $s(z) = s(x)$ and $(u, v) \in s(\mathsf{T}_2) = \mathrm{E}_2$. Thus $(u, v) \in \mathrm{E}_1 \cup \mathrm{E}_2$. The proof that $(u, v) \notin \mathrm{E}_2$ implies $(u, v) \in \mathrm{E}_1$ follows an analogous approach. Thus $\mathrm{E} = \mathrm{E}_1 \cup \mathrm{E}_2$

Lemma 4 (Merge) $\mathsf{region}(d, \mathsf{S}_1, \mathsf{T}_1) * \mathsf{region}(d, \mathsf{S}_2, \mathsf{T}_2) \models \mathsf{region}(d, \mathsf{S}_1 \Cup \mathsf{S}_2, \mathsf{T}_1 \Cup \mathsf{T}_2)$.

Proof. Let s, G be such that $s, \mathsf{G} \models \mathsf{region}(d, \mathsf{S}_1, \mathsf{T}_1) * \mathsf{region}(d, \mathsf{S}_2, \mathsf{T}_2)$. This implies that $\mathrm{V} = s(\mathsf{S}_1) \cup s(\mathsf{S}_2) = s(\mathsf{S}_1 \Cup \mathsf{S}_2)$ and $\mathrm{E} = s(\mathsf{T}_1) \cup s(\mathsf{T}_2) = s(\mathsf{T}_1 \Cup \mathsf{T}_2)$. The result immediately follows.