Expressiveness and Complexity of Graph Logic

Anuj Dawar[1], Philippa Gardner[2], and Giorgio Ghelli[3]

[1] University of Cambridge Computer Lab., Cambridge, UK, anuj.dawar@cl.cam.ac.uk
[2] Department of Computing, Imperial College, London, UK, pg@doc.ic.ac.uk
[3] Dipartimento di Informatica, Università di Pisa, Pisa, Italy, ghelli@di.unipi.it

Abstract. We investigate the complexity and expressive power of the spatial logic for querying graphs introduced by Cardelli, Gardner and Ghelli (ICALP 2002). We show that the model-checking complexity of versions of this logic with and without recursion is PSPACE-complete. In terms of expressive power, the version without recursion is a fragment of the monadic second-order logic of graphs and we show that it can express complete problems at every level of the polynomial hierarchy. We also show that it can define all regular languages, when interpretation is restricted to strings. The expressive power of the logic with recursion is much greater as it can express properties that are PSPACE-complete and therefore unlikely to be definable in second-order logic.

1 Introduction

Spatial logics are logics to describe graphs, memory heaps, or trees, characterized by the presence of the composition (or separation) operator. These logics allow one to reason compositionally about mobile processes [6] and about code with side effects [15]. Recently, languages to query tree-shaped and graph-shaped databases have been proposed, based on spatial logics featuring first-order quantification, composition, and recursive formulas [5, 4]. The languages seem to be pragmatically viable [8], but a precise assessment of their expressive power and evaluation cost is needed to understand whether spatial logics are a good foundation for semi-structured data query languages. With this aim, we are concerned in particular with determining, for a logic L:

- the combined complexity of the model-checking problem for L, i.e. the complexity class that contains the set $\text{Truths}_L = \{(G, \sigma, \phi) \ : \ \phi \in L,\ G \vDash^{\sigma} \phi\}$;
- the *expressivity* of the query language associated with L, i.e. the set of all sets $G_\phi = \{G \ : \ G \vDash \phi\}$, for a sentence $\phi \in L$; expressivity is also known as "data complexity" of L.

We focus on GL_μ, a spatial logic to describe graphs introduced in [4], which exhibits the crucial operators in the simplest setting and on its recursion-free fragment GL. Our main result is that, although model-checking for GL_μ is PSPACE-complete as it is for FO (first-order) and MSO (monadic second-order), GL_μ can define sets of graphs which are PSPACE-complete and therefore unlikely to be expressible in second-order logic.

We establish that the set Truths_{GL} is in PSPACE in Section 2.1, by a translation into MSO. In Section 3 we show that Truths_{GL} is PSPACE-hard and we show that, for any $\phi \in GL$, G_ϕ is in the polynomial hierarchy (PH) and that for each level Σ_i^p of PH, there is a $\phi \in GL$ such that G_ϕ is Σ_i^p-complete. Finally, in Section 4 we examine the expressiveness and complexity of the logic with recursion.

1.1 Preliminaries

The formulas of Graph Logic are interpreted over *labelled, directed* graphs. While in [4] these graphs are originally described by terms in a suitable algebra, an alternative presentation as relational structures is also provided. The approach we take here is based on the latter. To be precise, a graph is a structure $G = (X \cup E \cup A, \mathsf{edge})$ where,

– X is a set of vertices, E a set of edges and A a set of labels. These sets are all finite and mutually disjoint. Moreover, $X \subseteq \mathcal{X}$ and $A \subseteq \mathcal{A}$, where $\mathcal{X}$ is a fixed infinite set of *names* and $\mathcal{A}$ a fixed infinite set of labels (*names* do not actually name anything, they are just the universe of constants from which elements of the graph are drawn).
– $\mathsf{edge} : E \to A \times X \times X$ is a function that associates with each edge a label and a source and destination vertex.

This presentation differs from the one given in [4] in that we do not have an explicit function src mapping names to vertices in the graph. Essentially this is because we do not consider the operation of name hiding that plays a significant role in the original presentation of the term algebra of graphs. However, all of our results easily apply to the case where hiding is available. It is only the translation of Graph Logic to monadic second-order logic presented in Section 2.1 that requires some modification to cope with hiding, as we will point out.

The essential graph building operation is graph composition. Intuitively, the composition $G_1 \mid G_2$ is formed by taking the *disjoint* union of the graphs G_1 and G_2, while identifying nodes which have the same name. More formally, suppose $G_1 = (X_1 \cup E_1 \cup A_1, \mathsf{edge}_1)$ and $G_2 = (X_2 \cup E_2 \cup A_2, \mathsf{edge}_2)$. Then, the composition $G = G_1 \mid G_2$ is defined to be the graph $(X \cup E \cup A, \mathsf{edge})$ where:

– $X = X_1 \cup X_2$;
– $E = E_1 \uplus E_2$;
– $A = A_1 \cup A_2$; and
– $\mathsf{edge} = \mathsf{edge}_1 \uplus \mathsf{edge}_2$.

Definition 1. *The formulas of (recursion-free) Graph Logic (GL) are built up from a set $\mathcal{X}$ of node names, a set $\mathcal{A}$ of label names, a set $V_\mathcal{X}$ of node variables and a set $V_\mathcal{A}$ of label variables. A formula is one of*

$\mathbf{0}$	
T	
$\alpha(\xi_1, \xi_2)$	*where* $\alpha \in \mathcal{A} \cup V_\mathcal{A}$ *and* $\xi_i \in \mathcal{X} \cup V_\mathcal{X}$
$\xi_1 = \xi_2$ *or* $\alpha_1 = \alpha_2$	*where* $\alpha_i \in \mathcal{A} \cup V_\mathcal{A}$ *and* $\xi_i \in \mathcal{X} \cup V_\mathcal{X}$
$\phi \mid \psi$ *or* $\phi \wedge \psi$ *or* $\neg\phi$	*where* ϕ *and* ψ *are formulas*
$\exists x.\phi$ *or* $\exists a.\phi$	*where* $x \in V_\mathcal{X}, a \in V_\mathcal{A}$ *and* ϕ *is a formula.*

In the following, we use bold face for constants ($\mathbf{a}$, $\mathbf{x}$, ...), italics for variables (a, x, ...) and greek letters for terms (α, ξ...), which are either constants or variables.

For the semantics, suppose ϕ is a formula and $G = (X \cup E \cup A, \mathsf{edge})$ is a graph. Let $\sigma : V_\mathcal{X} \cup V_\mathcal{A} \to \mathcal{X} \cup \mathcal{A}$ be an assignment of node and label names to the node and label variables respectively. Extend σ in a canonical fashion to the domain $V_\mathcal{X} \cup V_\mathcal{A} \cup \mathcal{X} \cup \mathcal{A}$, by letting $\sigma(\mathbf{z}) = \mathbf{z}$ for $\mathbf{z} \in \mathcal{X} \cup \mathcal{A}$. The satisfaction relation $G \vDash^\sigma \phi$ is defined by:

1. $G \vDash^\sigma \mathsf{T}$ for any G.
2. $G \vDash^\sigma \mathbf{0}$ if, and only if, $E = \emptyset$.
3. $G \vDash^\sigma \xi_1 = \xi_2$ if, and only if, $\sigma\xi_1 = \sigma\xi_2$.
4. $G \vDash^\sigma \alpha_1 = \alpha_2$ if, and only if, $\sigma\alpha_1 = \sigma\alpha_2$.
5. $G \vDash^\sigma \alpha(\xi_1, \xi_2)$ if, and only if, $E = \{\mathbf{e}\}$ and $\mathsf{edge}(\mathbf{e}) = (\sigma\alpha, \sigma\xi_1, \sigma\xi_2)$.
6. $G \vDash^\sigma \phi \mid \psi$ if, and only if, $G = G_1 \mid G_2$ for some G_1 and G_2 and $G_1 \vDash^\sigma \phi$ and $G_2 \vDash^\sigma \psi$.
7. $G \vDash^\sigma \exists x.\phi$ if, and only if, $G \vDash^{\sigma'} \phi$ for some σ' which agrees with σ for all arguments other than x.
8. $G \vDash^\sigma \exists a.\phi$ if, and only if, $G \vDash^{\sigma'} \phi$ for some σ' which agrees with σ for all arguments other than a.

The semantics for the Boolean connectives is standard. Note that the quantifiers are interpreted as ranging over the infinite sets of names $\mathcal{X}$ and $\mathcal{A}$, even though the graph G is finite. This makes no difference to the power of the logic to express properties that are invariant under renamings, as we shall see.

1.2 Related Work

A review of the vast field of work on the complexity and expressivity of various logics is beyond the scope of this paper. We only cite papers that study these questions in the context of the so-called spatial logics, which are logics characterized by the presence of a "separation" (or *composition*) operator like the $|$ operator introduced above.

Calcagno et al. studied the complexity of model-checking and validity for the separation logic, a spatial logic used to describe mutable data structures stored in a heap [2]. They prove undecidability of validity for the logic with quantifiers, and then focus on the quantifier-free fragment, which we do not consider here. Their most interesting result is decidability of validity for the quantifier-free fragment in presence of the adjunct operator "magic wand", operator which we do not consider here. The same result is proved for the quantifier-free Static Ambient Logic in [3].

Charatonik et al. studied the complexity of model-checking for the ambient logic [7]. This is a spatial logic for the properties of processes, characterized by spatial and temporal modalities, which are not of interest in the context of static graphs. They prove that model-checking is PSPACE-complete for the simplest fragments of the logic, but it remains so even when the model is enriched with communication and mobility, and the logic is enriched with spatial and temporal modalities.

Dal-Zilio et al. defined a translation of the quantifier-free spatial ambient logic enriched with Kleene star operator into Presburger automata [10]. Once more, they focus on operators we do not consider (adjunct and Kleene star), but they only study the quantifier-free fragment of the logic.

2 Graph Logic and MSO

We consider the expressive power of GL and compare it to the monadic second-order logic of graphs. Section 2.1 shows that formulas of GL can be translated into this logic (with a minor qualification with regard to the range of quantification), while Section 2.2 illustrates the expressivity of GL by looking at the definitions of some specific properties of graphs. In Section 2.3 we show that the expressive power of GL coincides with that of monadic second-order logic over graphs that are strings.

2.1 Monadic Second-Order Logic

In this section we show that for every formula of GL, there is an equivalent formula of monadic second-order logic (MSO). As the structures over which we interpret our formulas are three-sorted, so is our version of MSO. Moreover, we treat edge syntactically as a 4-ary relation, rather than a function. To be precise, the formulas of MSO are built up from the names $\mathcal{X}$, the labels $\mathcal{A}$, three sorts of first-order variables: $V_{\mathcal{X}}$, $V_{\mathcal{A}}$ and $V_{\mathcal{E}}$, and a collection $\mathcal{S}$ of set variables. A formula is one of:

$\mathsf{edge}(e, \alpha, \xi_1, \xi_2)$	where $\alpha \in \mathcal{A} \cup V_{\mathcal{A}}$, $e \in V_{\mathcal{E}}$ and $\xi_i \in \mathcal{X} \cup V_{\mathcal{X}}$
$e_1 = e_2$ or $\alpha_1 = \alpha_2$ or $\xi_1 = \xi_2$	where $e_i \in V_{\mathcal{E}}, \alpha_i \in \mathcal{A} \cup V_{\mathcal{A}}, \xi_i \in \mathcal{X} \cup V_{\mathcal{X}}$
$e \in S$	where $e \in V_{\mathcal{E}}$ and $S \in \mathcal{S}$
$\phi \wedge \psi$ or $\neg\phi$	where ϕ and ψ are formulas
$\exists e.\phi$ or $\exists x.\phi$ or $\exists a.\phi$	where $x \in V_{\mathcal{X}}, e \in V_{\mathcal{E}}, a \in V_{\mathcal{A}}$ and ϕ is a formula
$\exists S.\phi$	where $S \in \mathcal{S}$ and ϕ is a formula.

The semantics is standard except that set variables are interpreted as ranging over *sets of edges*. In this sense, this version of monadic second-order logic is akin to the logic MS_2 of [9].

In order to define a translation of formulas of GL to MSO, there is one issue that needs to be addressed carefully. The node and label quantifiers in GL are interpreted over the infinite sets $\mathcal{X}$ and $\mathcal{A}$. However, in the standard semantics of MSO, the range of the first-order quantifiers is the set of nodes

X and the set of labels A that are *in* the graph. In order to define a reasonable translation, we show that the quantifiers in GL can be restricted to a finite set, without loss of generality.

Let $G = (X \cup E \cup A, \mathsf{edge})$ be a graph and ϕ a formula of GL which contains k distinct node variables and l distinct label variables. Let $X' \subset \mathcal{X}$ be a finite set of node names such that: $X \subset X'$; each node constant occurring in ϕ is in X'; and X' contains at least k distinct names that are not in X and do not occur in ϕ. Similarly, let $A' \subset \mathcal{A}$ be a finite set containing A, all label constants occurring in ϕ and at least l other labels.

Lemma 1. *If $G \vDash^{\sigma} \phi$ then there is a $\sigma' : V_{\mathcal{X}} \cup V_{\mathcal{A}} \to X' \cup A'$ such that $G \vDash^{\sigma'} \phi$.*

Proof. Let $\mathbf{x}$ be a name that is in the range of σ but not in X'. If $\mathbf{x} = \sigma x$ for some variable that occurs free in ϕ, by the choice of X', there must be some $\mathbf{y} \in X'$ such that $\mathbf{y} \neq \sigma y$ for any y that is free in ϕ and such that $\mathbf{y}$ is not in the graph G and does not occur in ϕ. Thus, the permutation of $\mathcal{X}$ that only exchanges $\mathbf{x}$ and $\mathbf{y}$ is an automorphism of G and therefore $G \vDash^{\sigma, x \mapsto \mathbf{y}} \phi$. Proceeding in this way with all variables that are mapped to elements outside X', and similarly with label variables that are mapped to Y', we obtain the desired σ'.

Now, for a sentence ϕ, write $G \vDash^{X', A'} \phi$ if G satisfies ϕ when the interpretation of the node and label quantifiers is restricted to the sets X' and A' respectively. We use Lemma 1 to establish the following.

Lemma 2. *For any sentence ϕ of GL, $G \vDash \phi$ if, and only if, $G \vDash^{X', A'} \phi$.*

Proof. Follows easily by applying Lemma 1 to each subformula of ϕ that begins with an existential quantifier.

To define the translation of formulas of GL, we define, inductively, for each formula ϕ of GL a formula $[\![\phi]\!]^S$ of MSO, which is to be read as "the translation of ϕ relativized to the set of edges S". In the following, we use $S = S' \mid S''$ as an abbreviation for the MSO formula:

$$\forall e.\ ((e \in S \ \Leftrightarrow\ (e \in S' \vee e \in S'')) \wedge (e \in S' \ \Leftrightarrow\ e \notin S''))$$

$$
\begin{array}{lll}
[\![\alpha(\xi, \xi')]\!]^S & =_{def} \exists e \in S.\ [\mathsf{edge}(e, \alpha, \xi, \xi') \wedge \forall e' \in S.\ e = e'] & \\
[\![\phi' \mid \phi'']\!]^S & =_{def} \exists S', S''.\ S = S' \mid S'' \wedge [\![\phi']\!]^{S'} \wedge [\![\phi'']\!]^{S''} & \\
[\![\mathbf{0}]\!]^S & =_{def} \neg\exists e.\ e \in S & \\
[\![\neg\phi]\!]^S & =_{def} \neg[\![\phi]\!]^S & [\![\phi \wedge \phi']\!]^S =_{def} [\![\phi]\!]^S \wedge [\![\phi']\!]^S \\
[\![\exists x.\phi]\!]^S & =_{def} \exists x.\ [\![\phi]\!]^S & [\![\exists a.\phi]\!]^S =_{def} \exists a.\ [\![\phi]\!]^S \\
[\![\xi = \xi']\!]^S & =_{def} \xi = \xi' & [\![\alpha = \alpha']\!]^S =_{def} \alpha = \alpha'
\end{array}
$$

To state precisely in what sense this is a valid translation, we need one further definition. For any graph $G = (X \cup E \cup A, \mathsf{edge})$ and any formula $\phi \in GL$ with k distinct node variables and l distinct label variables, define the graph $G^+ = (X' \cup E \cup A', \mathsf{edge})$ where X' is X along with all node constants occurring in ϕ and exactly k additional nodes, and A' is A along with all label constants occurring in ϕ and exactly k additional labels. Then, we have:

Theorem 1. *For any graph $G = (X \cup E \cup A, \mathsf{edge})$, any formula $\phi \in GL$ and any σ whose range is contained in $X' \cup A'$*

$$G \vDash^{\sigma} \phi \ \Leftrightarrow\ G^+ \vDash^{\sigma} [\![\phi]\!]^E.$$

Proof. Straightforward induction on the formula.

Remark: As we remarked in Section 1, in defining the logic GL, [4] considers an algebra for graph descriptions that does not involve name hiding and it is this version of the logic that the above translation corresponds to. If name hiding is allowed, the operation of graph composition is subtly different. That is, if G_1 and G_2 are graphs, their composition $G_1 \mid G_2$ is defined by taking the disjoint union of the sets of edges and identifying nodes with the same name *provided that the name is not hidden*. Formulas of GL can still be translated to MSO under this interpretation if we include in our graph structures a unary predicate Named for the set of nodes whose names are not hidden.

The effect of Theorem 1 is that it immediately establishes upper bounds on the expressive power of GL and the complexity of the model-checking problem. It is known by [13, 17] that a property of relational structures is definable in second-order logic if, and only if, it is in the polynomial hierarchy PH. Thus, we have:

Theorem 2. *For any sentence $\phi \in GL$, the set of graphs $\{G \,:\, G \models \phi\}$ belongs to* PH.

Moreover, for each level Σ_i^p of the polynomial hierarchy, there are properties definable in MSO that are Σ_i^p-complete. We show in Section 3 that this is also the case for GL. On the other hand, there are computationally simple properties that are not expressible in MSO which therefore could not be expressible in GL either. For instance, the following is a direct consequence of Theorem 1:

Corollary 1. *There is no formula ϕ of GL such that $G \models \phi$ if, and only if, G has an even number of edges.*

It is also known that the combined complexity of MSO is PSPACE-complete. That is, the decision problem $\text{Truths}_{\text{MSO}} = \{(G, \sigma, \phi) \,:\, \phi \in \text{MSO},\ G \models^{\sigma} \phi\}$ is PSPACE-complete. Since the translation given above is itself computable in polynomial time, it establishes that Truths_{GL} is in PSPACE. In Section 3 we show that it is, in fact, PSPACE-complete.

2.2 Expressivity of Graph Logic

We have seen that, in terms of expressive power, GL can be considered to be a fragment of monadic second-order logic. Translation in the other direction is not possible for trivial reasons having to do with the range of quantification. For instance, the sentence $\exists x \forall y \forall a \forall e(\neg \mathsf{edge}(e, a, x, y) \wedge \neg \mathsf{edge}(e, a, y, x)$ of MSO is true in a graph G if, and only if, it contains isolated nodes. In GL, with quantifiers ranging over the infinite set of all names, it is not possible to distinguish isolated nodes in the graph from nodes not in the graph. Thus, for the purpose of a translation, it makes sense to restrict ourselves to graphs that contain no isolated nodes. Still, we conjecture that there is no translation possible. That is, there are formulas of MSO that are not equivalent, even in restriction to the class of finite graphs with no isolated nodes, to any formula of GL. Establishing this is difficult. To illustrate this difficulty, we show how we can express some natural MSO properties in GL. Below we describe informally how these properties are expressed by formulas of GL. The details of the definitions are in the full paper.

Connectivity. Graph connectivity is a problem well-known to be definable in MSO, but not in first-order logic (see [12]). We note that it is definable in GL.

The following formulas constrain the in- and out-degrees of nodes:

$$\begin{array}{llll} \text{In}_{\geq 1}(x) & =_{def} \exists a, y.\ a(y,x) \mid \mathsf{T} & \text{Out}_{\geq 1}(x) & =_{def} \exists a, y.\ a(x,y) \mid \mathsf{T} \\ \text{In}_{\geq 2}(x) & =_{def} \text{In}_{\geq 1}(x) \mid \text{In}_{\geq 1}(x) & \text{Out}_{\geq 2}(x) & =_{def} \text{Out}_{\geq 1}(x) \mid \text{Out}_{\geq 1}(x) \\ \text{In}_{0}(x) & =_{def} \neg \text{In}_{\geq 1}(x) & \text{Out}_{0}(x) & =_{def} \neg \text{Out}_{\geq 1}(x) \\ \text{In}_{1}(x) & =_{def} \text{In}_{\geq 1}(x) \wedge \neg \text{In}_{\geq 2}(x) & \text{Out}_{1}(x) & =_{def} \text{Out}_{\geq 1}(x) \wedge \neg \text{Out}_{\geq 2}(x) \end{array} \tag{1}$$

We will also use the formula Here, defined by $\text{Here}(x) =_{def} \text{In}_{\geq 1}(x) \vee \text{Out}_{\geq 1}(x)$. This formula selects those nodes that are actually in the graph. We henceforth use the abbreviation $\forall x \in G.\ \phi$ for $\forall x.\ \text{Here}(x) \Rightarrow \phi$, and similarly $\forall x_1, \ldots, x_n \in G.\ \phi$.

Now, define the formula

$$\text{Path}(x,y) =_{def} \text{In}_0(x) \wedge \text{Out}_0(y) \wedge \forall z \in G((z \neq x \Rightarrow \text{In}_1(z)) \wedge (z \neq y \Rightarrow \text{Out}_1(y))). \quad (2)$$

If, for a graph G, $G \vDash^{\sigma} \text{Path}(x,y)$, then G consists of a single path from σx to σy along with possibly other simple cycles disjoint from this path. Thus, $G \vDash^{\sigma} (\text{Path}(x,y) \mid \mathsf{T})$ holds if, and only if, G contains a path from σx to σy. Indeed, if the formula is satisfied, then $G = G_1 \mid G_2$, where G_1 consists of a path and possibly some cycles, implying that G contains the required path. Conversely, if G contains a path from σx to σy, it can be expressed as the composition of this path and the rest of the graph, thereby satisfying the formula. Thus, we have that the formula $\forall x, y \in G.(\text{Path}(x,y) \mid \mathsf{T})$ expresses that a graph is strongly connected.

Disjoint Paths. Using the formula Path defined above, t is easy to construct a formula that expresses the property that there are two *edge-disjoint* paths between distinguished nodes $\mathbf{a}$ to $\mathbf{b}$ in a graph G. The formula is $\text{Path}(\mathbf{a},\mathbf{b}) \mid \text{Path}(\mathbf{a},\mathbf{b}) \mid \mathsf{T}$. We show how to construct a formula that expresses that there are two *node-disjoint* paths from $\mathbf{a}$ to $\mathbf{b}$, which is a well-known NP-complete problem [14]. The formula $\text{TwoPath}(\mathbf{a},\mathbf{b})$

$$\begin{aligned}\text{TwoPath}(\mathbf{a},\mathbf{b}) =_{def}\ & \text{Out}_{\geq 2}(\mathbf{a}) \wedge \text{In}_0(\mathbf{a}) \wedge \text{In}_{\geq 2}(\mathbf{b}) \wedge \text{Out}_0(\mathbf{b}) \\ & \wedge \forall x \in G((x \neq \mathbf{a} \wedge x \neq \mathbf{b} \Rightarrow \text{In}_1(x)) \wedge (x \neq \mathbf{a} \wedge x \neq \mathbf{b} \Rightarrow \text{Out}_1(x)))\end{aligned}$$

is true in a graph G if, and only if, G consists of two node-disjoint paths from $\mathbf{a}$ to $\mathbf{b}$ and possibly some additional simple cycles. Thus, $\text{TwoPath}(\mathbf{a},\mathbf{b}) \mid \mathsf{T}$ expresses the existence of two node-disjoint paths from $\mathbf{a}$ to $\mathbf{b}$.

2-colourability. As a final example, we construct a formula that expresses that a graph is 2-colourable. Similarly to the formulas $\text{In}_{\geq 2}$ and $\text{Out}_{\geq 2}$, we define the formula $\text{Deg}_{\geq n}(x)$ to be the formula that asserts that x has at least n neighbours (regardless of the direction of the edges). Thus, for instance

$$\text{Deg}_{\geq 2}(x) =_{def} (\exists a, y.\ (a(y,x) \vee a(x,y))) \mid (\exists a, y.\ (a(y,x) \vee a(x,y))) \mid \mathsf{T}.$$

Also, define $\text{Deg}_{=n}(x)$ to be the formula $\text{Deg}_{\geq n}(x) \wedge \neg\text{Deg}_{\geq n+1}(x)$. We also define the formulas

$$\begin{aligned}\text{Cycles} &=_{def} \forall x \in G.\ \text{Deg}_{=2}(x) \\ \text{Edges} &=_{def} \forall x \in G.\ \text{Deg}_{=1}(x).\end{aligned}$$

Note that, if every node in a graph G has exactly two neighbours, then G is the disjoint union of simple cycles (ignoring directions on the edges). Similarly, if every node in G has exactly one neighbour, than G is the disjoint union of simple edges. Thus, the sentences Cycles and Edges express, respectively, that G is a disjoint collection of cycles and that G is a disjoint collection of edges. Also observe that a collection of simple cycles can be decomposed into two graphs each of which is a disjoint collection of edges if, and only if, all the cycles are of even length. Thus, the sentence $\text{Cycles} \wedge \neg(\text{Edges} \mid \text{Edges})$ is satisfied by G if, and only if, G is a collection of cycles at least one of which is of odd length. Since a graph is 2-colourable if, and only if, it contains no cycles of odd length, 2-colourability is defined by the sentence: $\neg[(\text{Cycles} \wedge \neg(\text{Edges} \mid \text{Edges})) \mid \mathsf{T}]$.

These examples illustrate the varied expressive power of GL. Nonetheless, it seems unlikely that GL is as expressive as MSO. We conjecture that *Hamiltonicity*—the class of graphs that contain a Hamiltonian cycle; and *3-colourability* are both inexpressible in GL. These can both be seen to be expressible in MSO[4].

[4] That is, in the version of MSO we have defined, which allows quantification over sets of edges. If second-order quantification is restricted to sets of vertices, then Hamiltonicity is not definable [12, Cor. 6.3.5], though 3-colourability still is.

In order to separate the expressive power of MSO from that of GL, we need a method that can demonstrate that a given property is not definable in the latter. A natural such class of techniques is the Ehrenfeucht-Fraïssé style games. Games for spatial logics such as GL are introduced in [11]. It is not difficult to use such a game to show that GL cannot express that a graph has an even number of edges, yielding a direct proof of Corollary 1. It remains a challenge to deploy this game method to prove the inexpressibility of more complex problems, such as Hamiltonicity or 3-colourability.

2.3 Strings

We now look at the expressive power of GL on a particular class of graphs corresponding to strings over a finite alphabet. A *string graph* over the finite alphabet $A \subset \mathcal{A}$ is either the empty graph or a graph of the shape $\mathbf{a}_1(\mathbf{x}_1, \mathbf{x}_2) \mid \ldots \mid \mathbf{a}_{n-1}(\mathbf{x}_{n-1}, \mathbf{x}_n)$, where each of the labels $\mathbf{a}_i$ is in A. Identifying the graph $\mathbf{a}_1(\mathbf{x}_1, \mathbf{x}_2) \mid \ldots \mid \mathbf{a}_{n-1}(\mathbf{x}_{n-1}, \mathbf{x}_n)$ with the word $\mathbf{a}_1 \ldots \mathbf{a}_{n-1}$, we can see each sentence ϕ of GL as defining a language—namely the set of words that satisfy ϕ.

While the relationship of GL with MSO over arbitrary graphs remains unresolved we are able to show that on string graphs they have equal expressive power. It is well-known (by a result of Büchi [1]) that a language is definable in MSO if, and only if, it is regular[5]. It follows that any language definable in GL is regular. We now show the converse, that every regular language is definable by a formula of GL. We do this by translating regular expressions into formulae of GL. The crucial case is the Kleene star. The translation to GL is based on the observation that a string graph is in the language L^* if, and only if, it can be decomposed into two graphs, each of which is the disconnected sum of strings in L.

In order to do this, we first introduce some auxiliary formulae to count the incoming and outgoing edges. In addition to the formulae introduced at (1), (3) and (2), we need:

$$\begin{aligned} \mathrm{In}_{\leq 1}(x) &=_{def} \mathrm{In}_0(x) \vee \mathrm{In}_1(x) \\ \mathrm{Out}_{\leq 1}(x) &=_{def} \mathrm{Out}_0(x) \vee \mathrm{Out}_1(x) \end{aligned}$$

We can now introduce some additional predicates. NoCycles means that no subgraph of the current graph is a cycle. LinearFromTo(x, y) means that the graph is linear from x to y. Finally, SetOfWords(ϕ) means that the graph is a set of disconnected linear graphs, and each of these linear graphs satisfies the formula ϕ.

$$\begin{aligned} \mathrm{NotEmptySetOfCycles} &=_{def} (\neg \mathbf{0}) \wedge \forall x \in G.\ (\mathrm{In}_1(x) \wedge \mathrm{Out}_1(x)) \\ \mathrm{NoCycles} &=_{def} \neg(\mathsf{T} \mid \mathrm{NotEmptySetOfCycles}) \\ \mathrm{LinearFromTo}(x, y) &=_{def} \mathrm{NoCycles} \wedge \mathrm{Path}(x, y) \\ \mathrm{Linear} &=_{def} \exists x, y.\ \mathrm{LinearFromTo}(x, y) \\ \mathrm{SetOfWords}(\phi) &=_{def} \mathrm{NoCycles} \wedge (\forall x \in G.\ \mathrm{In}_{\leq 1}(x) \wedge \mathrm{Out}_{\leq 1}(x)) \\ &\quad \wedge (\forall x, y \in G.\ (\mathrm{In}_0(x) \wedge \mathrm{Out}_0(y) \\ &\qquad \Rightarrow\ (\mathsf{T} \mid \Rightarrow \mathrm{LinearFromTo}(x, y)\ \Rightarrow\ \phi))) \end{aligned}$$

We can now translate every regular expression L into an equivalent formula $\mathsf{F}(L)$. The critical case is Kleene star. A word w belongs to L^* if it is the concatenation of a sequence of words $w_1, \ldots, w_n$, each belonging to L. In this case, it is possible to split the corresponding graph $G(w)$ into the two

[5] The original result uses a different representation of strings as relational structures, but the proof is robust and easily translated to our setting.

subgraphs G_1 and G_2 which are not themselves words, but each is a set of disconnected words, each of them individually in L. On the other hand, if it is possible to split $G(w)$ into two such graphs, then $w \in L^*$. This is expressed by the formula $\mathsf{F}(L^*)$ below. Observe that each of the two graphs G_1 and G_2 may be empty.

$$\begin{aligned}
\mathsf{F}(\epsilon) &=_{def} \mathbf{0} \\
\mathsf{F}(L;L') &=_{def} \text{Linear} \wedge \exists x, y.\ (\mathsf{F}(L) \wedge \text{Out}_0(x)) \mid (\text{In}_0(x) \wedge \mathsf{F}(L')) \\
\mathsf{F}(L+L') &=_{def} \mathsf{F}(L) \vee \mathsf{F}(L') \\
\mathsf{F}(L^*) &=_{def} \mathbf{0} \vee \text{Linear} \wedge (\text{SetOfWords}(\mathsf{F}(L)) \mid \text{SetOfWords}(\mathsf{F}(L)))
\end{aligned}$$

This gives us

Theorem 3. *A language is definable in GL if, and only if, it is regular.*

3 Complexity of Graph Logic

In this section we establish lower bounds on the data complexity and combined complexity of GL that match the upper bounds established through the translation to MSO.

3.1 Combined complexity

We begin by showing that the set $\text{Truths}_{GL} = \{(G, \sigma, \phi) \ :\ \phi \in GL,\ G \vDash^{\sigma} \phi\}$ is PSPACE-hard, by a reduction from validity of Quantified Boolean Formulas (QBF). We use a fixed graph in the reduction. The same technique has been used in [7] to prove the same result for the ambient logic, and in [2] to prove undecidability of validity for the separation logic. Indeed, the proof is based on the PSPACE-completeness of the first-order logic of graphs. In the reduction, we do not use the operator $\mid$.

A Quantified Boolean Formula is a term generated by the following grammar:

$$\Phi \ ::=\ \exists P.\ \Phi \ \mid\ \neg\Phi \ \mid\ \Phi \wedge \Phi' \ \mid\ P$$

where P ranges over propositional variables. The definitions of validity and satisfiability for QBF are standard (see [16]). QBF formulas Φ are translated into GL formulas as follows, where for each P, $x_P \in V_{\mathcal{X}}$ is a different name variable, and $\mathbf{t} \in \mathcal{X}$ is a name constant; the translation can be evaluated in Logspace.

$$\begin{aligned}
[\![\exists P.\ \Phi]\!] &=_{def} \exists x_P.\ [\![\Phi]\!] & [\![P]\!] &=_{def} x_P = \mathbf{t} \\
[\![\neg\Phi]\!] &=_{def} \neg[\![\Phi]\!] & [\![\Phi \wedge \Phi']\!] &=_{def} [\![\Phi]\!] \wedge [\![\Phi']\!]
\end{aligned}$$

Lemma 3. *Satisfiability of closed QBF formulas can be reduced to the model-checking problem $\{(G, \sigma, \phi) \ :\ \phi \in GL,\ G \vDash^{\sigma} \phi\}$ for a fixed graph G.*

Proof. We take G to be the fixed graph that satisfies $\mathbf{a}(\mathbf{t}, \mathbf{f})$. Given an assignment σ of truth values to the propositional variables in a formula Φ, define σ' to be the variable assignment such that $\sigma(X) = \text{true} \ \Leftrightarrow\ \sigma'(x_X) = \mathbf{t}$. An easy induction then shows that Φ is true under σ if, and only if, $G \vDash^{\sigma'} [\![\Phi]\!]$, hence the closed formula Φ is satisfiable if, and only if, $(G, \epsilon, [\![\Phi]\!]) \in \text{Truths}_{GL}$.

Theorem 4. *The model-checking problem for GL is PSPACE-complete.*

The expressive power of GL is intermediate between that of first-order logic (FO) and MSO, in that any first-order formula on graphs can be translated to GL. Thus, Theorem 4 is unsurprising as the model-checking problem for both FO and MSO is PSPACE-complete.

3.2 Data complexity

We prove here that, for each positive integer k there exists a sentence ϕ_k such that the problem: 'given G, decide whether $G \models \phi_k$' is hard for the k^{th} universal level of the polynomial hierarchy, i.e. $G_{\phi_k} \in \Pi_i^p$-hard. Since GL is closed under negation, we also obtain hard problems for the existential levels of the hierarchy. In the proof we use the derived operator $\phi \mid\Rightarrow \psi =_{def} \neg(\phi \mid \neg\psi)$. By this definition, $G \models^\sigma \phi \mid\Rightarrow \psi$ if, and only if, for every partition of G into G', G'', if $G' \models^\sigma \phi$, then $G'' \models^\sigma \psi$. We will also use the following formulas:

$$\begin{array}{ll} \mathrm{In}_{\geq 2}(x) & =_{def} \exists a, y, a', y'.\ a(y,x) \mid a'(y',x) \mid \mathsf{T} \\ \mathrm{In}_{\geq 1}(x) & =_{def} \exists a, y.\ a(y,x) \mid \mathsf{T} \\ \mathrm{In}_0(x) & =_{def} \neg \mathrm{In}_{\geq 1}(x) \\ \mathrm{Out}_{\geq 1}(x) & =_{def} \exists a, y.\ a(x,y) \mid \mathsf{T} \end{array} \tag{3}$$

The proof begins by showing a logspace-computable function $G(\Phi)$, that codes propositional formulas as graphs in a way that allows us to express questions about the satisfiability and the validity of Φ as GL formulas over $G(\Phi)$.

Informally, G translates a propositional formula Φ into a graph $G(\Phi)$ of the same size, that represents Φ as a circuit. The edges of the graph have labels from the set *And*, *Not*, *Var*, *FanOut*, *Switch* and *Result*. There is an edge labelled *Var* for each variable, the source of which is the target of an edge labelled *Switch* (the purpose of these edges is that a selection of them will encode an assignment of truth values to the variables). The target of each *Var* edge is the source of a number of edges labelled *FanOut*, one for each occurrence of the variable in Φ. The source of each *And* edge has two incoming edges and its target has one outgoing edge. Similarly, the source of each *Not* edge has one incoming edge and its target one outgoing edge. Finally, there is only one edge labelled *Result* , that marks the root of the formula, and its source and target are the same.

More formally, we define the translation G as follows. In the first line $X_1, \ldots, X_n$ are the variables of Φ. The translation is based on a quadruple $x = (x_{si}, x_{so}, x_v, x_p)$; x_{si}, x_{so}, x_v are three functions that associate nodes to the free variables of Φ, and x_p associates a node to each occurrence in Φ and to the special occurrence ϵ^-. As usual, an occurrence is a string of 0's and 1's, possibly empty (ϵ), such that, when α is associated to a formula rooted in a binary operator, $\alpha.0$ and $\alpha.1$ are associated to its two subformulas, and similarly for unary operators. The operator $(\alpha)^-$ is defined as $(\alpha.0)^- = (\alpha.1)^- = \alpha$, $(\epsilon)^- = \epsilon^-$. The four functions in x are all injective and their codomains are disjoint. These functions do not use up space, since they are not stored in a table, but perform some fixed bit-manipulation on the input to produce the output.

$$\begin{array}{ll} G(\Phi, x) & =_{def} [\![\Phi, \epsilon]\!]^x \mid \mathit{Switch}(x_{si}(X_1), x_{so}(X_1)) \mid \mathit{Var}(x_{so}(X_1), x_v(X_1)) \mid \\ & \quad \ldots \mid \mathit{Switch}(x_{si}(X_n), x_{so}(X_n)) \mid \mathit{Var}(x_{so}(X_n), x_v(X_n)) \\ & \quad \mid \mathit{Result}(x_p(\epsilon^-), x_p(\epsilon^-)) \\ [\![\Phi \wedge \Phi', \alpha]\!]^x & =_{def} [\![\Phi, \alpha.0]\!]^x \mid [\![\Phi', \alpha.1]\!]^x \mid \mathit{And}(x_p(\alpha), x_p(\alpha^-)) \\ [\![\neg\Phi, \alpha]\!]^x & =_{def} [\![\Phi, \alpha.0]\!]^x \mid \mathit{Not}(x_p(\alpha), x_p(\alpha^-)) \\ [\![X_i, \alpha]\!]^x & =_{def} \mathit{FanOut}(x_v(X_i), x_p(\alpha^-)) \end{array}$$

Let $(\Phi)_\alpha$ be the subformula of Φ rooted at the occurrence α. Observe that for every occurrence α in Φ we have exactly one edge $\mathbf{a}(\mathbf{y}, x_p(\alpha^-))$, labelled And, Not, or FanOut, depending on whether the root of $(\Phi)_\alpha$ is $\wedge$, $\neg$, or a variable, unless $\alpha = \epsilon$, in which case we also have an edge $\mathit{Result}(x_p(\epsilon^-), x_p(\epsilon^-))$. Also observe that:

– Switch edges have no incoming edge and one outgoing edge;

- Var edges have one $Switch$ incoming edge and many $FanOut$ outgoing edges;
- $FanOut$ edges have one Var incoming edge and one And or Not outgoing edge;
- And edges have two $FanOut$, And, or Not incoming edge and one And, Not, or $Result$ outgoing edge;
- Not edges have one $FanOut$, And, or Not incoming edge and one And, Not, or $Result$ outgoing edge;
- the $Result$ edge has itself plus one $FanOut$, And, or Not as the only incoming edges, and itself as the only outgoing edge.

The previous enumeration is exhaustive: no edge has any incoming or outgoing edges more than what is listed above.

Corresponding to any assignment σ of truth values to the variables in Φ, there is a decomposition of $G(\Phi)$ into two graphs G' and G'' where G' contains the edges corresponding to subformulas of Φ that are made true by σ. We construct a pair of formulas of GL $TruePart$ and $FalsePart$ that define the well-formedness of the graphs G' and G''. For instance, $TruePart$ asserts that a graph contains an *And* edge if, and only if, it contains both its preceding edges and $FalsePart$ asserts that a graph contains an *And* edge if, and only if, it contains at least one of its preceding edges. A propositional formula is satisfiable if there exists an assignment whose partition leaves the root edge $Result$ in the $TruePart$ (see $GSat$ below). The propositional formula is valid if, for any assignment, hence for any possible $FalsePart$, the formula root is in the $TruePart$ ($GVal$). The formulas $TruePart$, and $FalsePart$, and the partition $G_t(\Phi, x, \sigma)$, $G_f(\Phi, x, \sigma)$ are defined as follows.

$$
\begin{aligned}
TruePart \quad &= \forall x, y.(Switch(x,y) \mid \mathsf{T}) \;\Rightarrow\; \mathrm{Out}_{\geq 1}(y) \\
&\quad \wedge \forall x, y.\ (Var(x,y) \mid \mathsf{T}) \;\Rightarrow\; \mathrm{In}_{\geq 1}(x) \\
&\quad \wedge \forall x, y.\ (FanOut(x,y) \mid \mathsf{T}) \;\Rightarrow\; \mathrm{In}_{\geq 1}(x) \\
&\quad \wedge \forall x, y.\ (And(x,y) \mid \mathsf{T}) \;\Rightarrow\; \mathrm{In}_{\geq 2}(x) \\
&\quad \wedge \forall x, y.\ (Not(x,y) \mid \mathsf{T}) \;\Rightarrow\; \mathrm{In}_{0}(x) \\
&\quad \wedge \forall x\ (Result(x,x) \mid \mathsf{T}) \;\Rightarrow\; \mathrm{In}_{\geq 2}(x) \\
FalsePart \quad &= \forall x, y.(Switch(x,y) \mid \mathsf{T}) \;\Rightarrow\; \mathrm{Out}_{\geq 1}(y) \\
&\quad \wedge \forall x, y.\ (Var(x,y) \mid \mathsf{T}) \;\Rightarrow\; \mathrm{In}_{\geq 1}(x) \\
&\quad \wedge \forall x, y.\ (FanOut(x,y) \mid \mathsf{T}) \;\Rightarrow\; \mathrm{In}_{\geq 1}(x) \\
&\quad \wedge \forall x, y.\ (And(x,y) \mid \mathsf{T}) \;\Rightarrow\; \mathrm{In}_{\geq 1}(x) \\
&\quad \wedge \forall x, y.\ (Not(x,y) \mid \mathsf{T}) \;\Rightarrow\; \mathrm{In}_{0}(x) \\
&\quad \wedge \forall x\ (Result(x,x) \mid \mathsf{T}) \;\Rightarrow\; \mathrm{In}_{\geq 2}(x) \\
G_t(\Phi, x, \sigma) \quad &= \{\mathbf{a}(\mathbf{y}, x_p(\alpha^-)) \;:\; \text{the subformula } (\Phi)_\alpha \text{ is true in } \sigma\} \\
&\quad \cup \{Var(x_{so}(X), x_v(X)) \;:\; \sigma(X) = \text{true}\} \\
&\quad \cup \{Switch(x_{si}(X), x_{so}(X)) \;:\; \sigma(X) = \text{true}\} \\
G_f(\Phi, x, \sigma) \quad &= \{\mathbf{a}(\mathbf{y}, x_p(\alpha^-)) \;:\; \text{the subformula } (\Phi)_\alpha \text{ is false in } \sigma\} \\
&\quad \cup \{Var(x_{so}(X), x_v(X)) \;:\; \sigma(X) = \text{false}\} \\
&\quad \cup \{Switch(x_{si}(X), x_{so}(X)) \;:\; \sigma(X) = \text{false}\}
\end{aligned}
$$

Lemma 4. *For every assignment σ to the free variables of Φ:*

$$G_t(\Phi, x, \sigma) \vDash TruePart \;\wedge\; G_f(\Phi, x, \sigma) \vDash FalsePart$$

For every partition of $G(\Phi, x)$ into G_t, G_f, and σ such that $\sigma(X) = false \;\Leftrightarrow\; Switch(x_{si}(X), x_{so}(X)) \in G_f$:

$$
\begin{aligned}
&G_t \vDash TruePart \;\wedge\; G_f \vDash FalsePart \\
&\Rightarrow\; G_t = G_t(\Phi, x, \sigma) \;\wedge\; G_f = G_f(\Phi, x, \sigma)
\end{aligned}
$$

Proof. The first part is true by construction. For the second part, by construction. all the switches of true variables are in G_t; by $\forall x, y.(Switch(x, y) \mid \mathsf{T}) \Rightarrow \text{Out}_{\geq 1}(y)$, all the variables associated to these switches are in G_t, and by $\forall x, y.\ (Var(x, y) \mid \mathsf{T}) \Rightarrow \text{In}_{\geq 1}(x)$ only these variables are. Hence, all the false variables are in G_f.

Now we can prove that, for every occurrence α of Φ, an edge $\mathbf{a}(\mathbf{y}, x_p(\alpha^-))$ is in G_t when $(\Phi)_\alpha =$ true and is in G_f otherwise. We prove it by induction on the well-founded order on the occurrences of Φ defined by the transitive closure of $\alpha < \alpha^-$; the order is well-founded because Φ is finite (observe that the minimal elements are the longest occurrences in Φ).

When α is minimal, then $(\Phi)_\alpha$ is a variable, hence $\mathbf{a}(\mathbf{y}, x_p(\alpha^-))$ is a *FanOut* edge, which is in $\{\mathbf{a}(\mathbf{y}, x_p(\alpha^-)) : (\Phi)_\alpha = \text{true}\}$ if, and only if, its input variable is true. The condition $\forall x, y.\ (FanOut(x, y) \mid \mathsf{T}) \Rightarrow \text{In}_{\geq 1}(x)$ ensures that every *FanOut* edge of a true variable is in G_t. The same reasoning holds exchanging *true*, G_t, with *false*, G_f.

When α is not minimal, then $(\Phi)_\alpha$ is rooted either in $\wedge$ or in $\neg$; we first assume $\alpha \neq \epsilon$.

In the first case, $\mathbf{a}(\mathbf{y}, x_p(\alpha^-)) = And(x_p(\alpha), x_p(\alpha^-))$, and, if it is in G_t, by $\forall x, y.\ (And(x, y) \mid \mathsf{T}) \Rightarrow \text{In}_{\geq 2}(x)$ both of its incoming edges are in G_t too, else, by induction, both $(\Phi)_{\alpha.0}$ and $(\Phi)_{\alpha.1}$ are true, hence $(\Phi)_\alpha$ is true too. The same reasoning holds for G_f, with the difference that in this case we are only assured that one of the two incoming edges is in G_f, but this is enough to ensure that $(\Phi)_\alpha$ is false.

If $(\Phi)_\alpha$ is rooted in $\neg$, then $\mathbf{a}(\mathbf{y}, x_p(\alpha^-)) = Not(x_p(\alpha), x_p(\alpha^-))$, and, if it is in G_t, $\forall x, y.\ (Not(x, y) \mid \mathsf{T}) \Rightarrow \text{In}_0(x)$ implies that its incoming edge is in G_f, hence, by induction, $(\Phi)_{\alpha.0}$ is false, hence $(\Phi)_\alpha$ is true. The same reasoning holds exchanging *true*, G_t, with *false*, G_f.

Finally, if $\alpha = \epsilon$, $\mathbf{a}(\mathbf{y}, x_p(\alpha^-))$ may also be $Result(\epsilon^-, \epsilon^-)$; if it is in G_t, then the other $\mathbf{a}(\mathbf{y}, x_p(\epsilon^-))$ is in G_t too, hence the whole subformula is true, hence $Result(\epsilon^-, \epsilon^-)$ is in $G_t(\Phi, x, \sigma)$. The same reasoning holds exchanging *true*, G_t, with *false*, G_f.

Conversely, we can show that, if $G(\Phi, x) = G_f \mid G_t$ and $G_f \vDash FalsePart$, then there is a truth assignment σ such that $G_f = G_f(\Phi, x, \sigma)$ and $G_t = G_t(\Phi, x, \sigma)$. This truth assinment is obtained by setting $\sigma(X)$ true for all X for which the edge $Switch(x_{si}(X), x_{so}(X))$ is in G_t.

We can now define the formulas $GSat$ and $GVal$ that characterise the sets of graphs encoding satisfiable and the valid formulas respectively.

$$
\begin{aligned}
ResultHere &=_{def} \exists x.\ Result(x, x) \mid \mathsf{T} \\
GSat &=_{def} (TruePart \wedge ResultHere) \mid FalsePart \\
GVal &=_{def} FalsePart \mid\!\Rightarrow (TruePart \wedge ResultHere)
\end{aligned}
$$

Theorem 5. *The set $\{G : G \vDash GVal\}$ is co-NP-complete.*

Proof. We reduce PVAL, the validity of propositional formulas, to $\{G : G \vDash GVal\}$, by proving that $\Phi \in \text{PVAL} \Leftrightarrow G(\Phi, x) \vDash GVal$.

Assume that $\Phi \in$ PVAL. We want to prove that $G(\Phi, x) \vDash GVal$, i.e. that for every partition $G(\Phi, x)$ as $G_f \mid G_t$, if $G_f \vDash FalsePart$, then $G_t \vDash TruePart \wedge ResultHere$.

Let σ be the truth assignment associated with the partition G_f, G_t. Thus,$G_t = G_t(\Phi, x, \sigma)$ and $G_f = G_f(\Phi, x, \sigma)$. Since $G_f \vDash FalsePart$ and Φ is valid, $\mathbf{a}(\mathbf{y}, x_p(\epsilon^-))$ is in G_t, hence, by definition of $G_t(\Phi, x, \sigma)$ the $Result$ loop is in G_t, hence $G_t \vDash ResultHere$.

For the other direction, assume that $G(\Phi, x) \vDash GVal$. This implies that, for every partition $G(\Phi, x)$ as $G_f \mid G_t$, if $G_f \vDash FalsePart$, then $G_t \vDash TruePart \wedge ResultHere$. We can now conclude that for every σ the set of all occurrences α such that $(\Phi)_\alpha$ is true includes the root ϵ, hence the formula is valid.

Theorem 6. *The set $\{G : G \vDash GSat\}$ is NP-complete.*

Proof. We reduce PSAT, the satisfiability of propositional formulas, to $\{G : G \vDash GSat\}$, by reasoning as in the previous case.

We are now ready to prove that, for each k there exists a formula ϕ_k such that the problem: 'given G, decide whether $G \models^{\sigma} \phi_k$' is Π_k^p-hard. The standard hard problem for a class Π_k^p of the polynomial hierarchy is the validity of a Quantified Boolean Formula with k alternation of quantifiers, i.e. a formula like the following one (where no quantifier appears in Φ and the last quantifier is $\exists$ if k is even, as we assume below, and is $\forall$ if k is odd):

$$\forall X_1^1 \ldots X_{i_1}^1 . \exists Y_1^2 \ldots Y_{i_2}^2 . \ldots \forall X_1^{k-1} \ldots X_{i_{k-1}}^{k-1} . \exists Y_1^k \ldots Y_{i_k}^k . \Phi.$$

We encode such formulas into graphs which has edge labels *And*, *Not*, *Var*, *FanOut*, and *Result* as before. In addition, there is, for each $1 \leq i \leq k$, a label *SwitchE*^{i} (or *SwitchA*^{i} if the i^{th} quantifier block is universal). With variables thus labelled as universally or existentially quantified and marked with an index indicating where they occur in the formula, we are able to construct a formula $GVal_k$, using alternations of $\mid$ and negation, to express that the formula encoded by $G(\Phi)$ is valid.

Theorem 7. *For each k there exists a GL formula ϕ_k (ψ_k) that characterizes a set of graphs that is complete for Π_k^p (Σ_k^p).*

Proof. We only show this for Π_k^p and for even k. As for the previous theorems, for every k, we exhibit a translation $G_k(\Phi, x)$ and a formula $GVal_k$ such that, for every closed QBF with maximal level k, Φ_k is valid iff $G_k(\Phi, x) \models GVal_k$.

Consider a formula

$$\forall X_1^1 \ldots X_{i_1}^1 . \exists Y_1^2 \ldots Y_{i_2}^2 . \ldots \exists Y_1^k \ldots Y_{i_k}^k . \Phi.$$

We consider the following translation $G_k(\Phi, x)$, identical to the previous one but for the partition of the switches according to the level and the quantifier of their variables.

$$\begin{aligned} G_k(\Phi, x) \quad =_{def} \; & [\![\Phi, \epsilon]\!]^x \mid SwitchA^1(x_{si}(X_1^1), x_{so}(X_1^1)) \mid Var(x_{so}(X_1^1), x_v(X_1^1)) \\ & \mid SwitchA^1(x_{si}(X_{i_1}^1), x_{so}(X_{i_1}^1)) \mid Var(x_{so}(X_{i_1}^1), x_v(X_{i_1}^1)) \\ & \mid SwitchE^2(x_{si}(Y_1^2), x_{so}(Y_1^2)) \mid Var(x_{so}(Y_1^2), x_v(Y_1^2)) \\ & \mid SwitchE^2(x_{si}(Y_{i_2}^2), x_{so}(Y_{i_2}^2)) \mid Var(x_{so}(Y_{i_2}^2), x_v(Y_{i_2}^2)) \\ & \ldots \\ & \mid SwitchE^k(x_{si}(Y_{i_k}^k), x_{so}(Y_{i_k}^k)) \mid Var(x_{so}(Y_{i_k}^k), x_v(Y_{i_k}^k)) \\ & \mid Result(x_p(\epsilon^-), x_p(\epsilon^-)) \\ [\![\Phi \wedge \Phi', \alpha]\!]^x =_{def} \; & \ldots \end{aligned}$$

The formula $GVal_k$ is defined as follows.

$$\begin{aligned} & SwitchesA^1 \mid\Rightarrow (SwitchesE^2 \mid \ldots \\ & \quad (SwitchesA^{k-1} \mid\Rightarrow (SwitchesE^k \mid ((TruePart_i \wedge ResultHere) \mid FalsePart_i))) \ldots) \end{aligned}$$

Where $SwitchesA^i$ and $SwitchesE^i$ are defined as:

$$\begin{aligned} SwitchesA^i &=_{def} \forall a, x, y. \; (a(x, y) \mid \mathsf{T}) \;\Rightarrow\; a = SwitchA^i \\ SwitchesE^i &=_{def} \forall a, x, y. \; (a(x, y) \mid \mathsf{T}) \;\Rightarrow\; a = SwitchE^i \end{aligned}$$

$TruePart_k$ is the same as $TruePart$ but we replace $Switch(x, y)$ in the implication

$$\forall x, y. (Switch(x, y) \mid \mathsf{T}) \;\Rightarrow\; \text{Out}_{\geq 1}(y)$$

with a disjoint listing of the different switch edges we have:

$$\forall x, y. ((SwitchA^1(x, y) \vee \ldots \vee SwitchE^k(x, y)) \mid \mathsf{T}) \;\Rightarrow\; \text{Out}_{\geq 1}(y).$$

$FalsePart_k$ is defined by generalizing $FalsePart$ in the same way.

The formula $GVal_k$ says that: for each assignment to first-level universal variables there exists an assignment to second-level existential variables such that ... for each assignment to $k-1^{th}$-level universal variables there exists an assignment to k^{th}-level existential variables such that the formula holds.

One consequence of this is that the alternation of $|$ with negation forms an infinite hierarchy of expressive power in GL. Hence, it is not possible to obtain a normal form similar to the conjunctive or disjunctive normal forms of boolean operators, characterized by a fixed number of alternations between $|$ and negation.

Corollary 2. *Unless the polynomial hierarchy collapses, the alternation of* $|$ *and negation form a strict hierarchy in* GL*.*

4 Recursion

In [4], a fixed-point operator $\mu R.\phi$ is defined for GL, where R is a variable that can appear inside ϕ as an atomic formula, and only in positive positions, i.e. each occurrence of R is within the scope of an even number of negations in ϕ. We use GL_μ to denote the logic with recursion.

Satisfaction $G \vDash^\sigma \phi$ is now defined as the minimal relation that satisfies the properties specified in Section 1 plus the following:

$$G \vDash^\sigma \mu R.\phi \text{ if, and only if, } G \vDash^\sigma \phi\{R \leftarrow (\mu R.\phi)\}$$

More formally, we can define the semantics such that $[\![\phi]\!]_{\sigma;\rho}$ denotes the set of all graphs that satisfy ϕ under assignments σ and ρ, where ρ is a valuation for the recursion variables.

In the definition below, we follow the notation of [4] for describing graphs. Thus, $\mathbf{a}(\mathbf{x},\mathbf{y})$ is a term denoting the graph with a single edge labelled $\mathbf{a}$ between the nodes $\mathbf{x}$ and $\mathbf{y}$. Larger graphs can be built up using the composition operator $|$. It should be clear from context whether such a term or a formula of GL_μ is intended.

Definition 2 (Satisfaction). *Let* $\mathcal{G}$ *be the set of all graphs. Let* σ *denote an assignment for name and label variables, and let* ρ *map recursion variables to elements of* $\mathcal{P}(\mathcal{G})$*. The* satisfaction interpretation

$[\![\_]\!]_{\sigma;\rho} : \mathcal{F} \to \mathcal{P}(\mathcal{G})$ *is defined as:*

$$
\begin{aligned}
[\![\mathbf{0}]\!]_{\sigma;\rho} &= \{G \ : \ G \equiv \mathbf{0}\} \\
[\![\alpha(\xi_1, \xi_2)]\!]_{\sigma;\rho} &= \{G \ : \ G \equiv \alpha\sigma(\xi_1\sigma, \xi_2\sigma)\} \\
[\![\phi \mid \psi]\!]_{\sigma;\rho} &= \{G \ : \ G \equiv G_1 \mid G_2 \ \wedge \ G_1 \in [\![\phi]\!]_{\sigma;\rho} \ \wedge \ G_2 \in [\![\psi]\!]_{\sigma;\rho}\} \\
[\![\mathsf{T}]\!]_{\sigma;\rho} &= \mathcal{G} \\
[\![\phi \ \wedge \ \psi]\!]_{\sigma;\rho} &= [\![\phi]\!]_{\sigma;\rho} \cap [\![\psi]\!]_{\sigma;\rho} \\
[\![\neg\phi]\!]_{\sigma;\rho} &= \mathcal{G} \setminus [\![\phi]\!]_{\sigma;\rho} \\
[\![\exists x. \ \phi]\!]_{\sigma;\rho} &= \bigcup_{\mathbf{x} \in \mathcal{X}} [\![\phi]\!]_{\sigma, x \mapsto \mathbf{x};\rho} \\
[\![\exists a. \ \phi]\!]_{\sigma;\rho} &= \bigcup_{\mathbf{a} \in \mathcal{A}} [\![\phi]\!]_{\sigma, a \mapsto \mathbf{a};\rho} \\
[\![R]\!]_{\sigma;\rho} &= R\rho \\
[\![\mu R.\phi]\!]_{\sigma;\rho} &= \bigcap \{S \in \mathcal{P}(\mathcal{G})) \ : \ [\![\phi]\!]_{\sigma;\rho, R \mapsto S} \subseteq S\} \\
[\![\xi_1 = \xi_2]\!]_{\sigma;\rho} &= \mathcal{G}, \ \textit{if } \xi_1\sigma = \xi_2\sigma; \ \emptyset \ \textit{otherwise} \\
[\![\alpha_1 = \alpha_2]\!]_{\sigma;\rho} &= \mathcal{G}, \ \ \textit{if } \alpha_1\sigma = \alpha_2\sigma; \ \emptyset \ \textit{otherwise}
\end{aligned}
$$

Definition 2 is shown to be well-defined by structural induction on formulas. For the recursive case, observe that the set $\mathcal{P}(\mathcal{G})$ is a complete lattice. Define the *satisfaction relation* $G \vDash^{\sigma} \phi$ for a formula ϕ with no free recursion variables if and only if $G \in [\![\phi]\!]_{\sigma;\rho}$, for any ρ.

It is instructive to compare this logic with other logics of recursion, such as LFP, the extension of first-order logic with an operator for forming the least fixed points of relational expressions (see [12] for an exposition). For a relational variable R and a formula ϕ in which R only appears positively, LFP allows the expression $\mu R.\phi$ which defines the least relation such that $R \leftrightarrow \phi$. Just as the graph composition operator of GL can be simulated by a monadic second-order quantifier one might think that recursion can be simulated by the fixed-point operator of LFP. There is, however, a crucial difference. While the fixed-points in LFP are defined in the lattice of relations on a graph G, the form recursion takes here defines fixed-points of maps on the lattice of *sets of subgraphs* of G. Thus, while the evaluation of a fixed-point in LFP can be performed by an iteration that is guaranteed to converge in a polynomial number of steps, the number of steps required to reach a fixed-point of a recursive formula in GL is potentially exponential in the size of the graph G. This is amply illustrated by the result in Section 4.2 that exhibits a PSPACE-complete problem that is definable in the logic. This is why the result in Section 4.1 showing that the model-checking complexity of the logic is still in PSPACE is remarkable.

We now explore the complexity and the expressivity of this recursion operator.

4.1 Combined Complexity

The set $\{(G, \sigma, \phi) \ : \ \phi \in GL_{\mu}, \ G \vDash^{\sigma} \phi\}$ is PSPACE-hard, by results in Section 3. We now exhibit a PSPACE algorithm to decide the problem, establishing a tight upper bound on its complexity.

The algorithm extends the standard model-checking algorithm for SO, and is listed in Table 1.

We assume that no variable in the formula is bound in two distinct places. We associate a counter to each variable and a bitmask to each $\mid$ operator in the formula. We have a variable bitmap *mask* that specifies which edges in the graph are included in the current subgraph. To check whether $\phi \mid \psi$ holds, we let the corresponding bitmask *m* iterate over all the submasks of the current mask. For each value

inputs: *G*, ψ; other global variables: *stack*, and the implicit call stack;

Evaluate(ψ,G) =
    let mask= $1_1 1_2 \ldots 1_n$ where n = sizeof(G);
    return(eval(ψ, mask));

eval($\neg\phi$, mask) =
    return(not eval(ϕ,mask));

eval($\phi \wedge \psi$, mask) =
    if (eval(ϕ, mask) andif eval(ψ, mask)) {return(true);}
    else return (false);

eval($\exists x.\phi$, mask) =
    for i in 1..n do
        push($\langle$"x" $= i\rangle$, stack);
        if eval(ϕ, mask) {pop(stack); return(true);}
        pop(stack)
    return(false);

eval($\phi \mid \psi$, mask) =
    for submask in submasks(mask) do
        if (eval(ϕ, submask) andif eval(ψ, compl(submask))) {return(true);}
    return (false);

eval($a(x, y)$, mask) =
    if (get(stack,"a")(get(stack,"x"),get(stack,"y")) $\in$ $G \cap$mask) {return(true);}
    else return(false);

eval($\mu R.\psi$, mask) =
    push($\langle$"R" $= mask\rangle$, stack);
    res = eval(ψ, mask);
    pop(stack); return(res);

eval(R, mask)
    if mask== get(stack,"R") return(false)
    else find $\mu R.\phi$ in the input formula ψ
          and return(eval($\mu R.\phi$, mask));

Table 1. The model-checking algorithm for GL_μ

of *m* we check ϕ against the subgraph identified by *m* and ψ against its complement. $\phi \mid \psi$ holds if and only if we find a value for *m* such that both checks succeed. Actually, we cannot have a different variable for each $\mid$, since the algorithm is written to work for formulas of any size. Hence, *m* is just a local variable of the procedure that checks the $\mid$ case, which will be automatically saved on the call stack when a subroutine is called, and restored when the subroutine exits. Thus, we have, on the call stack, as many bitmask variables as we need.

To check whether $\exists x.\phi$ holds, we let the corresponding counter *x* enumerate all the names that appear either in the graph or in the formula, plus one fresh name for each variable in the formula. $\exists x.\phi$ holds if and only if we find a value for *x* such that ϕ model-checks. As above, we do not have a different counter for each variable *x*, but we use a stack. In this case we push a pair *"x"=x* on an explicit stack every time a quantification $\exists x$ is met. We do not use the call stack for *x* because later, to check whether $\alpha(\xi_1, \xi_2)$ holds, we will have to substitute all the variables among α, ξ_1, ξ_2 with their value, and we can retrieve those values by exploring the explicit stack.

To model-check $\mu R.\phi$, we first push on the stack a pair *R-current mask*. Then, when R is met, we first check whether the current mask is still equal to that associated to R on the stack. If the current mask is a strict subset of that stored in R, we substitute R for $\mu R.\phi$ and continue. If it is equal to that stored in R, then this branch can only loop forever, hence *false* is returned.

We prove that this algorithm runs in polynomial space and that it is correct. The proofs are given in the appendix.

Theorem 8. Evaluate(ψ, G) *always terminates and can be executed with polynomial space.*

Proof. Let n be the maximum of the number of edges and the number of nodes in G. The algorithm uses the call stack and the variable *stack*. Each recursive call pushes its local variables, the return address, and the call parameters on the call stack. The worst case is that of $\mid$, where we have one local variable that is n bits long (*submask*) plus the *mask* parameter that is n bits long as well. Hence, each stack frame on the call stack is linearly bounded by the input size. The same is true for the *stack* variable, where each stack frame has either n size (if it is a mask) or $\log(n)$ size, in the $\exists x/a.\phi$ cases. Moreover, each procedure call performs at most one push, and always pops what it pushed, hence the *stack* variable never contains more frames then the call stack. Hence we have only to show that the call stack growth is bounded by a polynomial. This bound implies termination as well, since all the for loops in the code are bounded.

Every frame in the call stack contains a bit mask. This mask is always equal to, or included into, the one of the preceding frame. Let a stack-chunk be a sequence of stack frames which all contain the same mask. The stack will always be composed by at most $n+1$ stack-chunks, where n is the size of the input graph, since $n+1$ is the length of the longest chain of n-bit masks ordered by strict inclusion. A single stack-chunk may contain two frames that correspond to the evaluation of the same R variable only if the second one is the last frame on the stack, since the first evaluation of R with mask m is followed by an evaluation of $\mu R.\psi$ which pushes $\langle \mathrm{R} = m \rangle$ on *stack*, so that the next evaluation of R with mask m returns immediately. Hence, any stack-chunk contains at most $k+1$ recursion-variable frames, if k is the number of recursive variables in ϕ. Finally, the sub-chunk included between two consecutive recursion-variable frames cannot contain more than l frames, where l is the longest path in the syntax tree of ϕ, since any other case but R walks one step down along ϕ. This gives an $O(nkl)$ bound on the number of frames of the call stack.

Theorem 9. Evaluate(ψ, G) = true *if, and only if,* $G \vDash \psi$.

Proof. Consider the proof system of Table 2. The formula $\underline{G}$ in rule $(\forall\ \mu)$ is the formula that contains one $\mathbf{a}(\mathbf{x}, \mathbf{y})$ literal for each edge in the graph separated by $\mid$, hence is only satisfied by G (up to isomorphism). $\mathcal{X}(G, \sigma, \phi)$ is a set that contains all the names in G, σ, ϕ, plus one fresh name. No other fresh name needs to be checked, since they would all give the same result.

We will prove soundness of the algorithm w.r.t. the proof system, and soundness of the proof system w.r.t. satisfaction, i.e.:

$\text{eval}_{\psi,G,()}(\psi, m) = \text{true}$	$\Rightarrow$	$G \cap m;\, () \vdash \psi$	soundness of eval, case true
$\text{eval}_{\psi,G,()}(\psi, m) = \text{false}$	$\Rightarrow$	$G \cap m;\, () \nvdash \psi$	soundness of eval, case false
$\phi\sigma$ is closed, $G; \sigma \vdash \phi$	$\Rightarrow$	$G \vDash \phi\sigma$	soundness of $\vdash$
$\phi\sigma$ is closed, $G; \sigma \nvdash \phi$	$\Rightarrow$	$G \vDash \neg\phi\sigma$	soundness of $\nvdash$

Here, $\text{eval}_{\psi,G,s}(\phi, \text{mask})$ is the result of calling eval(ϕ, mask) when input formula is ψ, input graph is G, and current stack is s. $G \cap m$ is the subset of G identified by the mask m.

Our proof relies on the following definitions and properties.

$\Sigma^{+}_{\phi;\psi}(s, R = m)$	$=$	$\Sigma^{+}_{\phi;\psi}(s)\{R \mapsto \mu R.\phi'\}$	if R is positive in $\phi;\psi$, $\mu R.\phi'$ is a subterm of ψ
$\Sigma^{+}_{\phi;\psi}(s, R = m)$	$=$	$\Sigma^{+}_{\phi;\psi}(s)\{R \mapsto ((\mu R.\phi') \wedge \neg(\underline{G \cap m}))\}$	if R is negative in $\phi;\psi$
$\Sigma^{-}_{\phi;\psi}(s, R = m)$	$=$	$\Sigma^{+}_{\neg\phi;\psi}(s, R = m)$	
$\Sigma^{+/-}_{\phi;\psi}(s, x = \mathbf{x})$	$=$	$\Sigma^{+/-}_{\phi;\psi}(s)\{x \mapsto \mathbf{x}\}$	
$\text{eval}_{\psi,G,s}(\phi, m) = \text{true}$	$\Rightarrow$	$G \cap m;\, \Sigma^{+}_{\phi;\psi}(s) \vdash \phi$	$(1a)$
$\text{eval}_{\psi,G,s}(\phi, m) = \text{false}$	$\Rightarrow$	$G \cap m;\, \Sigma^{-}_{\phi;\psi}(s) \nvdash \phi$	$(1b)$

The translation of a stack $s, R = m$ into a substitution $\Sigma^{+}_{\phi;\psi}(s, R = m)$ depends on where we are in the process of building the proof, since rules $(\vdash \mu)$ and $(\nvdash \mu)$ push two different values for R in the substitution, and the algorithm is looking for a proof of either $(\vdash \mu)$ or $(\nvdash \mu)$. The property $(1ab)$ expresses the fact that this variability can be captured by a polarity computation, expressed by the functions $\Sigma^{+/-}_{\phi;\psi}()$. Here "$R$ is negative in $\phi;\psi$" means (informally): substitute in ϕ every free recursive variable with its definition in ψ; iterate this process an arbitrary number of times; R is negative in $\phi;\psi$ if no positive occurrence of R will ever appear. Hence, for example, R is negative in $R';\mu R.\neg\mu R'.\neg(R \mid \neg R')$. (The positivity condition over R' implies that, if no positive occurrence of R appears after we expand R' once, it is not going to appear after two expansions.)

Property $(1ab)$ is easy to prove by induction on the depth of the call stack of the algorithm, and by cases. We show the only hard case, $\text{eval}_{\psi,G,s}(R, m) = \text{false}$.

$\text{eval}_{\psi,G,s}(R, m) = \text{false} \;\Rightarrow$	
either $R = m$ is in s, or $\text{eval}_{\psi,G,s}(\mu R.\phi, m) = \text{false}$	
in the second case, by induction	$G \cap m;\, \Sigma^{-}_{\mu R.\phi;\psi}(s) \nvdash \mu R.\phi$
in the first case	$G \cap m;\, \Sigma^{-}_{\mu R.\phi;\psi}(s) \vdash \underline{G \cap m}$
hence	$G \cap m;\, \Sigma^{-}_{\mu R.\phi;\psi}(s) \nvdash \neg\underline{G \cap m}$
in both cases, we can apply rule $(\nvdash \wedge)$:	$G \cap m;\, \Sigma^{-}_{\mu R.\phi;\psi}(s) \nvdash (\mu R.\phi) \wedge \neg\underline{G \cap m}$
R must be positive in $\mu R.\phi$, hence	
$R(\Sigma^{-}_{\mu R.\phi;\psi}(s)) = (\mu R.\phi) \wedge \neg\underline{G \cap m}$, hence:	$G \cap m;\, \Sigma^{-}_{\mu R.\phi;\psi}(s) \nvdash R(\Sigma^{-}_{\mu R.\phi;\psi}(s))$
by $(\nvdash R)$:	$G \cap m;\, \Sigma^{-}_{\mu R.\phi;\psi}(s) \nvdash R$
$\Sigma^{-}_{\phi;\psi}(s)$ only depends on the polarity of variables inside ϕ, which does not change	
if we expand a variable with its definition:	$G \cap m;\, \Sigma^{-}_{R;\psi}(s) \nvdash R$

Now we have to prove soundness of the proof system, by induction on the size of a proof, and by cases on the last rule applied. All cases are trivial but $(\nvdash \mu)$. We want to prove that, for any pair ϕ, σ, such that σ binds all free variables in ϕ but R,

$$G \vDash \neg(\phi(\sigma\{R \mapsto ((\mu R.\phi) \wedge \neg\underline{G})\})) \;\Rightarrow\; G \vDash \neg((\mu R.\phi)\sigma)$$

i.e.

$$G \notin [\![\phi(\sigma\{R \mapsto ((\mu R.\phi) \wedge \neg \underline{G})\})]\!] \;\Rightarrow\; G \notin [\![\mu R.\phi\sigma]\!]$$

We will exploit the following properties, proved in [4]:

$$\begin{array}{ll} [\![\phi\{R \leftarrow \psi\}]\!]_{\sigma;\rho} = [\![\phi]\!]_{\sigma;\rho\{R \mapsto [\![\psi]\!]_{\sigma;\rho}\}} & (2) \\ [\![\mu R.\phi]\!]_{\sigma;\rho} = \mathit{fixpoint}(\lambda S.\ [\![\phi]\!]_{\sigma;\rho\{R \mapsto S\}}) & \\ \text{the function } \lambda S.\ [\![\phi]\!]_{\sigma;\rho\{R \mapsto S\}} \text{ is monotone in } S & \end{array}$$

We can rewrite $[\![\phi(\sigma\{R \mapsto ((\mu R.\phi) \wedge \neg \underline{G})\})]\!]$ as follows, where $M = [\![\mu R.\phi\sigma]\!]$, and $F(\phi,\sigma)$ is defined as $\lambda S.\ [\![(\phi\sigma)]\!]_{();\{R \mapsto S\}}$

$$\begin{array}{ll} [\![\phi(\sigma\{R \mapsto ((\mu R.\phi) \wedge \neg \underline{G})\})]\!] & = \\ [\![(\phi\sigma)\{R \leftarrow ((\mu R.\phi\sigma) \wedge \neg \underline{G})\}]\!] & = \text{by (2)} \\ [\![(\phi\sigma)]\!]_{();\{R \mapsto [\![((\mu R.\phi\sigma) \wedge \neg \underline{G})]\!]\}} & = \\ [\![(\phi\sigma)]\!]_{();\{R \mapsto M \setminus G\}} & = \\ F(\phi,\sigma)(M \setminus G) & \end{array}$$

Now, M is a fixed point of the monotone function $F(\phi,\sigma)$, hence $F(\phi,\sigma)(M \setminus G) \subseteq F(\phi,\sigma)(M) \subseteq M$. $G \notin F(\phi,\sigma)(M \setminus G)$ implies that $F(\phi,\sigma)(M \setminus G) \subseteq (M \setminus G)$, hence, by definition, $M \subseteq F(\phi,\sigma)(M \setminus G) \subseteq (M \setminus G)$, hence $G \notin M$, c.v.d.

$$\frac{G;\sigma \vdash \phi \ \wedge\ G;\sigma \vdash \psi}{G;\sigma \vdash \phi \wedge \psi}\ (\vdash \wedge) \qquad \frac{G;\sigma \not\vdash \phi \ \vee\ G;\sigma \not\vdash \psi}{G;\sigma \not\vdash \phi \wedge \psi}\ (\not\vdash \wedge) \qquad \frac{G;\sigma \not\vdash \phi}{G;\sigma \vdash \neg\phi}\ (\vdash \neg) \qquad \frac{G;\sigma \vdash \phi}{G;\sigma \not\vdash \neg\phi}\ (\not\vdash \neg)$$

$$\frac{\exists \mathbf{x} \in \mathcal{X}(G,\sigma,\psi).\ G;\sigma\{x \mapsto \mathbf{x}\} \vdash \phi}{G;\sigma \vdash \exists x.\phi}\ (\vdash \exists) \qquad \frac{\forall \mathbf{x} \in \mathcal{X}(G,\sigma,\psi).\ G;\sigma\{x \mapsto \mathbf{x}\} \not\vdash \phi}{G;\sigma \not\vdash \exists x.\phi}\ (\not\vdash \exists)$$

$$\frac{G = \alpha\sigma(\xi\sigma,\xi'\sigma)}{G;\sigma \vdash \alpha(\xi,\xi')}\ (\vdash \alpha(\xi,\xi')) \qquad \frac{G \neq \sigma(\xi\sigma,\xi'\sigma)}{G;\sigma \not\vdash \alpha(\xi,\xi')}\ (\not\vdash \alpha(\xi,\xi')) \qquad \frac{\xi\sigma = \xi'\sigma}{G;\sigma \vdash \xi = \xi'}\ (\vdash \xi = \xi') \qquad \frac{\xi\sigma \neq \xi'\sigma}{G;\sigma \not\vdash \xi = \xi'}\ (\not\vdash \xi = \xi')$$

$$\frac{\exists G', G''.\ G = G'|G'' \ \wedge\ G';T \vdash \phi \wedge G'';T \vdash \psi}{G;\sigma \vdash \phi \mid \psi}\ (\vdash \mid) \qquad \frac{\forall G', G''.\ G = G'|G'' \ \Rightarrow\ G';T \not\vdash \phi \vee G'';T \not\vdash \psi}{G;\sigma \not\vdash \phi \mid \psi}\ (\not\vdash \mid)$$

$$\frac{G;\sigma \vdash R\sigma}{G;\sigma \vdash R}\ (\vdash R) \qquad \frac{G;\sigma \not\vdash R\sigma}{G;\sigma \not\vdash R}\ (\not\vdash R) \qquad \frac{G;(\sigma\{R \mapsto (\mu R.\phi)\}) \vdash \phi}{G;\sigma \vdash \mu R.\phi}\ (\vdash \mu) \qquad \frac{G;(\sigma\{R \mapsto ((\mu R.\phi) \wedge \neg \underline{G})\}) \not\vdash \phi}{G;\sigma \not\vdash \mu R.\phi}\ (\not\vdash \mu)$$

Table 2. Proof system for $G \vdash \phi$ and $G \not\vdash \phi$

4.2 Expressive Power

While recursion does not take the combined complexity out of PSPACE, it adds expressive power to the logic. As a simple example, here is a formula that characterises the graphs with an even number of edges, which is not expressible in either MSO or LFP.

$$\mu F.\ \mathbf{0} \ \vee\ (\ (\exists a, x, y.\ a(x, y)) \mid (\exists a, x, y.\ a(x, y)) \mid F\ )$$

In this section we show that we can express a PSPACE-complete problem in this language. This is achieved by an encoding of quantifier Boolean formulas (QBF) as graphs. The encoding is similar to the one in Section 3.2 except we do not have different edge labels for the different number of quantifier alternations. Instead, we have two labels, *Forall* and *Exists*, in addition to *Switch*, and the alternation of edges with these labels leading up to a *Var* edge indicate the quantifier type and index of the corresponding Boolean variable. The details are given in the proof of Theorem 10 in the appendix. They allow us to define a formula $GVal_\mu$ of GL_μ which defines those graphs that encode valid QBF formulas.

$$\begin{aligned}\mu R.\ &((\mathit{TruePart} \wedge \mathit{ResultHere}) \mid \mathit{FalsePart})\\ &\vee (\exists x, y.\ \mathit{QRoot}(x) \wedge (\mathit{Forall}(x, y) \mid (\mathit{SwitchesOf}(y) \mid\Rightarrow R)))\\ &\vee (\exists x, y.\ \mathit{QRoot}(x) \wedge (\mathit{Exist}(x, y) \mid (\mathit{SwitchesOf}(y) \mid R)))\end{aligned}$$

The formula can be read as: either (1) the formula has no quantifier and is valid, or (2) the outermost quantifier is a $\forall$, and the rest of the formula ($\mathit{Forall}(x, y) \mid \ldots$) is valid, for any assignment to the variables quantified by the outermost forall ($\mathit{SwitchesOf}(y) \mid\Rightarrow \ldots$), or (3) the outermost quantifier is an $\exists$, and there exists an assignment of the quantified variables ($\mathit{SwitchesOf}(y) \mid \ldots$) such that the rest of the formula is valid.

Theorem 10. *There exists a GL_μ formula that characterizes a set of graphs that is PSPACE-complete.*

Proof. We define a translation of closed QBF and a recursive formula $GVal_\mu$ that characterizes the translation of valid formulas. The thesis follows since validity of QBF is complete for PSPACE.

We first define a variant of the translation $G_n(\Phi, x)$ of Section 3.2, where the depth of a quantifier is identified by the length of the chain of *Forall*-*Exist* edges that leads to it. We translate both formulas and assignment; an assignment is represented as the set of those variables that are assigned to true; any other free variable is implicitly assigned to false. Consider the following formula-assignment pair, where Φ is quantifier-free.

$$\phi = \forall X_1^1 \ldots X_{i_1}^1 . \exists Y_1^2 \ldots Y_{i_2}^2 . \ldots \exists Y_1^n \ldots Y_{i_n}^n . \Phi \qquad \sigma = \{Z_1, \ldots, Z_m\}$$

It is translated as:

$$\begin{aligned}G_\mu(\Phi, \sigma, x) =_{def}\ &\mathit{Switch}(x_0, x_{so}(Z_1)) \mid \ldots \mid \mathit{Switch}(x_0, x_{so}(Z_m))\\ &\mid \mathit{Forall}(x_0, x_1)\\ &\quad \mid \mathit{Switch}(x_1, x_{so}(X_1^1)) \mid \mathit{Var}(x_{so}(X_1^1), x_v(X_1^1))\\ &\quad \mid \ldots \mid \mathit{Switch}(x_1, x_{so}(X_{i_1}^1)) \mid \mathit{Var}(x_{so}(X_{i_1}^1), x_v(X_{i_1}^1))\\ &\quad \mid \mathit{Exist}(x_1, x_2)\\ &\qquad \mid \mathit{Switch}(x_2, x_{so}(Y_1^2)) \mid \mathit{Var}(x_{so}(Y_1^2), x_v(Y_1^2))\\ &\qquad \mid \ldots \mid \mathit{Switch}(x_2, x_{so}(Y_{i_2}^2)) \mid \mathit{Var}(x_{so}(Y_{i_2}^2), x_v(Y_{i_2}^2))\\ &\qquad \mid \mathit{Forall}(x_2, x_3)\\ &\qquad\quad \mid \ldots\\ &\qquad\qquad \mid \mathit{Exist}(x_{n-1}, x_n)\\ &\qquad\qquad\quad \mid \mathit{Switch}(x_n, x_{so}(Y_1^n)) \mid \mathit{Var}(x_{so}(Y_1^n), x_v(Y_1^n))\\ &\qquad\qquad\quad \mid \ldots \mid \mathit{Switch}(x_n, x_{so}(Y_{i_n}^n)) \mid \mathit{Var}(x_{so}(Y_{i_n}^n), x_v(Y_{i_n}^n))\\ &\qquad\qquad\quad \mid [\![\Phi, \epsilon]\!]^x \mid \mathit{Result}(x_p(\epsilon^-), x_p(\epsilon^-))\end{aligned}$$

The formula $GVal_\mu$ is defined as follows.

$$\begin{array}{l}\mu R.\ ((TruePart\ \wedge\ ResultHere)\mid FalsePart)\\ \quad\vee(\exists x,y.\ QRoot(x)\ \wedge\ (Forall(x,y)\mid(SwitchesOf(y)\mid\Rightarrow R)))\\ \quad\vee(\exists x,y.\ QRoot(x)\ \wedge\ (Exist(x,y)\mid(SwitchesOf(y)\mid R)))\end{array}$$

Where $QRoot(x)$ and $SwitchesOf(y)$ and defined as:

$$\begin{array}{ll}QRoot(x) & =_{def}\ \neg\exists x'.\ (Forall(x',x)\ \vee\ Exist(x',x))\mid\mathsf{T}\\ SwitchesOf(x) & =_{def}\ \forall a,y',z.\ (a(y',z)\mid\mathsf{T})\ \Rightarrow\ y'=y\ \wedge\ a=Switch\end{array}$$

The graph $G_\mu(\phi,\sigma,x)$ represents a formula-assignment pair ϕ, σ. The action of removing an outermost $Forall(x_i,x_{i+1})$ or $Exist(x_i,x_{i+1})$ and a set of $Switch(x_{i+1},z)$ edges corresponds to removing the outermost set of quantifications from ϕ and fixing a valuation σ' for the corresponding quantified variables. The subgraph left is not exactly the translation of $\phi',\sigma\sigma'$ because the newly valuated variables have their switches starting from x_{i+1} instead of x_0, but this is irrelevant since nothing in $GVal_\mu$ depends on this difference. The formula $GVal_\mu$ determines the validity of a graph representing a formula-assignment pair ϕ-σ, obtained by translation followed by some instances of this removal operation, as follows.

If no $Forall/Exist$ quantifier is left, $GVal_\mu$ just evaluates ϕ-σ. If there is an outermost universal quantifier, $GVal_\mu$ strips the quantifier, obtaining a formula ϕ', and verifies whether for each assignment σ' for the quantified variables (i.e., for each set of outermost switches that are removed) ϕ'-$\sigma\sigma'$ is valid. If the outermost quantifier is existential, $GVal_\mu$ strips the quantifier, obtaining a ϕ', and verifies whether an assignment σ' exists such that ϕ'-$\sigma\sigma'$ is valid.

This theorem shows that the recursion operator allows the characterization of sets of graphs that we cannot hope to define in second-order logic.

5 Conclusion

We have investigated the complexity and expressive power of the graph logic introduced by Cardelli, Gardner and Ghelli. The graph composition operator $\mid$ in that logic has a natural translation into second-order logic using an existential quantifier over sets of edges. In terms of complexity, the $\mid$ operator is as powerful as the monadic second-order quantifier: GL can express complete problems at all levels of the polynomial hierarchy. The recursion operator in the logic of Cardelli et al. also proves interesting from the point of view of its expressive power. It allows us to define recursions of exponential length and to express PSPACE-complete problems. Nonetheless, the model-checking complexity of the full logic, with $\mid$ and recursion, remains within PSPACE.

References

1. J. R. Büchi. Weak second order arithmetic and finite automata. *Zeitschrift f. Mathematische Logik und Grundlagen d. Mathematik*, 6:66–92, 1960.
2. C. Calcagno, H. Yang, and P. W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *FSTTCS: 21st Conference*, volume 2245 of *Springer LNCS*, pages 108–119, 2001.
3. Cristiano Calcagno, Luca Cardelli, and Andrew D. Gordon. Deciding validity in a spatial logic for trees. *ACM SIGPLAN Notices*, 38(3):62–73, 2003.
4. L. Cardelli, P. Gardner, and G. Ghelli. A spatial logic for querying graphs. In *ICALP: Automata, Languages, and Programming, 29th International Colloquium*, volume 2380 of *Springer LNCS*, pages 597–610, 2002.
5. L. Cardelli and G. Ghelli. TQL: A query language for semistructured data based on the ambient logic. *Mathematical Structures in Computer Science*, 2003. To appear.

6. Luca Cardelli and Andrew D. Gordon. Anytime, anywhere: modal logics for mobile ambients. In *POPL*, pages 365–377, 2000.
7. W. Charatonik, S. Dal-Zilio, A. D. Gordon, S. Mukhopadhyay, and J-M. Talbot. Model checking mobile ambients. *Theoretical Computer Science*, 308:277–331, 2003.
8. G. Conforti, G. Ghelli, A. Albano, D. Colazzo, P. Manghi, and C. Sartiani. The query language TQL. In *International Workshop on the Web and Databases (WebDB), Madison, Wisconsin, USA*, pages 19–24, 2002.
9. B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Graph Transformations*, chapter 5, pages 313–400. World Scientific, 1997.
10. S. Dal-Zilio, D. Lugiez, and C. Meyssonnier. A logic you can count on. In *POPL: 31st ACM Symposium on Principles of Programming Languages*, pages 135–146, 2004.
11. A. Dawar, P. Gardner, and G. Ghelli. Games for the ambient logic. forthcoming.
12. H-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 2 edition, 1999.
13. R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol 7*, pages 43–73, 1974.
14. S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.
15. Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *Proceedings of the 15th International Workshop on Computer Science Logic*, pages 1–19. Springer-Verlag, 2001.
16. Ch. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1995.
17. L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1976.