Separating Graph Logic from MSO

Timos Antonopoulos and Anuj Dawar

University of Cambridge Computer Laboratory, Cambridge CB3 0FD, UK
{timos.antonopoulos,anuj.dawar}@cl.cam.ac.uk

Abstract. Graph logic (GL) is a spatial logic for querying graphs introduced by Cardelli et al. It has been observed that in terms of expressive power, this logic is a fragment of Monadic Second Order Logic (MSO), with quantification over sets of edges. We show that the containment is proper by exhibiting a property that is not GL definable but is definable in MSO, even in the absence of quantification over labels. Moreover, this holds when the graphs are restricted to be forests and thus strengthens in several ways a result of Marcinkowski. As a consequence we also obtain that Separation Logic, with a separating conjunction but without the magic wand, is strictly weaker than MSO over memory heaps, settling an open question of Brochenin et al.

1 Introduction

Graph Logic (GL) was introduced by Cardelli et al. [2] as a query language on labelled directed graphs, modelled on spatial logic. It extends the first-order logic of such graphs (with quantification over vertices, edges and labels) with a spatial connective: thus, a formula $(\varphi|\psi)$ is true in a graph G if, and only if, G can be decomposed into two subgraphs G_1 and G_2 such that $G_1 \models \varphi$ and $G_2 \models \psi$. Here, when we say that G is decomposed into G_1 and G_2, we mean that these two graphs may share vertices but not edges.

It is easy to see that any formula of GL can be translated into an equivalent formula of second-order logic in which the second-order quantifiers are restricted to sets of edges. This is a version of monadic second-order logic (MSO) known as MS_2 in the works of Courcelle (see [4]) and also as *guarded second order logic* or GSO (see [8]). The expressive power and complexity of GL were systematically investigated in [6], where it was shown that, like MSO, GL can express complete problems at every level of the polynomial hierarchy. Moreover, when we restrict ourselves to labelled graphs that code *words* in the natural way, then GL can (just like MSO) define exactly the regular languages. A conjecture that was left open in [6] was that the containment of GL in MS_2 is strict, i.e. that there is a property of graphs definable in MS_2 that is not definable in GL.

Marcinkowski [9] settled this conjecture positively. The property he constructs crucially relies on the presence of an unbounded set of labels in the graphs considered, and on the ability of formulas of GL and (an enhanced) monadic second-order logic, which he calls MSO+ to quantify over labels. This reliance on label-quantification is something Marcinkowski calls a "win on technicalities"

L. de Alfaro (Ed.): FOSSACS 2009, LNCS 5504, pp. 63–77, 2009.

and he explicitly leaves open the more fundamental question of whether GL *without quantification over labels*, which he calls GL−, is strictly weaker than MSO. We settle this question in this paper. To be precise, we show that there is a property of unlabelled, directed *forests* that is expressible in MSO but not in GL. Since, on forests, one can replace quantification over sets of edges with quantification overs sets of vertices, this yields the stronger result that GL does not even contain MS_1.

As a corollary, we obtain a result concerning separation logic (SL), a logic of assertions used in Hoare-style proof systems [10]. Brochenin et al. [1] consider the expressive power of $\mathrm{SL}(*)$, which is separation logic with a separating conjunction $(*)$ but without the magic wand $(-\!\!*)$ and conjecture that it is properly contained in MSO. Since SL is essentially the same as GL over structures known as memory heaps, and since the forests we use to separate GL from MSO can be coded as such heaps, a direct consequence of our proof is a positive resolution of this conjecture.

The MSO-definable property of forests that we demonstrate is not definable in GL is the following: a forest contains a tree in which the number of leaves is a multiple of 3. The use of forests rather than trees is crucial to the proof and thus the question of whether there is a regular tree language that is not definable in GL remains open. We show that a natural candidate, the class of binary trees with an even number of leaves, is definable in GL. Indeed, any regular binary tree language accepted by a bottom-up automaton that is the product of two-state automata can be defined in GL.

In the rest of the paper, we first introduce, in Section 2, the logics we deal with. In Section 4 we investigate the power of GL on trees by showing that a certain class of regular tree languages is included in those definable in GL. Section 3 presents the main result, while Section 5 explores the consequences for separation logic.

2 Background and Preliminaries

Graph Logic was introduced in [2] as a logic for querying graphs. It is based on a view of graphs as terms in a suitable algebra, involving a composition operator. Thus, the graphs are built up from single edges by repeated compositions. In the present paper, we treat the logic instead as an extension of the ordinary first-order logic of graphs, with the composition operator appearing in the formulas. The two views are equivalent, as discussed in [6]. We assume familiarity with the syntax and semantics of first-order logic (FO).

2.1 Graph Logic

Fix a set X of vertex names. A graph G consists of a finite set E of edges, and an incidence map $I_G : E \to X \times X$ that associates with each edge a pair of vertices that we call its endpoints. It is easy to see that any such graph can be seen as a finite, directed, unlabelled graph (with no isolated nodes) in the usual sense. We

will briefly consider the case of edge labels below. The syntax of Graph Logic can then be specified as follows.

Definition 2.1 (GL Syntax). *Let V be a countable set of vertex variables. A GL formula is defined to be one of the following for $t_i \in X \cup V$, and $x \in V$ and for GL formulas φ_i:*

$$\varphi := \mathbf{0} \mid \top \mid E(t_1, t_2) \mid t_1 = t_2 \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1|\varphi_2 \mid \exists x\ \varphi_1.$$

This appears at first sight to be an extension of first-order logic with one extra formula formation rule: $\varphi_1|\varphi_2$. However, the interpretation of the atomic formulas is different from the usual rules for FO. To be precise, $E(t_1, t_2)$ is true in a graph G just in case G consists of a single edge e whose end points are named by the terms t_1 and t_2. To define the semantics of the composition operator, we need to first define graph composition.

Let G_1 and G_2 be two graphs with edge sets E_1 and E_2 and incidence maps I_1 and I_2 respectively. The graph $G = G_1 \mid G_2$ is defined to be the graph whose edge set and incidence map are the disjoint unions $E_1 \uplus E_2$ and $I_1 \uplus I_2$. Note that G_1 and G_2 may share vertices. The semantics of the formula $\varphi_1|\varphi_2$ is now defined by saying $G \models (\varphi_1|\varphi_2)$ just in case $G = G_1 \mid G_2$ for some G_1 and G_2 such that $G_1 \models \varphi_1$ and $G_2 \models \varphi_2$. Formally, the semantics is as given below.

Definition 2.2 (GL Semantics). *Let G be a graph with edge set E and incidence map I. Also let $\alpha : V \to X$ be an assignment of vertex names to variables. We write $\hat{\alpha}$ for the extension of α to the domain $X \cup V$ by letting $\hat{\alpha}(x) = x$ for all $x \in X$. The semantics for the Boolean connectives are defined as usual. The following holds for the rest of the GL constructs.*

$$\begin{array}{ll}
(G, \alpha) \models \mathbf{0} & \textit{iff } E = \emptyset, \\
(G, \alpha) \models \top & \textit{for any graph } G, \\
(G, \alpha) \models E(t_1, t_2) & \textit{iff } E = \{e\} \textit{ and } I(e) = (\hat{\alpha}(t_1), \hat{\alpha}(t_2)), \\
(G, \alpha) \models t_1 = t_2 & \textit{iff } \hat{\alpha}(t_1) = \hat{\alpha}(t_2), \\
(G, \alpha) \models \varphi_1|\varphi_2 & \textit{iff there exist } G_1, G_2 \textit{ s.t. } G = G_1|G_2 \textit{ and } (G_i, \alpha) \models \varphi_i, \\
(G, \alpha) \models \exists x\ \varphi_1 & \textit{iff there is an } a \in X \textit{ s.t. } (G, \hat{\alpha}[x \mapsto a]) \models \varphi_1,
\end{array}$$

where $\hat{\alpha}[x \mapsto a]$ denotes the map that agrees with $\hat{\alpha}$ at all values other than x and maps x to a.

In spite of the variance in the interpretation of atomic formulas, it is not difficult to prove that every FO formula on finite graphs can be translated into a formula of GL. In particular, an atomic FO formula of the form $E(a, b)$ asserting the existence of an edge between vertices a and b is equivalent to the GL formula $(E(a, b)|\top)$. In turn, every GL formula can be translated into one of second-order logic (see [6] for details).

The *monadic second-order logic* (MSO) of graphs comes in two variants, called MS_1 and MS_2 by Courcelle [4]. Both extend FO with second-order quantifiers that can quantify over sets. In the case of MS_1, this second-order quantification is limited to sets of *vertices*, while MS_2 allows quantification over sets of *edges*.

To be precise MS_1 extends FO with a countable set $\mathcal{Z}$ of second-order variables. For every $Z \in \mathcal{Z}$ and term t, $Z(t)$ is an atomic formula and for every formula φ, $\exists Z\varphi$ is also a formula. The latter is satisfied by a graph G just in case there is a set $A \subseteq X$ of vertices such that φ is true in G when all free occurrences of Z are interpreted by the set A. Similarly, MS_2 allows atomic formulas $Z(t_1, t_2)$ and quantifiers $\exists Z\varphi$. The latter is true in a graph G just in case there is a set $A \subseteq E$ of edges such that φ is true in G when all free occurrences of Z are interpreted by the set A.

It is not difficult to see that each formula of GL can be translated into an equivalent formula of MS_2. It is not the case that GL can be translated into MS_1 and in Section 6 we construct an example exhibiting this. The question left open from [6] and [9] is whether there is a sentence of MS_2 that is not equivalent to any sentence of GL. This is the question we answer in the present paper.

Apart from Section 6, we are concerned in this paper with graphs that are trees or forests. On such graphs, MS_2 is no more expressive than MS_1. Indeed, since each vertex has at most one incoming edge, there is an easily definable one-to-one map from edges to vertices that allows us to replace quantification over edges with quantification over vertices. Thus, for our purposes, it makes sense to speak just of MSO, without distinguishing the two varieties. We will simply speak about comparing GL and MSO.

In the intended applications of GL, graphs do not just consist of sets of edges, but also come with labels. There are two ways in which labels can be introduced into the language of GL. If the set of possible labels is a fixed finite set Σ, we may regard each $\sigma \in \Sigma$ as defining a set of edges E_σ with associated incidence relation $I : E_\sigma \to X \times X$. The logic then allows atomic formulas $E_\sigma(t_1, t_2)$ for each $\sigma \in \Sigma$. On the other hand, if the set of labels is unbounded, we include in the language the set Σ and a countable set of label variables L. A graph now is a set of edges E together with an incidence relation $I : X \times \Sigma \times X$. We allow atomic formulas $E_\sigma(t_1, t_2)$ for each $\sigma \in \Sigma \cup L$ and also allow equality testing and quantification over labels. Thus, for $\sigma, \tau \in \Sigma \cup L$, we have atomic formulas $\sigma = \tau$ and for any formula φ, we can form the formula $\exists l\varphi$ for $l \in L$. Marcinkowski [9] termed this extended logic GL+ and proved that it was strictly weaker than MSO+, the corresponding extension of MS_2. Our result strengthens his. In the rest of this paper, we will not be concerned with GL+ or MSO+.

Regular Languages. The case of bounded alphabets Σ includes many interesting applications of MSO. For instance, Σ-labelled graphs which consist of a single path from a source s to a terminal t can be identified with words over the alphabet Σ. It is well known, by results of Büchi, Elgot and Trakhtenbrot (see [7]) that in this case MSO can define exactly the regular languages. It was shown in [6] that the same is true of GL. The correspondence between MSO and finite automata extends further to Σ-labelled trees, and it remains an open question (see [9]) whether GL can express all regular tree languages. In Section 6, we show that one suggested candidate for separation—the class of binary trees with an even number of leaves—is definable in GL. Indeed, any binary tree language accepted by a bottom-up deterministic two-state automaton is shown to be

definable. However, if the graphs are *forests* rather than trees, our main result in Section 3 proves that GL is strictly weaker than MSO.

Recall the following. A deterministic bottom-up tree automaton is a tuple $A = (\Sigma, \delta_2, \delta_1, q_1, Q, Q_f)$, where Σ is the alphabet, Q the set of states, $q_1 \in Q$ the initial state, $Q_f \subseteq Q$ the set of accepting states and $\delta_1 : Q \times \Sigma \to Q$ and $\delta_2 : Q \times \Sigma \times Q \times \Sigma \to Q$ are two transition functions. Given a Σ-labelled binary tree, the automaton A assigns a state in Q to each node of the tree. The leaves are assigned the initial state q_1. Each node a with one child is assigned $\delta_1(q, \sigma)$ where σ is the label on the edge leaving a and q is the state assigned to its child. Similarly, each node a with two children is assigned $\delta_2(q, \sigma, q', \sigma')$ where σ and σ' are the labels on the two edges leaving a and q and q' are the states assigned to its two children. A accepts a tree T if the state assigned to the root of T is in Q_f.

2.2 GL Games

The main tool for proving non-expressibility of a property in FO or MSO is Ehrenfeucht-Fraïssé Games (see [7]). Such games have also been adapted to spatial logics (see [5]). Here we present an adaptation of the game for GL.

A GL Game is played by two players, Spoiler and Duplicator, and consists of k rounds, for some $k \in \mathbb{N}$. The game is played on two graphs F and G with the initial position being $\langle (F, \bar{c}), (G, \bar{c}) \rangle$ for some tuple $\bar{c}$ of vertex names. The position at a round $i \leq k$ is defined to be a pair of structures with distinguished vertices $\langle (F^i, \bar{a}), (G^i, \bar{b}) \rangle$ where $\bar{a} = a_1, \ldots, a_p$ and $\bar{b} = b_1, \ldots, b_p$ are tuples of vertex names extending $\bar{c}$, and F^i and G^i are subgraphs of F and G respectively.

At the beginning of the ith round, Spoiler chooses one of the two structures $(F^i, \bar{a})$ or $(G^i, \bar{b})$ and makes either a *colouring* move or a *first order* move. Suppose, without loss of generality, that Spoiler chooses $(F^i, \bar{a})$. For a first order move, he chooses a vertex $a_{p'}$ in F^i and Duplicator must respond with a vertex $b_{p'}$ in G^i. The position at the next round is $\langle (F^i, \bar{a}, a_{p'}), (G^i, \bar{b}, b_{p'}) \rangle$.

If he chooses to play a colouring move, Spoiler finds two graphs F_1^i and F_2^i such that $F^i = F_1^i | F_2^i$. Duplicator must respond with two graphs G_1^i and G_2^i such that $G^i = G_1^i | G_2^i$. Spoiler then decides whether the position at the next round is $\langle (F_1^i, \bar{a}), (G_1^i, \bar{b}) \rangle$ or $\langle (F_2^i, \bar{a}), (G_2^i, \bar{b}) \rangle$. In general, when describing a colouring move, we will say that Spoiler has coloured the edges in F_1^i *white* and those in F_2^i *black* and Duplicator has responded by colouring G_1^i white and G_2^i black.

The game ends after k rounds, or if one of the two graphs in some position is empty or consists of a single edge. Let $h : X \rightharpoonup X$ be the partial map defined by $a_j \mapsto b_j$. Spoiler wins if one of the following three conditions holds, and Duplicator wins in all the other cases.

1. Exactly one of the graphs is empty.
2. One of the graphs is a single edge, with both endpoints in the domain of h and h is not an isomorphism between the two graphs.
3. The mapping h is not one-to-one.

We define the quantifier rank of a GL formula, by counting first-order quantifiers and composition operators equally. To be precise, the quantifier rank of a

GL formula φ is $\mathrm{rank}(\varphi)$ which is defined as usual for the Boolean connectives and as follows for the rest:

If $\varphi = \mathbf{0}, \top, t_1 = t_2, E(t_1, t_2)$	then $\mathrm{rank}(\varphi) = 0$.
If $\varphi = \varphi_1 \vert \varphi_2$	then $\mathrm{rank}(\varphi) = \max(\mathrm{rank}(\varphi_1), \mathrm{rank}(\varphi_2)) + 1$.
If $\varphi = \exists x\ \varphi_1$	then $\mathrm{rank}(\varphi) = \mathrm{rank}(\varphi_1) + 1$.

The proof of the following lemma then follows the standard methods for Ehrenfeucht-Fraïssé games.

Lemma 2.3 ([6]). *If Duplicator has a winning strategy for the k-round game on graphs F and G with initial position $\langle (F, \bar{c}), (G, \bar{c}) \rangle$, then for any GL formula φ with rank$(\varphi) \leq k$ and using only names from $\bar{c}$, it holds that*

$$F \models \varphi \text{ if, and only if, } G \models \varphi.$$

The main use of this lemma is to show that some property P is not expressible in GL. This can be formulated as in the following corollary. Since we are interested in properties that are invariant under the choice of vertex names, we can restrict ourselves to games in which $\bar{c}$ is empty in the initial position.

Corollary 2.4. *A property P is inexpressible in GL, if and only if for each $k \in \mathbb{N}$, there exist structures F_k and G_k, such that $F_k \in P$ and $G_k \notin P$, and Duplicator has a winning strategy for the k-round GL played game on $\langle F_k, G_k \rangle$.*

We write $F \equiv_k^{\mathrm{GL}} G$ to denote that F and G cannot be distinguished by any sentence φ of GL with $\mathrm{rank}(\varphi) \leq k$. Similarly, $F \equiv_k^{\mathrm{MSO}} G$ denotes that F and G agree on all MSO sentences with quantifier rank at most k. A translation of GL formulas to MSO is given in [6] that takes a GL formula of quantifier rank k to an MSO formula of rank at most $2k + 1$. From this, Lemma 2.5 below follows.

Lemma 2.5. *For any $k \in \mathbb{N}$, and graphs G, H, if $G \equiv_{2k+1}^{\mathrm{MSO}} H$ then $G \equiv_k^{\mathrm{GL}} H$.*

Disjoint Unions. We will often make use of constructions involving disjoint unions of graphs as well as unions disjoint apart from a fixed number of named vertices. We make these notions precise and fix notation here. Recall that we are primarily interested in properties of graphs that are invariant under isomorphisms or, equivalently, invariant under renaming of vertices. Given graphs G and H, we write $G \oplus H$ for the *disjoint union* of G and H. This is a graph $G' \vert H$ for a graph G' that is isomorphic to G but shares no vertices with H. For a fixed tuple $\bar{a}$ of vertex names, we write $G \oplus_{\bar{a}} H$ for the graph $G' \vert H$ where G' is obtained from G by renaming all vertices apart from those in $\bar{a}$ to be distinct from any vertex in H. In other words, $G \oplus_{\bar{a}} H$ is a graph obtained from the disjoint union of G and H while identifying vertices in $\bar{a}$. For an indexed family $\{G_i \mid i \in I\}$ of graphs, we write $\bigoplus_{i \in I} G_i$ to denote the disjoint union of all the graphs in the family. For a natural number n, we also write $n \cdot G$ for the graph $\bigoplus_{1 \leq i \leq n} G_i$ where each G_i is isomorphic to G.

3 Separating GL from MSO

In this section we present the main result, namely that there are properties of forests that are expressible in MSO but not in GL. The property in question is that one of the trees in the forest has a number of leaves that is a multiple of three. To see that this property is MSO definable, note that there is a simple 3-state deterministic bottom-up tree automaton that checks whether a given tree T has a number of leaves that is a multiple of three. Thus, there is an MSO sentence θ that defines this class of trees. Since we can also construct an MSO formula $\mathsf{path}(x, y)$ that asserts that there is a path from x to y, we can use this to obtain a formula that asserts the existence of a root x such that the tree of nodes reachable from x satisfies θ.

We begin with a simple intuitive example to illustrate why the colouring move in a GL game involves a loss of information for Spoiler, which Duplicator can exploit in a way that she cannot in the corresponding MSO game. Consider two graphs, G_1 and G_2. Let $G = G_1 \oplus G_2$ and let G' be $G_1 \oplus_v G_2$ for some vertex v. On a game played on $\langle G, G' \rangle$, if Spoiler plays a colouring move that splits either graph into G_1 and G_2, it is clear that Duplicator has a winning response unless the vertex v was previously chosen. Thus, information on how the original graph (G or G') was connected has been lost.

This simple example illustrates the idea behind the construction of the two forests on which the game will be played. The forests we consider consist of a single *comb*—i.e. a binary tree consisting of a simple path with other simple paths branching off from it—and a large number of disjoint simple paths. These simple paths act as *noise* which allow Duplicator, in response to a suitable colouring move by Spoiler, to remove some of the branches of the comb (thereby changing the number of leaves) and hide them among the noise. We now proceed to a more detailed description of the construction.

Definition 3.1. *A* fork *is a node in a binary tree that has two distinct children.*

A comb *is a binary tree where for any two distinct forks v_1 and v_2, either v_1 is an ancestor of v_2 or v_2 is an ancestor of v_1.*

Let C be a comb, and r its root. As long as there is at least one fork in C, there exist two leaves t, t' such that the path from r to either of them contains all the forks of C. Fix t to be one of those leaves and call the path from r to t, the *spine* of the comb C. Each fork a of C, has one child on the spine of C while the other is a vertex that is the root of a subtree consisting of a simple path to a leaf b. For each fork a, we call the path from a to b, a *tooth* of the comb C. Note that the number of leaves of a comb is one more than the number of teeth.

Let $n, s \in \mathbb{N}$. We write $C_{n,s}$ for the comb with n teeth where the length of each tooth, as well as the distance between any two successive forks is s. We also define $C^{-i}_{n,s}$, for $1 \leq i \leq n$, to be the comb $C_{n,s}$ with the i-th tooth missing. That is, the distance between the $(i-1)$st fork and the next one is $2s$. Finally, let S_n denote a string (i.e. a tree consisting of a single path) of length n.

Suppose a, a' are successive forks in the spine of a comb. Then the substring of the spine with endpoints a, a' is called a *segment*. A *block* of a comb is a segment with endpoints a, a', together with the tooth attached to a'.

The following lemma states some easy consequences of the fact that MSO can only define regular languages when restricted to strings.

Lemma 3.2. *For each $k \in \mathbb{N}$, there exist $s, n, l \in \mathbb{N}$, such that:*

1. *for every $w > s$ and every m, $S_w \equiv_k^{\mathrm{MSO}} S_{w+ms}$,*
2. *for every $t > n$, and every m, $C_{t-2l,ms} \equiv_k^{\mathrm{MSO}} C_{t+2,ms}$.*

Lemma 3.2 essentially gives specific periodic properties for the sizes of strings and of combs of constant segment length, with respect to $\equiv_k^{\mathrm{MSO}}$-equivalence, for any $k \in \mathbb{N}$. One consequence we can derive is that $C_{n,s} \equiv_k^{\mathrm{MSO}} C_{n+1,s}^{-i}$, for any n and the s given by the lemma, since $S_s \equiv_k^{\mathrm{MSO}} S_{2s}$ and the comb $C_{n+1,s}^{-i}$ can be seen as $C_{n,s}$ with its ith segment of length s replaced by a segment of length $2s$.

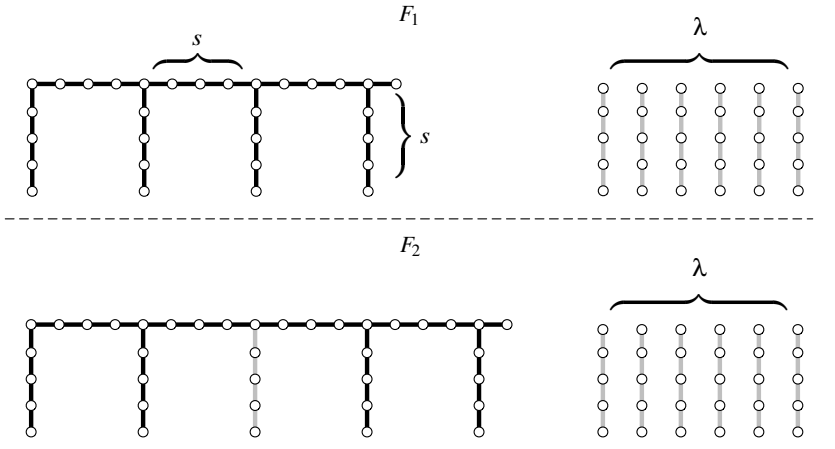
The next lemma can be proved by a standard application of Ehrenfeucht-Fraïssé games and appears in [3] for general structures $\mathbb{A}$ and $\mathbb{B}$.

Lemma 3.3. *For any $k \in \mathbb{N}$, there is $\lambda \in \mathbb{N}$, such that if $\mathbb{A}$ is the disjoint union of λ pairwise $\equiv_k^{\mathrm{MSO}}$-equivalent structures, and $\mathbb{B}$ is the disjoint union of $\lambda + 1$ such structures, each $\equiv_k^{\mathrm{MSO}}$-equivalent to the ones in $\mathbb{A}$, then $\mathbb{A} \equiv_k^{\mathrm{MSO}} \mathbb{B}$.*

Before giving the details of the construction, we consider an example. Consider two forests $F_1 \cong C_{n,s} \oplus \lambda \cdot S_s$, and $F_2 \cong C_{n+1,s} \oplus \lambda \cdot S_s$. That is, each one consists of the disjoint union of a comb with a large number of strings. Suppose that, in the GL game played on this pair of structures for $\lfloor (k-1)/2 \rfloor$ moves, Spoiler plays a colouring move on F_1, by colouring the comb $C_{n,s}$ in black, and the set of disjoint strings $\lambda \cdot S_s$ in white. Then Duplicator can respond by colouring for some $1 \leq i \leq n$, the subgraph $C_{n+1,s}^{-i}$ in black and the λ copies of S_s together with the remaining tooth of the comb in white, as shown in the figure below. By Lemmas 3.2 and 3.3, whichever the pair of graphs on which Spoiler chooses to continue the game, they are $\equiv_k^{\mathrm{MSO}}$-equivalent. Since $\equiv_k^{\mathrm{MSO}}$-equivalence implies $\equiv_{\lfloor (k-1)/2 \rfloor}^{\mathrm{GL}}$, this means that Duplicator has a winning strategy for the rest of the game. This shows that using the colouring move to distinguish the comb from the noise, which would have been the right move for Spoiler in the MSO game on this pair of structures, is a losing move in the GL game. Extending this idea, we show that by taking the comb to be big enough and by adding enough noise, we can ensure that Spoiler has no good moves.

Let $\mathcal{F}$ be the class of forests in which some tree has 0 (mod 3) leaves. In what follows, we show that for any $k \in \mathbb{N}$, there exist two forests F and G, such that $F \in \mathcal{F}$, $G \notin \mathcal{F}$, and Duplicator wins the k-round game on F and G. We proceed with the details of the construction of these two forests.

Fix $k \in \mathbb{N}$ and let s and n be as given by Lemma 3.2 for k, and λ as given by Lemma 3.3 for k. By Lemma 3.2, for every $w \in \mathbb{N}$ there is a $w' \in \{1, \ldots, 2s\}$ with $S_w \equiv_k^{\mathrm{MSO}} S_{w'}$ (indeed, either $w \leq 2s$ and we can take $w = w'$ or we can find a suitable w' and m such that $w = w' + ms$). We define $N = \prod_{1 \leq i,j \leq 2s} (i + j)$.



Note that this has the property that $(i + j) \mid N$ (i.e. $i + j$ divides N) for all $i, j \leq 2s$.

Let $f_k = (2^5 \cdot \lambda s N)^k \cdot 6n \cdot \lambda \cdot 2^{6N} - 1$. We define the forests F and G as follows.

$$F = C_{f_k,N} \oplus \bigoplus_{2s+1 \leq i \leq 3s} (\lambda \cdot 2^{3sk} \cdot S_i),$$
$$G = C_{f_k+2,N} \oplus \bigoplus_{2s+1 \leq i \leq 3s} (\lambda \cdot 2^{3sk} \cdot S_i).$$

The collection of strings $\bigoplus_{2s+1 \leq i \leq 3s} (\lambda \cdot 2^{3sk} \cdot S_i)$ in both graphs, is called *noise*, and each individual string from that collection is called a *noise-string*.

By Lemma 2.5, for each $\ell \geq 3$, and for any two graphs H_1 and H_2, $H_1 \equiv_\ell^{\mathrm{MSO}} H_2$ implies that $H_1 \equiv_{\lfloor (\ell-1)/2 \rfloor}^{\mathrm{GL}} H_2$. In the following, to simplify notation we show that for any $k \geq 1$, Duplicator can win the $\lfloor (k-1)/2 \rfloor$-round GL game on F and G. To show that $F \equiv_{\lfloor (k-1)/2 \rfloor}^{\mathrm{GL}} G$, we establish that Duplicator can maintain the following condition in the $\lfloor (k-1)/2 \rfloor$-round game on F and G.

> If the game position after i rounds is $(F_i, \bar{a})$ and $(G_i, \bar{b})$, then one of the two conditions below holds, for $k' = k - i$:
> 1. $(F_i, \bar{a}) \equiv_{k'}^{\mathrm{MSO}} (G_i, \bar{b})$, or
> 2. $F_i = F' \oplus_{c_1,c_2} \overline{F'}$ and $G_i = G' \oplus_{d_1,d_2} \overline{G'}$, where:
>   (a) $(F', c_1, c_2) \cong (C_{f_{k'},N}, r, t) \oplus \bigoplus_{2s+1 \leq i \leq 3s} (\lambda \cdot 2^{3sk'} \cdot S_i)$,
>   (b) $(G', d_1, d_2) \cong (C_{f_{k'}+2,N}, r, t) \oplus \bigoplus_{2s+1 \leq i \leq 3s} (\lambda \cdot 2^{3sk'} \cdot S_i)$,
>   (c) no element of $\bar{a}$ is in F' and no element of $\bar{b}$ is in G',
>   (d) $(\overline{F'}, c_1, c_2) \equiv_{k'}^{\mathrm{MSO}} (\overline{G'}, d_1, d_2)$.

Notice that for any $i \leq \lfloor (k-1)/2 \rfloor$, $(F_i, \bar{a}) \equiv_{k-i}^{\mathrm{MSO}} (G_i, \bar{b})$ implies $(F_i, \bar{a}) \equiv_{\lfloor (k-1)/2 \rfloor - i}^{\mathrm{GL}} (G_i, \bar{b})$. The condition above states that at each round i of the game, either both graphs are $\equiv_{k'}^{\mathrm{MSO}}$-equivalent, and thus Duplicator wins the game, or the following holds. In both graphs F_i and G_i after i rounds, there exist subgraphs F' and G' respectively, each composed of a large enough comb and enough noise-strings. Furthermore the complements of these graphs F' and G' inside F_i and G_i, are $\equiv_{k'}^{\mathrm{MSO}}$-equivalent.

Duplicator has a reply to any move Spoiler makes whenever condition (1) holds, namely $(F_i, \bar{a}) \equiv^{\mathrm{MSO}}_{k'} (G_i, \bar{b})$. We need to show, that if (2) holds, Duplicator has a response that maintains the condition. We proceed by induction on the number of rounds. In the beginning of the game, (2) holds by construction.

Suppose then that at some round i, (2) holds. Suppose furthermore that Spoiler makes a first order move, and chooses a vertex in one of the two graphs. If he chooses a vertex in $\overline{F'}$ or $\overline{G'}$, Duplicator can reply since $\overline{F'} \equiv^{\mathrm{MSO}}_{k'} \overline{G'}$. The subgraphs guaranteeing the condition (2) can then be found. Assume then that Spoiler picks a vertex v in F' or G', and by symmetry, assume that he does so in F'. Then two subgraphs ensuring the condition (2) holds, can also be found, for $k' - 1$. The details are omitted.

Consider the case where Spoiler makes a colouring move. As the arguments are similar for both cases, we assume that Spoiler colours the graph G_i. Each noise-string in G_i of length h receives one of 2^h possible colourings c, i.e. a labelling of each of the edges as either black or white. Since there are, in general, many more noise-strings of length h than this, some colouring may be repeated many times. We will call the most frequently occurring colouring c (or any one of them if there is a tie), the *primary colouring* of the noise-strings of length h.

A colouring move of Spoiler is considered in cases, depending on how he colours the subgraph F' or G'. We give an outline of the main procedure that Duplicator applies as a reply to many of Spoiler's moves. We denote with C the subgraph inside F' that is isomorphic to the comb $C_{f_{k'},N}$, and similarly we denote with D the respective subgraph isomorphic to $C_{f_{k'}+2,N}$ of G'. The remaining subgraphs in F' and G' comprising the set of noise-strings are denoted by F_S and G_S respectively.

The graph F_i (resp. G_i) comprises the subgraphs C, F_S and $\overline{F'}$ (resp. D, G_S and $\overline{G'}$). Since $F_S \cong G_S$ and $\overline{F'} \equiv^{\mathrm{MSO}}_{k'} \overline{G'}$, Duplicator has a reply to the way Spoiler colours G_S and $\overline{G'}$. For the response to the colouring of D, Duplicator proceeds as follows. She defines vertices c_3, c_4 and d_3, d_4 in (C, c_1, c_2) and (D, d_1, d_2) respectively, so that:

$$(C, c_1, c_2) = (C_1, c_1, c_3) \oplus_{c_3} (C_2, c_3, c_4) \oplus_{c_4} (C_3, c_4, c_2),$$
$$(D, d_1, d_2) = (D_1, d_1, d_3) \oplus_{d_3} (D_2, d_3, d_4) \oplus_{d_4} (D_3, d_4, d_2),$$

for some C_1, C_2, C_3 and D_1, D_2, D_3, such that $C_1 \equiv^{\mathrm{MSO}}_{k} D_1$ and $C_3 \equiv^{\mathrm{MSO}}_{k} D_3$. Furthermore, she ensures the following for C_2 and D_2. Given a choice of vertices d_3 and d_4, either the black and the white components of D_1 and D_3 are disconnected from the black and the white ones in D_2 respectively or not. In the first case, Duplicator ensures the same for c_3, c_4 and also makes sure that all the black and white components in D_2 appear in C_2 in equal numbers, and C_2 contains additional components that appear more than λ times in the graphs C_1, C_3 and F_S, and thus also more than λ times in D_1,D_3 and G_S, by definition. The resulting white and black subgraphs of F' and G' are therefore respectively $\equiv^{\mathrm{MSO}}_{k}$-equivalent (and therefore $\equiv^{\mathrm{MSO}}_{k'}$-equivalent).

In the second case, Duplicator ensures that the spine of C_2 is $\equiv^{\mathrm{MSO}}_{k}$-equivalent to the whole of D_2, and the teeth of C_2 are split into white and black components

that appear more than λ times in C_1,C_3 and F_S, and the splitting is such that the resulting components from the teeth of C_2 are disconnected from the spine of C_2. Again, the resulting white (respectively black) subgraphs of F' and G' are $\equiv_k^{\mathrm{MSO}}$-equivalent (and therefore $\equiv_{k'}^{\mathrm{MSO}}$-equivalent).

As was stated above, the argument for the colouring moves of Spoiler, is considered in cases, depending on how Spoiler chooses to colour the subgraphs D and G_S. In all cases considered except one, Duplicator can guarantee that condition (1) holds in the next stage, that is $(F_{i+1}, \bar{a}) \equiv_{k'-1}^{\mathrm{MSO}} (G_{i+1}, \bar{b})$. The one exception is the case where Spoiler colours in black some part of a segment in D, that is longer than s edges and furthermore: for all $h \in [2n+1, 3n]$, the noise-strings of length h are primarily coloured black; no substring of a segment, larger than s edges is coloured white in D by Spoiler; and Spoiler colours at most $8 \cdot \lambda \cdot s \cdot N$ blocks in D using both colours.

We omit the details of the argument, which lead us to the following theorem.

Theorem 3.4. *The class of forests that contain a tree with* 0 (*mod* 3) *number of leaves, is not definable in GL.*

4 GL on Binary Trees

It is known that GL is as expressive as MSO on words, and we have shown in the previous section that it is strictly weaker on some classes of graphs, in particular forests. A natural question that arises is whether GL is as expressive as MSO on trees, especially as the latter is a widely studied logic of trees. We are not able to settle this question, but we do show that GL can be more expressive than expected. In particular, the property of a tree having an even number of leaves is expressible in GL. Note that, we do not know how to extend this to forests—i.e. to show that use of the modulus 3 in Theorem 3.4 is essential. Nor do we know how to express in GL that a tree has a number of leaves that is a multiple of 3. Thus, a gap remains between Theorem 3.4 and Corollary 4.2 below.

We consider binary trees, where each vertex has at most two children.

Theorem 4.1. *Any binary tree language accepted by a bottom-up deterministic automaton with 2 states, is definable in GL.*

Proof. Suppose that $A = (\Sigma, \delta_2, \delta_1, q_1, Q, Q_f)$ is a deterministic bottom-up binary tree automaton with 2 states, i.e. $Q = \{q_1, q_2\}$. We show how to construct a formula that defines the class of binary trees in which A assigns q_1 to the root. We do this just for binary trees where the root is a fork. The result then easily extends to general binary trees, since such a tree consists of the composition of a tree with a fork root and a simple word, and the behaviour of the automaton on words is known to be expressible in GL.

Let $\mathcal{T}$ be the class of binary trees T such that A assigns the state q_1 to all the leaves and the root of T, and assigns the state q_2 to all the forks of T other than the root. Suppose there is a GL formula ψ defining the class of forests where each tree in the forest is in $\mathcal{T}$. Before defining this formula ψ explicitly, we show how it can be used to define the class of trees accepted by A.

In particular, we show that for any tree T, $T \models (\psi \mid \psi)$ if, and only if, the state assigned to the root of T by A is q_1. For the *only if* direction assume that $T \models (\psi \mid \psi)$. Then it can be split into two forests F_1 and F_2, where each one satisfies ψ. Then, the root and the leaves of each tree in F_1 and F_2, is assigned the state q_1, and since all roots have two children, if a tree T_1 in F_1 is connected to a tree T_2 in F_2 within the tree T, then the root of one is the same vertex as the leaf of the other. Therefore, A assigns q_1 to the root of T.

For the *if* direction, suppose that the root of a tree T is assigned the state q_1 by A. For each fork x in T to which A assigns the state q_1, define the subtree rooted at x and whose leaves are the closest descendants to x that are either leaves or forks to which A assigns the state q_1. By definition, all forks other than the root in such a tree, are assigned the state q_2 by A. We define F_1 and F_2 to be two forests, each containing the subtrees defined above, such that no two such subtrees that are connected in T, are both in the same forest. According to the above, $F_i \models \psi$, and therefore $T \models (\psi \mid \psi)$.

We now give an explicit definition of the formula ψ that is used in the argument above. Recall that $\mathcal{T}$ is the class of binary trees T such that A assigns the state q_1 to all the leaves and the root of T, and assigns the state q_2 to all the forks of T other than the root. The formulas $\mathsf{root}(x)$ and $\mathsf{leaf}(x)$ are used to identify roots and leaf vertices respectively, in first order logic. Finally, the formula $\mathsf{fork}(x)$ expresses in FO that the vertex x is a fork, and $\mathsf{one\text{-}child}(x)$ expresses in FO that the vertex x has a single child.

Any path between two vertices x and y, where x and all vertices in that path apart from y, are non-forks, is called a *unary branch*. On a unary branch, a tree automaton works as a word automaton, and uses only the transition function δ_1. On words we know that GL and MSO are equi-expressive, so let the formula φ_{q_i,q_j} for $q_i, q_j \in \{q_1, q_2\}$ be the GL formula that defines the class of words on which the automaton A, starting with state q_i at the first vertex, assigns the state q_j at the last one.

The vertices x and y in some tree T, with x an ancestor of y, are the endpoints of a unary branch if and only if $(T, x, y) \models \mathsf{unaryBranch}(x, y)$, where:

$$\begin{aligned}
\mathsf{Path}(x,y) &= \mathsf{root}(x) \wedge \mathsf{one\text{-}child}(x) \wedge \neg\mathsf{root}(y) \wedge \mathsf{leaf}(y) \wedge \\
&\quad \wedge \forall z\ (z \neq x \wedge z \neq y \rightarrow \neg\mathsf{root}(z) \wedge \mathsf{one\text{-}child}(z)), \\
\mathsf{unaryBranch}(x,y) &= \mathsf{one\text{-}child}(x) \wedge (\mathsf{leaf}(y) \vee \mathsf{fork}(y)) \wedge (\mathsf{Path}(x,y) \mid \top) \wedge \\
&\quad \wedge \forall z\ (z \neq y \wedge (\mathsf{Path}(x,z) | \mathsf{Path}(z,y) | \top) \rightarrow \mathsf{one\text{-}child}(z)).
\end{aligned}$$

We present the following formulas that assist with defining the GL formula ψ.

$$\begin{aligned}
\mathsf{unary}\text{-}q_2q_j(x) &= \exists y\ (\mathsf{fork}(y) \wedge \mathsf{unaryBranch}(x,y) \wedge (\mathsf{Path}(x,y) \wedge \varphi_{q_2,q_j} \mid \top)), \\
\mathsf{unary}\text{-}q_1q_j(x) &= \exists y\ (\mathsf{leaf}(y) \wedge \mathsf{unaryBranch}(x,y) \wedge (\mathsf{Path}(x,y) \wedge \varphi_{q_1,q_j} \mid \top)), \\
\mathsf{unary}\text{-}q_i(x) &= \mathsf{one\text{-}child}(x) \wedge (\mathsf{unary}\text{-}q_1q_i(x) \vee \mathsf{unary}\text{-}q_2q_i(x)), \\
\mathsf{state}\text{-}q_2(x) &= (\mathsf{fork}(x) \wedge \neg\mathsf{root}(x)) \vee \mathsf{unary}\text{-}q_2(x), \\
\mathsf{state}\text{-}q_1(x) &= (\mathsf{leaf}(x) \vee \mathsf{root}(x)) \vee \mathsf{unary}\text{-}q_1(x).
\end{aligned}$$

The formula $\mathsf{unary}\text{-}q_2q_j(x)$ expresses that x is the top vertex of a unary branch to some y, a descendant of x, that is a fork, and furthermore that on this unary

branch, if the automaton A starts at state q_2 at y, it will reach the state q_j at x. The case is similar for the formula $\mathsf{unary}\text{-}q_1q_j(x)$, but in this case y is a leaf and the automaton reaches q_j at x if it starts with state q_1 at y. The formula $\mathsf{unary}\text{-}q_j(x)$ is simply the disjunction of the two formulas above. Finally, the formula $\mathsf{state}\text{-}q_1(x)$ holds at some vertex if it is a leaf, a root or if it is the top vertex of a unary branch, where A reaches q_1 according to the assumptions stated above. Similarly for $\mathsf{state}\text{-}q_2(x)$ applying to forks that are not roots.

We show that for any vertex x in any tree T in the class $\mathcal{T}$, $(T, x) \models \mathsf{state}\text{-}q_i(x)$ if, and only if, the state q_i is assigned to the vertex x by A. Now define

$$\varphi_{q_1}(x, y, z) = \bigvee_{\delta(q_i,\sigma,q_j,\sigma')=q_1} (\mathsf{state}\text{-}q_i(y) \wedge E_\sigma(x, y) \wedge \mathsf{state}\text{-}q_j(z) \wedge E_{\sigma'}(x, z)),$$
$$\mathsf{fork\text{-}roots} = \forall x \ (\mathsf{root}(x) \rightarrow \mathsf{fork}(x)),$$
$$\psi = \mathbf{0} \vee \big(\mathsf{fork\text{-}roots} \wedge \forall x \ \big((\exists y, z \ (y \neq z) \wedge \varphi_{q_1}(x, y, z)) \leftrightarrow \mathsf{root}(x)\big)\big).$$

Notice that for any forest F, $F \models \psi$ if and only if for every tree T in F, $T \models \psi$. This is because ψ expresses that a combination of states and symbols that lead to the state q_1, occurs at a fork if and only if this fork is the root of the tree it belongs to. Furthermore, whether a vertex satisfies any of the formulas given above, depends only on the tree the vertex belongs to.

Thus, we are left with the following claim to prove, the proof of which is omitted due to lack of space.

Claim. For any tree T, $T \models \psi$ if, and only if, $T \in \mathcal{T}$.

Corollary 4.2. *The class of binary trees with an even number of leaves is GL definable.*

5 Separation Logic

Separation Logic (SL) is a logic for analyzing programs that involve pointer variables for memory management, introduced by Reynolds and widely studied since then (see [10]). We give a brief account here, and refer to [1] for details.

The structures on which Separation Logic works, consist of a partial function representing the memory heap of a program. Let Loc be a countable set of *locations*, namely memory addresses and let Var be a set of variables.

Definition 5.1 ([1]). *A memory state is a pair (s, h) such that $s : Var \rightarrow Loc$ and h is a partial function of type $h : Loc \rightharpoonup Loc$. The function s is the store and the function h is the heap of the memory state.*

When the domains of two partial functions h_1 and h_2 are disjoint, this fact is denoted by $h_1 \perp h_2$, and their disjoint union is denoted by $h_1 * h_2$. The syntax of a Separation Logic formula is inductively defined as:

$$\psi := x = y \mid x \hookrightarrow y \mid \psi_1 \wedge \psi_2 \mid \neg\psi_1 \mid \exists x.\psi_1(x) \mid \psi_1 * \psi_2 \mid \psi_1 \mathrel{-\!\!*} \psi_2,$$

where $x, y \in$ Var and ψ_1, ψ_2 are SL formulas. The semantics of the non-obvious connectives is given below.

$$
\begin{aligned}
(s,h) &\models x \hookrightarrow y \quad \Leftrightarrow h(s(x)) = s(y),\\
(s,h) &\models \psi_1 * \psi_2 \;\Leftrightarrow \text{there are } h_1, h_2 \text{ such that } h = h_1 * h_2\\
&\qquad\qquad\qquad\quad \text{and } (s,h_i) \models \psi_i, i \in \{1,2\},\\
(s,h) &\models \psi_1 \mathrel{-\!\!*} \psi_2 \Leftrightarrow \text{for any } h' \text{ disjoint from } h \text{ such that}\\
&\qquad\qquad\qquad\quad (s,h') \models \psi_1, (s, h * h') \models \psi_2.
\end{aligned}
$$

Note that if a formula φ does not have any free variables then $(s,h) \models \varphi$ if, and only if, $(s',h) \models \varphi$ for all s'. In this case we simply write $h \models \varphi$.

In [1], the syntactic fragment SL($*$) is considered, in which the conjunction operation, $*$, is present but its adjoint the magic wand, $\mathrel{-\!\!*}$, is absent. They show that this logic is decidable by a translation to MSO. They conjecture that the inclusion of SL($*$) in MSO in terms of expressive power is strict.

We can associate with a heap h the graph G_h consisting of the set of edges (x,y) such that $h(y) = x$. It is straightforward to see that for every sentence φ of SL($*$) there is a sentence φ^* of GL such that $G_h \models \varphi^*$ if, and only if, $h \models \varphi$. Note, in particular, that for every forest G there is an h such that $G \cong G_h$.

Say that a location l is a *leaf* of h if $h(l)$ is defined and there is no l' such that $h(l') = l$. Define a *component* C of h to be a connected component of the graph G_h. Then the following is a consequence of Theorem 3.4.

Theorem 5.2. *There is no sentence θ of SL such that $h \models \theta$ if, and only if, some component of h has* 0 (*mod* 3) *leaves.*

As a consequence, we resolve the conjecture of Brochenin et al. [1].

Corollary 5.3. *SL($*$) is strictly less expressive than MSO on memory states.*

6 GL Is Not Included in MS_1

As we noted in Section 2, the expressive power of GL is included in MS_2, that is monadic second-order logic of graphs with quantification over sets of edges. Theorem 3.4 shows that this inclusion is proper. Furthermore, since the separation is shown on a class of graphs where the expressive power of MS_1 and MS_2 coincide, this shows that $\mathrm{MS}_1 \not\subseteq \mathrm{GL}$. We now note that, in general, $\mathrm{GL} \not\subseteq \mathrm{MS}_1$. In particular, we show this over the class of labelled graphs with two edge labels.

Recall that in an ordinary undirected graph $G = (V,E)$ a *Hamiltonian cycle* is a cycle that visits every vertex in V exactly once. It is not difficult to write a sentence μ of MS_2 that defines those graphs that contain a Hamiltonian cycle. However it is known that this property is not definable in MS_1, even on ordered graphs (see [7, Cor. 6.3.5]). It is not known whether or not Hamiltonicity of unordered graphs is definable in GL (indeed, this was an open quesion posed in [6]), but we are able to show that in the presence of a second edge label, which acts as a *successor relation*, it is definable. To explain the construction, note that in GL, once we select a set of edges using a composition operator, we are able to say that they form a cycle, but we cannot say in the subformula that the cycle visits all vertices, since some may have been lost in the decomposition. The presence of the successor relation allows us to assert that all vertices are still present.

Theorem 6.1. *GL is not included in* MS_1.

Proof. We consider a vocabulary with two edge labels S and E and two constants s and t. We restrict ourselves to graphs in which the S edges form a simple path from s to t. This condition is easily expressed by a GL sentence succ. Now, let cycle be the GL sentence that defines the graphs in which the E-edges form a simple cycle. Then the following sentence

$$(\mathsf{succ} \wedge \mathsf{cycle} \wedge \forall x[\exists y(S(y,x) \vee S(x,y)) \rightarrow \exists y(E(y,x) \vee E(x,y))] \mid \forall x, y \neg S(x,y))$$

defines the class of such graphs that contain a Hamiltonian cycle.

References

1. Brochenin, R., Demri, S., Lozes, É.: On the almighty wand. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 323–338. Springer, Heidelberg (2008)
2. Cardelli, L., Gardner, P., Ghelli, G.: A spatial logic for querying graphs. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 597–610. Springer, Heidelberg (2002)
3. Compton, K.J.: A logical approach to asymptotic combinatorics II: Monadic second-order properties. J. Comb. Theory, Ser. A 50(1), 110–131 (1989)
4. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Rozenberg, G. (ed.) Handbook of Graph Grammars, pp. 313–400. World Scientific, Singapore (1997)
5. Dawar, A., Gardner, P., Ghelli, G.: Adjunct elimination through games in static ambient logic*(Extended abstract)*. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 211–223. Springer, Heidelberg (2004)
6. Dawar, A., Gardner, P., Ghelli, G.: Expressiveness and complexity of graph logic. Inf. Comput. 205(3), 263–310 (2007)
7. Ebbinghaus, H.-D., Flum, J.: Finite Model Theory, 2nd edn. Springer, Heidelberg (1999)
8. Grädel, E., Hirsch, C., Otto, M.: Back and forth between guarded and modal logics. ACM Trans. Comput. Log. 3(3), 418–463 (2002)
9. Marcinkowski, J.: On the expressive power of graph logic. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 486–500. Springer, Heidelberg (2006)
10. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS, pp. 55–74. IEEE Computer Society, Los Alamitos (2002)