Typestate Checking and Regular Graph Constraints

Viktor Kuncak and Martin Rinard
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
{vkuncak,rinard}@lcs.mit.edu

MIT-LCS-TR-863, September 2002

Abstract

We introduce *regular graph constraints* and explore their decidability properties. The motivation for regular graph constraints is 1) type checking of changing types of objects in the presence of linked data structures, 2) shape analysis techniques, and 3) generalization of similar constraints over trees and grids.

Typestate checking for recursive and potentially cyclic data structures requires verifying the validity of implication for regular graph constraints. The implication of regular graph constraints also arises in shape analysis algorithms such as role-analysis and some analyses based on three-valued logic.

Over the class of lists regular graph constraints reduce to a nondeterministic finite state automaton as a special case. Over the class of trees the constraints reduce to a nondeterministic top-down tree automaton, and over the class of grids our constraints reduce to domino system and tiling problems.

We define a subclass of graphs called *heaps* as an abstraction of the data structures that a program constructs during its execution. We show that satisfiability of regular graph constraints over the class of heaps is decidable. However, determining the validity of implication for regular graph constraints over the class of heaps is undecidable. The undecidability of implication is the central result of the paper. The result is somewhat surprising because our simple constraints are strictly less expressive than existential monadic second-order logic over graphs. In the key step of our proof we introduce the class of *corresponder graphs* which mimic solutions of Post correspondence problem instances. We show undecidability by exhibiting a characterization of corresponder graphs in terms of presence and absence of homomorphisms to a finite number of fixed graphs.

The undecidability of implication of regular graph constraints implies that there is no algorithm that will verify that procedure preconditions are met or that the invariants are maintained when these properties are expressed in any specification language at least as expressive as regular graph constraints.

*This research was supported in part by DARPA Contract F33615-00-C-1692, NSF Grant CCR00-86154, NSF Grant CCR00-63513, and the Singapore-MIT Alliance.

Keywords: Type Checking, Shape Analysis, Program Verification, Graph Homomorphism, Post Correspondence Problem, Monadic Second-Order Logic

1 Introduction

Types capture important properties of objects in the program. In an imperative language properties of objects change over time. It is therefore desirable that types capture changing properties of objects. A typestate system is a system where types of objects change over time. A simple typestate system was introduced in [25], more recent examples include [14, 23, 28, 5]. We view typestate as a step towards statically checking properties of objects [17, 8].

One of the difficulties with defining object properties in object-oriented languages is that a property of an object may depend on properties of other objects in the heap. Some systems allow programmers to identify properties of an object x in terms of properties of objects y such that x references y. The idea that important properties of an object x may depend on properties of objects z such that z references x was introduced in the role system [14].

In general, properties of objects may be mutually recursive and the referencing graph of objects may be cyclic. Due to cycles in the heap the least fixpoint solution for the recursive object properties is not acceptable because there is no basis to ground the inductive definitions of these properties. We therefore say that a heap satisfies a set of properties if there *exists* some choice of predicates that satisfy the mutually recursive definitions. The existential quantification over predicates leads to constraints that have the form of existential monadic second-order logic [7]. For a presentation of role analysis and related systems from the perspective of monadic second-order logic, see [15].

In this paper we present a very simple form of constraints that we call *regular graph constraints*. A set of regular graph constraints can be specified by a single graph G. A heap H satisfies the constraints iff there exists a graph homomorphism from H to G. Regular graph constraints abstract the problem of mutually recursive definitions of properties over potentially cyclic graphs. The existential quantification over predicates is modeled in regular graph constraints as the existence of a homomorphism to a given fixed graph. Regular graph constraints are closed under conjunction and in certain cases are closed under disjunction (Section 2.9). Moreover, regular graph constraints generalize the notion of tree automata [27, 3] and domino systems [12], without going all the way to monadic second-order logic for richer domains. In this paper we consider as the domain of interpretation the class of *heaps*. Our notion of heap is an abstraction of garbage collected heap in a programming languages like Java or ML. Heaps contain a "root" node and a "null" node, all nodes are reachable from the root, and all edges are total functions mapping nodes to nodes.

In Section 2.8 we show that there is a simple and efficient algorithm that decides if regular graph constraints have a heap model. This results in a simple sanity test on regular graph constraint specifications that rules out the contradictory specifications.

We next turn to the problem of checking if one set of regular graph constraints implies another set of regular graph constraints over the set of heaps. Our main contribution (Section 3) is the proof that the implication problem is undecidable.

The implication problem of graphs arises in compositional checking of programs if procedure preconditions or postconditions are given as regular graph constraints. In Section 3.5.4 we show that the implication problem also arises when proving that an invariant holds after every program step. These verification problems are therefore undecidable. Our result places limitations on the completeness of systems such as role analysis [14] and shape analysis [20] that use homomorphic images to represent the abstraction of the heap. The undecidability of regular graph constraints means that semantically checking the implication of such homomorphic graph images is undecidable.

A common way of showing the undecidability of problems over graphs is to encode the Turing machine computation histories [22] as a special form of graphs called *grids*. The difficulty with showing the undecidability of implication of regular graph constraints is that regular graph constraints cannot define the subclass of grids among the class of heaps. Indeed, this is why *satisfiability* of regular graph constraints over heaps is *decidable*. To show the *undecidability* of the *implication* of regular graph constraints, we use the constraints on both sides of the implication to restrict the set of possible counterexample models for the implication. For this purpose we introduce a new class of graphs called *corresponder graphs*. Satisfiability of regular graph constraints over corresponder graphs mimics the solution of a Post correspondence problem instance, and is therefore undecidable. We give a method for constructing an implication such that all counterexamples for the validity of implication are corresponder graphs which satisfy a given regular graph constraint. This shows that the validity of the implication is undecidable.

Due to closure under conjunction, the implication is reducible to the equivalence of regular graph constraints. As a result, the equivalence of two regular graph constraints is also undecidable.

2 Regular Graph Constraints

In this section we define the class of graphs considered in this paper as well as its subclasses heaps, trees, lists, grids, and corresponder graphs. We present our regular graph constraints, give several equivalent formulations of the constraints and show that our constraints capture tree automata and domino systems as special cases. We then review some decidability properties, show that satisfiability of regular constraints over heaps is efficiently decidable and state some closure properties of regular graph constraints.

2.1 Preliminaries

If $r \subseteq A \times B$ and $S \subseteq A$, relational image of set S under r is defined as

$$r[S] = \{y \mid x \in S, \langle x, y\rangle \in r\}$$

We use ■ to mark the end of a proof and ♦ to mark the end of an example.

2.2 Graphs

We will be considering the following class of directed graphs in this paper. Our graphs contain two kinds of edges, which we represent by relations s_1 and s_2. These relations may represent fields in an object-oriented program. The constant root represents the root of the graph. We use edges terminating at null to represent partial functions and abstractions of graphs with partial functions.

Definition 1 *A* graph *is a relational structure*

$$G = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root}\rangle$$

where

- *V is a finite set of nodes;*
- $\mathsf{root}, \mathsf{null} \in V$ *are distinct constants,* $\mathsf{root} \neq \mathsf{null}$*;*
- *$s_1, s_2 \subseteq V \times V$ are two kinds of graph edges, such that for all nodes x*

  $$\langle \mathsf{null}, x\rangle \in s_i \text{ iff } x = \mathsf{null}$$

  for $i \in 1, 2$.

We use $\mathcal{G}$ to denote the class of all graphs.

An s_1-successor of a node x is any element of the set $s_1[\{x\}]$, similarly an s_2-successor of x is any element of $s_2[\{x\}]$. Note that there are exactly two edges originating from null. When drawing graphs we never show these two edges.

Definition 2 *A* heap *is a graph $G = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root}\rangle$ where relations s_1 and s_2 are total functions and where for all $x \neq \mathsf{null}$, node x is reachable from root. We use $\mathcal{H}$ to denote the class of all heaps.*

Definition 3 *The in-degree of a node x in a graph is the number of edges terminating at x.*

$$\mathsf{inDegree}(x) = |\{y \mid \exists i\ \langle y, x\rangle \in s_i\}|$$

Definition 4 *A* tree *is a connected acyclic graph such that* $\mathsf{inDegree}(x) \leq 1$ *for every node x.*

Definition 5 *A* list *is a tree with at most one non-null outgoing edge: for every node x, $s_1(x) = \mathsf{null}$ or $s_2(x) = \mathsf{null}$.*

2.3 Graphs as Constraints

A regular constraint on a graph G is a constraint stating that G can be homomorphically mapped to another graph G'.

Definition 6 *We say that a graph G satisfies the constraints given by a graph G', and write $G \to G'$, iff there exists a homomorphism from G to G'.*

A homomorphism between graphs is defined as follows.

Definition 7 *A function $h : V \to V'$ is a homomorphism between graphs*

$$G = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root} \rangle$$

and

$$G' = \langle V', s_1', s_2', \mathsf{null}', \mathsf{root}' \rangle$$

iff all of the following conditions hold:

1. $\langle x, y \rangle \in s_i$ *implies* $\langle h(x), h(y) \rangle \in s_i'$, *for all* $i \in \{1, 2\}$
2. $h(x) = \mathsf{root}'$ *iff* $x = \mathsf{root}$
3. $h(x) = \mathsf{null}'$ *iff* $x = \mathsf{null}$

If there exists a homomorphism from G to G', we call G a model *for G'.*

We can think of a homomorphism $h : V \to V'$ as a coloring of the graph G. The color $h(x)$ of a node x restricts the colors of the s_1-successors of x to the colors in $s_1[\{h(x)\}]$ and the colors of the s_2-successors to the colors in $s_2[\{h(x)\}]$.

Example 8 A graph G can be colored by k colors so that the adjacent nodes have different colors iff G is homomorphic to a complete graph without self-loops,

$$G' = \langle V', s_1', s_2', \mathsf{null}', \mathsf{root}' \rangle$$

with $V' = \{1, \ldots, k\}$, and

$$s_1' = s_2' = \{\langle x', y' \rangle \mid x' \neq y'\}.$$

♦

The identity function is a homomorphism from the graph to itself. Therefore, $G \to G$ for every graph G. The following fundamental property of homomorphisms also holds.

Proposition 9 (Homomorphisms compose) *Let*

$$\begin{aligned} G &= \langle V, s_1, s_2, \mathsf{null}, \mathsf{root} \rangle \\ G' &= \langle V, s_1', s_2', \mathsf{null}', \mathsf{root}' \rangle \\ G'' &= \langle V, s_1'', s_2'', \mathsf{null}'', \mathsf{root}'' \rangle \end{aligned}$$

and let $h : G \to G'$ and $h' : G' \to G''$ be homomorphisms. Then $h_0 : G \to G''$ where $h_0 = h' \circ h$ is also a homomorphism.

A consequence of Proposition 9 is that $\to$ is a transitive relation.

Definition 10 (Satisfiability) *A graph G' is satisfiable over the class of graphs C iff there exists a graph $G \in C$ such that $G \to G'$. Satisfiability problem over the class of graphs C is: given a graph G', determine if G' is satisfiable.*

Definition 11 (Implication) *We say that G_1 implies G_2 over the class of graphs C, and write*

$$G_1 \leadsto_C G_2,$$

iff

$$(H \to G_1) \text{ implies } (H \to G_2)$$

for all graphs $H \in C$.

We will omit C in $\leadsto_C$ if the class of graphs is clear from the context.

The following fact provides a sufficient condition for the graph implication to hold. It is a direct consequence of Proposition 9.

Proposition 12 *Let C be any class of graphs. Let $G \to G'$. Then $G \leadsto_C G'$.*

In Section 3 we show that the implication of graphs is undecidable over the class of heaps.

2.4 Paths

We next state several simple properties of paths that will be useful in Section 3.

Definition 13 (Path) *Let*

$$G = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root} \rangle$$

be a graph and $n \geq 0$. A path *in graph G, denoted $p \in \mathsf{Paths}(G)$ starting at x_0 and terminating at x_n is a sequence of alternating nodes and labels:*

$$p = x_0, l_0, x_1, l_1, \ldots, l_{n-1}, x_n$$

such that $x_0, \ldots, x_n \in V$; $l_0, \ldots, l_{n-1} \in \{1, 2\}$ and $\langle x_i, x_{i+1} \rangle \in s_i$ for all i, $0 \leq i < n$. We define $\mathsf{word}(p) \in \{1, 2\}^*$ by

$$\mathsf{word}(p) = l_0 l_1 \ldots l_{n-1}$$

Definition 14 (Slice) *A path is a* slice *if it starts at* root *and terminates at* null*.*

Definition 15 (Path Image) *Let h be a homomorphism from graph G_0 to graph G and let*

$$p = x_0, l_0, x_1, l_1, \ldots, l_{n-1}, x_n$$

be a path in G_0. Then

$$h[p] = h(x_0), l_0, h(x_1), l_1, \ldots, l_{n-1}, h(x_n)$$

is the image of path p under the homomorphism h.

The following facts are a consequence of the definition of homomorphism.

Proposition 16 *Let h be a homomorphism from graph G_0 to graph G and let p be a path in G_0. Then*

1. *$h[p]$ is a path in G;*
2. *if p is a slice then $h[p]$ is a slice;*
3. $\mathsf{word}(p) = \mathsf{word}(h[p])$.

Definition 17 *Let p be a path in G and e be a regular expression over the alphabet $\{1,2\}$. We write*

$$p \in_p e$$

iff $\mathsf{word}(p)$ *belongs to the language of the regular expression e.*

From Proposition 16 we directly obtain the following fact.

Proposition 18 (Regular Expression Test) *Let $G_0 \to G$. Then if e is any regular expression over the alphabet $\{1,2\}$ such that G_0 contains some slice $p \in_p e$, then G contains some slice $p' \in_p e$.*

Proof. Let p be a slice in G_0 and $\mathsf{word}(p) \in_p e$. By Proposition 16 we have that $h[p]$ is a slice in G and $\mathsf{word}(h[p]) = \mathsf{word}(p) \in_p e$. ■

We will use the contrapositive of Proposition 18 in the proof of Proposition 37 (Section 3).

2.5 Regular Constraints and EMSOL

We can express the property of being homomorphic to a fixed graph G' by an existential monadic second-order logic formula (EMSOL) of a special form.

Let

$$G' = \langle V', s_1', s_2', \mathsf{null}', \mathsf{root}' \rangle$$

be a fixed graph and let $V' = \{x_0', \ldots, x_{k-1}'\}$ with $x_0' = \mathsf{null}'$, $x_1' = \mathsf{root}'$. Then (1) is a formula in EMSOL interpreted over the graph

$$G = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root} \rangle$$

expressing that G is homomorphic to G'. We use the uppercase identifiers $X_0, \ldots, X_{k-1}$ to denote the second order variables. These variables range over the subsets of V. The lowercase identifiers are first-order variables ranging over the elements of V. Notation $X_i(z)$ means that z is an element of the set X_i. The predicate $s_i(x,y)$ means that $\langle x, y \rangle \in s_i$ holds in the graph G. (For the precise definition of monadic second-order logic see e.g. [10], pp28.)

$$\begin{array}{l} \exists X_0, \ldots, X_{k-1}. \\ \quad \mathsf{partit}(X_0, \ldots, X_{k-1}) \land \mathsf{singl}(X_0, \mathsf{null}) \land \mathsf{singl}(X_1, \mathsf{root}) \land \\ \quad \forall x \bigwedge\limits_{0 \le j < k} (X_j(x) \Longrightarrow P_j(x)) \end{array} \qquad (1)$$

where

$$P_j(x) = P_j^1(x) \land P_j^2(x)$$

$$P_j^i(x) = \forall y.\ \ s_i(x,y) \Longrightarrow \bigvee_{\substack{0 \le l < k \\ \langle x_j', x_l' \rangle \in s_i'}} X_l(y)$$

$$\mathsf{singl}(X, y) = (\forall z.\ X(z) \Longleftrightarrow y = z)$$

$$\mathsf{partit}(Y_0, \ldots, Y_{n-1}) =$$

$$\forall x. \bigvee_{0 \le i < n} Y_i(x) \ \land \bigwedge_{0 \le i < j < n} \neg(Y_i(x) \land Y_j(x))$$

Viewing graph as a formula justifies our previous definitions of graph satisfiability and implication. We can similarly talk about the graph conjunction, disjunction etc.

We may increase the ease of expression of some properties by relaxing the form of EMSOL formula (1) without changing the expressive power. The reason is that that the relaxed form can be converted into a normal form that can be described by a graph homomorphism. Let

$$\mathcal{B}_f(B_0, B_1, \ldots, B_{n-1}; A_0, \ldots, A_{m-1})$$

denote an arbitrary propositional combination of formulas $B_0, \ldots, B_{n-1}, A_0, \ldots, A_{m-1}$ in which $B_0, \ldots, B_{n-1}$ occur only negatively. Then every formula of the following form is expressible as a graph constraint.

$$\begin{array}{l} \exists X_0, \ldots, X_{k-1}. \mathsf{singl}(X_0, \mathsf{null}) \land \mathsf{singl}(X_1, \mathsf{root}) \land \\ \quad \forall x \forall y. \quad \mathcal{B}_1 \Big(s_1(x,y); \\ \qquad\qquad X_0(x), \ldots, X_{k-1}(x), \\ \qquad\qquad X_0(y), \ldots, X_{k-1}(y) \Big) \ \land \\ \qquad\qquad \mathcal{B}_2 \Big(s_2(x,y); \\ \qquad\qquad X_0(x), \ldots, X_{k-1}(x), \\ \qquad\qquad X_0(y), \ldots, X_{k-1}(y) \Big) \end{array} \qquad (2)$$

Compared to (1), the form (2) does not require $X_0, \ldots, X_{k-1}$ to form a partition of the set of all nodes, it has the quantifiers from P_j lifted to the topmost level, and allows arbitrary propositional combinations of predicates $X_i(x)$ and $X_i(y)$.

Example 19 The following formula is of the form (2). Let us assume that the formula is interpreted over the class of heaps. The formula states that the node root has in-degree 0.

$$\begin{array}{l} \exists X_0, X_1, X_2.\ \ \mathsf{singl}(X_0, \mathsf{null}) \land \mathsf{singl}(X_1, \mathsf{root}) \land \\ \quad \forall x \forall y.\ \ \neg(X_1(x) \land X_2(x)) \land (X_0(x) \Rightarrow X_2(x)) \land \\ \qquad\qquad s_1(x,y) \Longrightarrow \Big(\ (X_1(x) \Rightarrow X_2(y)) \land \\ \qquad\qquad\qquad\qquad (X_2(x) \Rightarrow X_2(y)) \ \Big) \\ \qquad\qquad s_2(x,y) \Longrightarrow \ \ \neg X_1(y) \end{array}$$

The formula uses the set X_2 that contains null as well as the nodes reachable from root along the s_1 edges. The formula states explicitly that X_1 and X_2 are disjoint. Because s_1 edges from X_2 can only lead to X_2, there are no s_1 edges to root. The constraint that root has no s_2 edges is specified directly, without introducing an auxiliary set of nodes. In general, negation and the implicit absence of constraints are often more convenient to express with a formula than with a graph homomorphisms.

Note that if we replaced the subformula $(X_0(x) \Rightarrow X_2(x))$ with $\neg(X_0(x) \land X_2(x))$ the resulting formula would require the existence of a cycle in the graph.

♦

Proposition 20 *The two families of formulas (1) and (2) denote the same family of sets of graphs.*

Proof. (Sketch)

Given a formula of form (1) we construct a formula of form (2) by transforming

$$X_j(x) \Rightarrow P_j^1(x) \land P_j^2(x)$$

into $X_j(x) \Rightarrow P_j^1(x)$ and $X_j(x) \Rightarrow P_j^2(x)$. This allows us to write constraints on s_1 and s_2 separately. We then lift the universal quantification over y to the top level. The partition constraint is expressible as a formula that has no occurrences of s_i.

Conversely, suppose we are given a formula in form (2) and suppose that the formula holds for a graph G with the set of nodes V. This means that there exist sets $S_1, \ldots, S_{k-1}$ that satisfy $\mathcal{B}_1$ and $\mathcal{B}_2$. We construct a family of sets $T_{i_1,\ldots,i_{k-1}}$ that forms a partition of set V such that every set S_j is expressible as a union of some sets $T_{i_1,\ldots,i_{k-1}}$. Namely, we define

$$T_{i_0,\ldots,i_{k-1}}(y) = S_0^{i_0} \cdots \wedge \cdots S_{k-1}^{i_{k-1}}$$

where $i_0, \ldots, i_{k-1} \in \{0,1\}$ and

$$S^0 = S$$
$$S^1 = V \setminus S$$

This motivates a construction where the second-order variables $X_0, X_1, X_2, \ldots, X_{k-1}$ are replaced with up to $2 + 2^{k-2}$ new variables $X_0, X_1, Y_0, \ldots, Y_{n-1}$. We then express variables $X_2, \ldots, X_{k-1}$ in terms of $X_0, X_1, Y_0, \ldots, Y_{n-1}$, write the boolean combinations $\mathcal{B}_f$ in disjunctive normal form and use the fact that $X_0, X_1, Y_0, \ldots, Y_{n-1}$ denote disjoint sets. As a result, it is possible to write the original formula in form (1). ∎

Note that, over any class containing all heaps, not every EMSOL formula corresponds to a regular graph constraint. This is in contrast with trees [27] and grids [12]. Even first-order logic can express a constraint that the graph is a grid. On the other hand, we have shown in ([13], pp93) that not even constraints stronger than regular graph constraints can express the gridness property.

For a normal form construction using full existential monadic second-order logic see see [21]. For using higher order logic to express heap properties more general than regular graph constraints, see [15].

2.6 Related Systems

In this section we show the relationship of our regular graph constraints with some other systems for defining sets of graphs. We also illustrate that decidability of satisfiability and implication are sensitive to the subclass of the graphs considered, and change in a non-monotonic way.

2.6.1 Words

Regular graph constraints over lists correspond to regular word languages. A regular graph constraint corresponds to a nondeterministic finite state automaton with the initial state root and the final state null.

2.6.2 Trees

Satisfiability and implication of regular graph constraints are decidable over the class of trees. The reason is that the entire MSOL is decidable over trees [27], and regular graph constraints are expressible in MSOL.

2.6.3 Pictures

Domino systems [12] are regular graph constraints over the grids.

Definition 21 *A grid $m \times n$ is a graph isomorphic to*

$$G = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root} \rangle$$

where

$$\begin{array}{rcl} V & = & \{1,\ldots,m\} \times \{1,\ldots,n\} \\ s_1 & = & \{\langle\langle i,j\rangle, \langle i,j+1\rangle\rangle \mid 1 \le i \le m; 1 \le j \le n-1\} \ \cup \\ & & \{\langle\langle i,n\rangle, \mathsf{null}\rangle \mid 1 \le i \le m\} \\ s_2 & = & \{\langle\langle i,j\rangle, \langle i+j,j\rangle\rangle \mid 1 \le i \le m-1; 1 \le j \le n\} \ \cup \\ & & \{\langle\langle j,m\rangle, \mathsf{null}\rangle \mid 1 \le j \le n\} \\ \mathsf{root} & = & \langle 1,1\rangle \end{array}$$

The chapter [12] uses the term *pictures* for grids. It is easy to see that over the domain of grids, regular graph constraint are equivalent to a domino system with s_1 edges denoting horizontal dominoes and s_2 edges denoting vertical dominoes. The graph homomorphism corresponds to the use of projection.

[12] states the equivalence of domino systems over pictures with negation-free regular expressions with projections, on-line tessellation automata, existential monadic second-order formulas and tiling systems.

We view the fact that, over the grids, regular graph constraints are equivalent to each of the systems above as an indication that the definition of regular graph constraints is natural.

Note When comparing our regular graph constraints to trees and domino systems, we notice that in our definition of a model there are no fixed labels associated with nodes. The only labeling of nodes comes from the graph homomorphism, which corresponds to projection in tree and picture languages. Our simplification makes our undecidability result strictly stronger. Furthermore, regular graph constraints can capture the distinction between a node with an edge terminating at null and a node with an edge terminating at a node that is not null. This distinction can be used for encoding in the structure of the graph any fixed labeling of graph nodes.

2.7 Decidability of Implication over Graphs

Satisfiability problem over the class of graphs is trivial. Namely, $G \to G$, so every graph is satisfiable. The implication problem over graphs is also decidable, in contrast to the implication problem over the class of heaps, which we will show undecidable in Section 3.

Proposition 22

$$G_1 \rightsquigarrow_{\mathcal{G}} G_2 \textit{ iff } G_1 \to G_2$$

Proof. Let $G_1 \rightsquigarrow_{\mathcal{G}} G_2$. Because $G_1 \to G_1$, we obtain $G_1 \to G_2$. Conversely, let $G_1 \to G_2$. By Proposition 12 we conclude $G_1 \rightsquigarrow_{\mathcal{G}} G_2$. ∎

Our regular graph constraints are weaker than finite graph acceptors of [26]. It is easy to see that finite graph acceptors can define the gridness property. Therefore, domino system satisfiability is reducible to satisfiability of finite graph acceptors, which makes finite graph acceptor satisfiability undecidable.

GraphCleanup: Repeat the following operations until the graph stabilizes:

1. remove an unreachable node
2. remove a node x such that $s_1[\{x\}] = \emptyset$ or $s_2[\{x\}] = \emptyset$

mark(x):

1. if x is marked then return, otherwise:
2. select(x);
3. pick a s_1-successor y of x; select($\langle x, y\rangle$); mark(y)
4. pick a s_2-successor z of x; select($\langle x, y\rangle$); mark(z)

SatisfiabilityCheck: Repeat the following operations until the graph stabilizes:

1. perform GraphCleanup;
2. if the resulting graph is empty, then G' is unsatisfiable;
3. otherwise a heap satisfying G' can be obtained as follows:
4. let all graph nodes be unmarked;
5. mark(root);
6. return subgraph containing selected nodes

Figure 1: Satisfiability check for Heaps

2.8 Satisfiability over Heaps

We show that satisfiability for heaps is efficiently decidable by the nondeterministic algorithm in Figure 1. The goal of the algorithm is to find, given a graph G', whether there exists a heap G such that $G \to G'$. Recall that the property of a heap is that every node has exactly one s_1 outgoing edge and exactly one s_2 outgoing edge. This property need not be satisfied by G', so we cannot take $G = G'$ to be the heap proving satisfiability of G'. The algorithm updates the current graph until it becomes a heap or an empty graph. (For the purpose of this algorithm we allow even null and root to be removed from the graph.) If nonempty, the result is a heap G such that $G \to G'$.

Proposition 23 *The procedure in Figure 1 is a correct algorithm for determining satisfiability of a graph over the class of heaps.*

Proof. The procedure consists of two parts: GraphCleanup and SatisfiabilityCheck. The GraphCleanup part eliminates useless nodes and determines whether there exists a heap H such that $H \to G'$. Graph Cleanup terminates because it decreases the size of the current graph G in every step. The mark phase terminates because it does a simple breadth-first search.

Observe that GraphCleanup does not reduce the set of heaps homomorphic to G'. Namely, if a node x of G is removed in GraphCleanup, then no node is mapped to x under any homomorphism h. Therefore, if GraphCleanup returns an empty graph, then G' is unsatisfiable.

Assume that GraphCleanup returns a nonempty graph G. Then G contains root and every node in x has a s_1-successor and a s_2-successor, but some of the nodes may have two s_1-successors or s_2-successors. Invoking mark will do a depth-first search on G and pick a subgraph H where every node has exactly one s_1-successor and one s_2-successor. The resulting graph H will therefore be a heap. We have $H \to G'$ because H is a subgraph of G'. ■

2.9 Closure Properties

In this section we give a construction for computing the conjunction of two graphs and a construction for computing the disjunction of two graphs. We will use these constructions in Section 3.

2.9.1 Conjunction

We show how to use a Cartesian product construction to obtain a conjunction of two graphs G_1 and G_2.

Definition 24 (Cartesian Product) *Let*

$$G^1 = \langle V^1, s_1^1, s_2^1, \mathsf{null}^1, \mathsf{root}^1\rangle$$
$$G^2 = \langle V^2, s_1^2, s_2^2, \mathsf{null}^2, \mathsf{root}^2\rangle$$

be graphs. Then $G^0 = G^1 \times G^2$ is the graph

$$G^0 = \langle V^0, s_1^0, s_2^0, \mathsf{null}^0, \mathsf{root}^0\rangle$$

such that:

$$\begin{aligned}
\mathsf{null}^0 &= \langle \mathsf{null}^1, \mathsf{null}^2\rangle \\
\mathsf{root}^0 &= \langle \mathsf{root}^1, \mathsf{root}^2\rangle \\
V^0 &= \{\mathsf{null}^0, \mathsf{root}^0\} \\
&\cup\ (V^1 \setminus \{\mathsf{null}^1, \mathsf{root}^1\}) \times (V^2 \setminus \{\mathsf{null}^2, \mathsf{root}^2\}) \\
s_i^0 &= \{\langle\langle x^1, x^2\rangle, \langle y^1, y^2\rangle\rangle \mid \langle x^1, y^1\rangle \in s_i^1; \langle x^2, y^2\rangle \in s_i^2\}, \\
& \quad i \in \{1, 2\}
\end{aligned}$$

Proposition 25 (Conjunction via Product) *For every graph G,*

$$G \to G_1 \times G_2 \quad \textit{iff} \quad G \to G_1 \textit{ and } G \to G_2$$

In other words, $G_1 \times G_2$ is a conjunction of G_1 and G_2.

Proof. $(\Longrightarrow)$: Let $G \to G^1$ and $G \to G^2$ with $h^1 : V \to V^1$ and $h^2 : V \to V^2$ where G^1 and G^2 are defined as in Definition 24 and

$$G = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root}\rangle$$

Let $h^0 = h^1 \times h^2$ where

$$(h^1 \times h^2)(x) = \langle h^1(x), h^2(x)\rangle$$

We claim that h^0 is a homomorphism from G to $G^0 = G^1 \times G^2$. It is straightforward to verify that properties 2 and 3 of homomorphism hold for h^0. Let $i \in \{1, 2\}$. If $\langle x, y\rangle \in s_i$ then $\langle h^1(x), h^1(y)\rangle \in s_i^1$ and $\langle h^2(x), h^2(y)\rangle \in s_i^2$. Because h^1 and h^2 satisfy properties 2 and 3 of homomorphism, we have $\langle h^1(x), h^2(x)\rangle, \langle h^2(y), h^2(y)\rangle \in V^0$ regardless of whether $x, y \in \{\mathsf{null}, \mathsf{root}\}$ or not. We therefore conclude $\langle\langle h^1(x), h^2(x)\rangle, \langle h^1(x), h^2(y)\rangle\rangle \in s_i^0$ by the definition of s_i^0. Hence, h^0 is a homomorphism and $G \to G^0$.

($\Longleftarrow$) : Let $G \to G^0$ with $h^0 : V \to V^0$. Define $h^1 = \pi_1 \circ h$ and $h^2 = \pi_2 \circ h$. Let's show that h^1 is a homomorphism (an analogous argument holds for h_2). It is straightforward to see that properties 2 and 3 hold for h_1. For property 1, let $i \in \{1,2\}$ and let $\langle x, y\rangle \in s_i$. Let $h^0(x) = \langle x^1, x^2\rangle$ and $h^0(y) = \langle y^1, y^2\rangle$. Because h^0 is a homomorphism, we have

$$\langle\langle x^1, x^2\rangle, \langle y^1, y^2\rangle\rangle \in s_i^0$$

By definition of s_i^0 we conclude $\langle x^1, y^1\rangle \in s_i^1$ which by definition of h^1 means $\langle h^1(x), h^1(y)\rangle \in s_i^1$. ■

2.9.2 Disjunction

Given our definition of graphs, there is no construction that would yield disjunction of arbitrary graphs over the family that contains all heaps. We illustrate this fact with an example. We then give a simple condition on graphs that ensures that the disjunction construction is possible over the domain of heaps.

Example 26 Let

$$\begin{array}{rcl} G^1 & = & \langle\{\mathsf{root}, \mathsf{null}\}, s_1^1, s_2^2, \mathsf{root}, \mathsf{null}\rangle \\ s_1^1 & = & \{\langle \mathsf{root}, \mathsf{null}\rangle\} \\ s_2^1 & = & \{\langle \mathsf{root}, \mathsf{root}\rangle\} \end{array}$$

and

$$\begin{array}{rcl} G^2 & = & \langle\{\mathsf{root}, \mathsf{null}\}, s_1^2, s_2^2, \mathsf{root}, \mathsf{null}\rangle \\ s_1^2 & = & \{\langle \mathsf{root}, \mathsf{root}\rangle\} \\ s_2^2 & = & \{\langle \mathsf{root}, \mathsf{null}\rangle\} \end{array}$$

In the class of heaps, the only model for G^1 is G^1 itself, and the only model of G^2 is G^2. Now assume that there exists graph

$$G^0 = \langle V^0, s_1^0, s_2^0, \mathsf{null}, \mathsf{root}\rangle$$

such that $G^1 \to G^0$ and $G^2 \to G^0$. From $G^1 \to G^0$ we conclude

$$\langle \mathsf{root}, \mathsf{null}\rangle \in s_1^0$$

and from $G^2 \to G^0$ we conclude

$$\langle \mathsf{root}, \mathsf{null}\rangle \in s_2^0$$

Therefore for the graph

$$\begin{array}{rcl} G^3 & = & \langle\{\mathsf{root}, \mathsf{null}\}, s_1^3, s_2^3, \mathsf{root}, \mathsf{null}\rangle \\ s_1^3 & = & \{\langle \mathsf{root}, \mathsf{null}\rangle\} \\ s_2^3 & = & \{\langle \mathsf{root}, \mathsf{null}\rangle\} \end{array}$$

we have $G_3 \to G^0$ as well. So there is no graph G^0 such that for all heaps G,

$$(G \to G^0) \text{ iff } ((G \to G^1) \text{ or } (G \to G^2))$$

♦

To ensure that we can find union graphs over the set of heaps, we will require $s_2(\mathsf{root}) = \mathsf{null}$.

Definition 27 (Orable Graphs) *A graph*

$$G = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root}\rangle$$

is orable *iff for all $x \in V$,*

$$\langle \mathsf{root}, x\rangle \in s_2 \quad \textit{iff} \quad x = \mathsf{null}$$

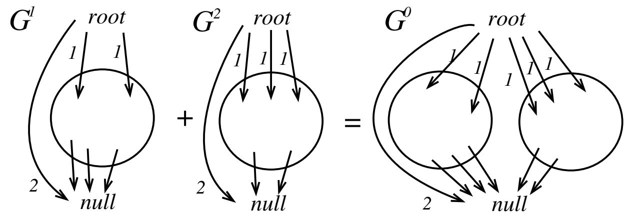

Figure 2: Graph Sum

Definition 28 (Graph Sum) *Let*

$$\begin{array}{l} G^1 = \langle V^1, s_1^1, s_2^1, \mathsf{null}, \mathsf{root}\rangle \\ G^2 = \langle V^2, s_1^2, s_2^2, \mathsf{null}, \mathsf{root}\rangle \end{array}$$

be orable graphs such that $V^1 \cap V^2 = \{\mathsf{null}, \mathsf{root}\}$. Then $G^0 = G^1 + G^2$ is the graph

$$G^0 = \langle V^0, s_1^0, s_2^0, \mathsf{null}, \mathsf{root}\rangle$$

where

$$\begin{array}{rcl} V^0 & = & V^1 \cup V^2 \\ s_1^0 & = & s_1^1 \cup s_2^2 \\ s_2^0 & = & s_2^1 \cup s_2^2 \end{array}$$

The following simple fact allows us to form arbitrary finite sums of orable graphs.

Proposition 29 *If G^1 and G^2 are orable graphs, then $G^1 + G^2$ is also orable.*

Proposition 30 (Disjunction via Sum) *Let G be a heap and G^1 and G^2 be orable graphs. Then*

$$G \to G^1 + G^2 \quad \textit{iff} \quad G \to G^1 \textit{ or } G \to G^2$$

Proof. ($\Longleftarrow$) : Assume without loss of generality $G \to G^1$. Because G^1 is a subgraph of $G^1 + G^2$, there exists an identity homomorphism from G^1 into $G^1 + G^2$. By Proposition 9 we conclude $G \to G^1 + G^2$.

($\Longrightarrow$) : Let $G^0 = G^1 + G^2$ where G^1 and G^2 are as as in Definition 28, and $G \to G^0$ with a homomorphism $h : V \to V^0$ where

$$G = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root}\rangle$$

We claim

$$h[V] \subseteq V^1 \quad \text{or} \quad h[V] \subseteq V^2 \tag{3}$$

Suppose the claim does not hold. Then there exist $x^0, y^0 \in V$ such that

$$\begin{array}{rcl} x^1 & \in & V^1 \setminus V^2 \\ y^2 & \in & V^2 \setminus V^1 \end{array}$$

where $x^1 = h(x^0)$ and $y^2 = h(y^0)$. By definition of heap, there exists a sequence of nodes $p = \mathsf{root}, z^0, \ldots, x^0$ forming a path from root to x^0 in G. Because $x^0 \notin \{\mathsf{null}, \mathsf{root}\}$, the path has length at least two and $z^0 \notin \{\mathsf{null}, \mathsf{root}\}$ (it may or may not be $z^0 = x^0$). Because $G^1 + G^2$ is orable, the edge from root to z^0 cannot be from s_2, so $s_1(\mathsf{root}) = z^0$. We claim $h(z^0) \in V^1 \setminus V^2$. Indeed, suppose $h(z^0) \in V^2$. By the properties of homomorphism and because in $G^1 + G^2$ there are no edges from nodes $V^1 \setminus \{\mathsf{null}, \mathsf{root}\}$ to nodes $V^2 \setminus \{\mathsf{null}, \mathsf{root}\}$, we have $h(w) \in V^2$ for every node w of the path p. This is a contradiction with $h(x^0) \in V^1 \setminus V^2$. We conclude

$$z^0 \in V^1 \setminus V^2$$

Repeating an analogous argument for node y^0, we conclude

$$z^0 \in V^2 \setminus V^1$$

because $z^0 = s_1(\mathsf{root})$ is the unique s_1-successor of root in the heap G. We have arrived at contradiction, so (3) is true. If $h[V] \subseteq V^1$ then $G \to G^1$ and if $h[V] \subseteq V^2$ then $G \to G^2$. ■

A product of orable graphs is orable.

Proposition 31 *Let G^1 and G^2 be orable graphs. Then $G^1 \times G^2$ is orable.*

In the sequel we will deal only with orable graphs.

3 Undecidability of Implication

This section presents the central result of this paper: *The implication of graphs is undecidable over the class of heaps.* Our proof proceeds in two steps. We first introduce a family of *corresponder graphs*. We show that satisfiability of graphs over the family of corresponder graphs is undecidable.

In the second step we show that satisfiability over corresponder graphs can be reduced the question of whether an implication between two graphs fails to hold. The key to the construction in the second step is that a conjunction of certain regular graph constraints and negations of regular graph constraints can precisely characterize the class of corresponder graphs.

3.1 Corresponder Graphs

Corresponder graphs are a subclass of the class of heaps. Figure 3 shows an example corresponder graph.

Definition 32 *Let $k \geq 2$, $n \geq 2$, and*

$$\begin{array}{ccccccccccc} 0 & = & u_0 & < & u_1 & < & \ldots & < & u_{k-1} & < & n \\ 0 & = & l_0 & < & l_1 & < & \ldots & < & l_{k-1} & < & n \end{array}$$

A corresponder graph

$$\mathsf{CG}(n, k, u_1, \ldots, u_{k-1}, l_1, \ldots, l_{k-1})$$

is a graph isomorphic to

$$G = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root}\rangle$$

where

$$\begin{array}{rcl} V & = & \{\mathsf{null}, \mathsf{root}\} \\ & \cup & \{C_0, C_1, \ldots, C_{2k-1}\} \\ & \cup & \{U_0, U_1, \ldots, U_{2n-1}\} \\ & \cup & \{L_0, L_1, \ldots, L_{2n-1}\} \\ s_1 & = & \{\langle \mathsf{root}, C_0\rangle\} \\ & \cup & \{\langle C_i, C_{i+1}\rangle \mid 0 \leq i < 2k-1\} \\ & \cup & \{\langle C_{2k-1}, \mathsf{null}\rangle\} \\ & \cup & \{\langle U_i, U_{i+1}\rangle \mid 0 \leq i < 2n-1\} \\ & \cup & \{\langle U_{2n-1}, \mathsf{null}\rangle\} \\ & \cup & \{\langle L_i, L_{i+1}\rangle \mid 0 \leq i < 2n-1\} \\ & \cup & \{\langle L_{2n-1}, \mathsf{null}\rangle\} \\ s_2 & = & \{\langle \mathsf{root}, \mathsf{null}\rangle\} \\ & \cup & \{\langle C_{2i}, U_{2u_i}\rangle \mid 0 \leq i < k\} \\ & \cup & \{\langle C_{2i+1}, L_{2l_i+1}\rangle \mid 0 \leq i < k\} \\ & \cup & \{\langle U_{2i}, L_{2i}\rangle \mid 0 \leq i < n\} \\ & \cup & \{\langle L_{2i+1}, U_{2i+1}\rangle \mid 0 \leq i < n\} \\ & \cup & \left\{\langle U_{2i+1}, \mathsf{null}\rangle \mid i \in \{0, \ldots, n-1\} \setminus \{l_0, \ldots, l_{k-1}\}\right\} \\ & \cup & \left\{\langle U_{2i+1}, \mathsf{root}\rangle \mid i \in \{l_0, \ldots, l_{k-1}\}\right\} \\ & \cup & \left\{\langle L_{2i}, \mathsf{null}\rangle \mid i \in \{0, \ldots, n-1\} \setminus \{u_0, \ldots, u_{k-1}\}\right\} \\ & \cup & \left\{\langle L_{2i}, \mathsf{root}\rangle \mid i \in \{u_0, \ldots, u_{k-1}\}\right\} \end{array}$$

The family CG *is the union of all corresponder graphs* $\mathsf{CG}(n, k, u_1, \ldots, u_{k-1}, l_1, \ldots, l_{k-1})$.

3.2 Corresponder Graph Satisfiability

For completeness we define the Post correspondence problem, PCP ([22], pp183).

Definition 33 *A PCP* instance *is a sequence of pairs of nonempty words:*

$$\langle v_0, w_0\rangle, \langle v_1, w_1\rangle, \ldots, \langle v_{m-1}, w_{m-1}\rangle$$

A solution *for a PCP instance is a sequence*

$$t_0, t_1, \ldots, t_{k-1}$$

such that

$$v_{t_0} v_{t_1} \ldots v_{t_{k-1}} = w_{t_0} w_{t_1} \ldots w_{t_{k-1}}$$

[22] contains the proof for the following theorem.

Theorem 34 *The following problem is undecidable: given a PCP instance, does it have a solution.*

We will use the following proposition to establish undecidability of graph implication.

Proposition 35 *Satisfiability of graphs over the class of corresponder graphs is undecidable.*

Proof. We give a reduction from PCP. Let $m \geq 2$ and let

$$\langle v_0, w_0\rangle, \langle v_1, w_1\rangle, \ldots, \langle v_{m-1}, w_{m-1}\rangle$$

be an instance of PCP where v_i, w_i are nonempty words. Introduce names a_i^j and b_i^j for letters in words v_i, w_i:

$$\begin{array}{rclc} v_i & = & v_i^0 v_i^1 \ldots v_i^{p_i-1} & 0 \leq i \leq m-1 \\ w_i & = & w_i^0 w_i^1 \ldots w_i^{q_i-1} & 0 \leq i \leq m-1 \end{array}$$

where $p_i = |v_i|$ and $q_i = |w_i|$. We construct a graph G such that there exists a corresponder graph G_0 with the property $G_0 \to G$ iff the PCP instance has a solution.

Figure 4 illustrates how a corresponder graph G_0 with a homomorphism from G_0 to G encodes a solution of the PCP instance $\langle c, bc\rangle, \langle ab, a\rangle$.

Let

$$G = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root}\rangle$$

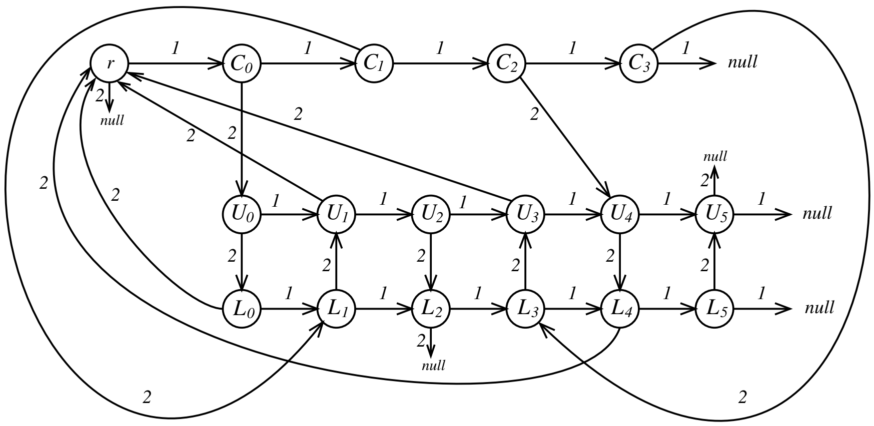

Figure 3: An Example Corresponder Graph ($k = 2$, $n = 3$)

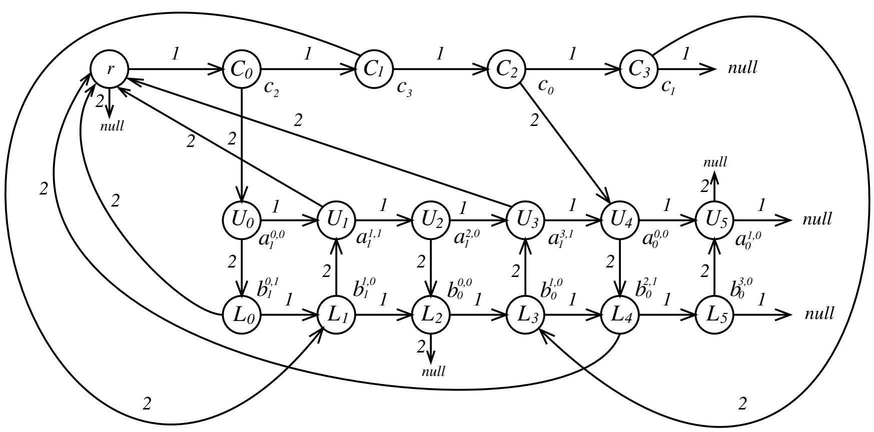

Figure 4: Corresponder Graph with a Homomorphism Encoding a Solution of $\langle c, bc\rangle, \langle ab, a\rangle$

Define the components of G are as follows. For every pair of words $\langle v_i, w_i\rangle$ introduce two nodes $c_{2i}, c_{2i+1} \in V$. These nodes will summarize C-nodes of a corresponder graph. For every position v_i^j of the word v_i introduce nodes $a_i^{2j,0}$ and $a_i^{2j+1,0}$ and for every position w_i^j introduce nodes $b_i^{2j,0}$ and $b_i^{2j+1,0}$. The a-nodes will summarize U-nodes and the b-nodes will summarize the L-nodes of the corresponder graph. Introduce also the additional nodes $b_i^{2j,1}$ to encode the information that $a_i^{2j,0}$ node has an incoming edge from a c-node. As we will see below, the $b_i^{2j,1}$ nodes have s_2 pointing to root as opposed to null, which ensures that every $a_i^{0,0}$ node has an incoming edge from a c-node. For analogous reasons we introduce $a_i^{2j+1,1}$ nodes. Let

$$\begin{array}{rcl} V & = & \{\mathsf{null}, \mathsf{root}\} \\ & \cup & \{c_0, c_1, \ldots, c_{2m-1}\} \\ & \cup & \{a_i^{j,0} \mid 0 \le i < m; 0 \le j < 2p_i\} \\ & \cup & \{b_i^{j,0} \mid 0 \le i < m; 0 \le j < 2q_i\} \\ & \cup & \{b_i^{2j,1} \mid 0 \le i < m; 0 \le j < q_i\} \\ & \cup & \{a_i^{2j+1,1} \mid 0 \le i < m; 0 \le j < p_i\} \end{array}$$

Define s_1 graph edges as follows.

The c_i nodes are connected into a list that begins with root and every c_{2i} is followed by c_{2i+1}. The pairs c_{2i}, c_{2i+1} for different i can repeat in the list any number of times and in arbitrary order. This list will encode a PCP instance solution.

The nodes representing word positions are linked in the order in which they appear in the word. The last position in a word can be followed by the first position of any other word, or by null. The nodes for the v_i words and the nodes for the w_i words form disjoint lists along the s_1 edges.

$$\begin{array}{l} s_1 = \{\langle \mathsf{root}, c_{2i}\rangle \mid 0 \le i < m\} \\ \quad \cup \{\langle c_{2i}, c_{2i+1}\rangle \mid 0 \le i < m\} \\ \quad \cup \{\langle c_{2i+1}, c_{2j}\rangle \mid 0 \le i, j < m\} \\ \quad \cup \{\langle c_{2i+1}, \mathsf{null}\rangle \mid 0 \le i < m\} \\ \quad \cup \{\langle a_i^{2j,0}, a_i^{2j+1,\alpha}\rangle \mid 0 \le i < m; 0 \le j < p_i; \alpha \in \{0,1\}\} \\ \quad \cup \{\langle a_i^{2j+1,\alpha}, a_i^{2j+2,0}\rangle \mid 0 \le i < m; 0 \le j < p_i - 1; \\ \qquad\qquad\qquad \alpha \in \{0,1\}\} \\ \quad \cup \{\langle a_i^{2p_i-1,\alpha}, a_j^{0,0}\rangle \mid 0 \le i, j < m; \alpha \in \{0,1\}\} \\ \quad \cup \{\langle a_i^{2p_i-1,\alpha}, \mathsf{null}\rangle \mid 0 \le i < m; \alpha \in \{0,1\}\} \\ \quad \cup \{\langle b_i^{2j,\alpha}, b_i^{2j+1,0}\rangle \mid 0 \le i < m; 0 \le j < q_i; \alpha \in \{0,1\}\} \\ \quad \cup \{\langle b_i^{2j+1,0}, b_i^{2j+2,\alpha}\rangle \mid 0 \le i < m; 0 \le j < q_i - 1; \\ \qquad\qquad\qquad \alpha \in \{0,1\}\} \\ \quad \cup \{\langle b_i^{2q_i-1,0}, b_j^{0,\alpha}\rangle \mid 0 \le i, j < m; \alpha \in \{0,1\}\} \\ \quad \cup \{\langle b_i^{2q_i-1,0}, \mathsf{null}\rangle \mid 0 \le i < m\} \end{array}$$

We define s_2 graph edges as follows.

Every c_j edge points to the position at the beginning of the word. Even numbered nodes point to the a^0-positions; odd numbered nodes point to b^1-positions.

The a_i and b_j word positions are connected so that an a-node points to a b-node for even indices, whereas a b-node points to an a-node for odd indices.

$$\begin{array}{l} s_2 = \{\langle \mathsf{root}, \mathsf{null}\rangle\} \\ \quad \cup \{\langle c_{2i}, a_i^{0,0}\rangle \mid 0 \le i < m\} \\ \quad \cup \{\langle c_{2i+1}, b_i^{1,0}\rangle \mid 0 \le i < m\} \\ \quad \cup \{\langle a_i^{0,0}, b_k^{2l,1}\rangle \mid 0 \le i, k < m; 0 \le l < q_k; v_i^0 = w_k^l\} \\ \quad \cup \{\langle a_i^{2j,0}, b_k^{2l,0}\rangle \mid 0 \le i, k < m; 0 < j < p_i; 0 \le l < q_k; \\ \qquad\qquad\qquad v_i^j = w_k^l\} \\ \quad \cup \{\langle b_k^{2l,0}, \mathsf{null}\rangle \mid 0 \le k < m; 0 \le l < q_k\} \\ \quad \cup \{\langle b_k^{2l,1}, \mathsf{root}\rangle \mid 0 \le k < m; 0 \le l < q_k\} \\ \quad \cup \{\langle b_k^{1,0}, a_i^{2j+1,1}\rangle \mid 0 \le i, k < m; 0 \le j < p_i; v_i^j = w_k^l\} \\ \quad \cup \{\langle b_k^{2l+1,0}, a_i^{2j+1,0}\rangle \mid 0 \le i, k < m; 0 \le j < p_i; \\ \qquad\qquad\qquad 0 < l < q_k; v_i^j = w_k^l\} \\ \quad \cup \{\langle a_i^{2j+1,0}, \mathsf{null}\rangle \mid 0 \le i < m; 0 \le j < p_i\} \\ \quad \cup \{\langle a_i^{2j+1,1}, \mathsf{root}\rangle \mid 0 \le i < m; 0 \le j < p_i\} \end{array}$$

This completes the definition of G.

Claim 36 *The PCP instance has a solution iff there exists a corresponder graph G_0 such that $G_0 \to G$.*

$(\Longrightarrow)$: Assume that the PCP instance has a solution $t_0, t_1, \ldots, t_{k-1}$. Then

$$v_{t_0} v_{t_1} \ldots v_{t_{k-1}} = w_{t_0} w_{t_1} \ldots w_{t_{k-1}}$$

Let $u_0 = l_0 = 0$,

$$\begin{array}{rcll} n & = & \sum_{j=0}^{k-1} |v_{t_j}| & \\ u_{i+1} & = & \sum_{j=0}^{i} |v_{t_j}|, & 0 \le i < k-1 \\ l_{i+1} & = & \sum_{j=0}^{i} |w_{t_j}|, & 0 \le i < k-1 \end{array}$$

and let

$$G_0 = \mathsf{CG}(n, k, u_1, \ldots, u_{k-1}, l_1, \ldots, l_{k-1})$$

be a corresponder graph. We construct a homomorphism h from G_0 to G as follows. We map C-nodes of G_0 into c-nodes of G:

$$\begin{array}{rcll} h(C_{2j}) & = & c_{2t_j}, & 0 \le j < k \\ h(C_{2j+1}) & = & c_{2t_j+1}, & 0 \le j < k \end{array}$$

For $0 \le f < n$, let $d_u(f)$ denote the largest index i such that $u_i \le f$. We map the U-nodes into a nodes as follows. Consider a node U_{2f}. Let $i = d(f)$. Then U_{2f} is the even node that represents the letter $v_i^{f-u_i}$ of the word v_i:

$$h(U_{2f}) = a_i^{2(f-u_i),0}, \qquad i = d_u(f),\ 0 \le f < n$$

The mapping of U_{2f+1} is similar. In this case we also encode the information whether L_{2f+1} has an s_2-edge from a C-node.

$$U_{2f+1} = a_i^{2(f-u_i)+1,\alpha_l(f)}, \qquad i = d_u(f),\ 0 \leq f < n$$

where

$$\alpha_l(f) = \begin{cases} 1, & f \in \{l_0, l_1, \ldots, l_{k-1}\} \\ 0, & \text{otherwise} \end{cases}$$

The mapping of L-nodes is analogous. Let $d_l(f)$ denote the largest index i such that $l_i \leq f$. Then

$$L_{2f+1} = b_i^{2(f-u_i)+1,0}, \qquad i = d_u(f),\ 0 \leq f < n$$

$$U_{2f} = b_i^{2(f-u_i),\alpha_u(f)}, \qquad i = d_u(f),\ 0 \leq f < n$$

where

$$\alpha_u(f) = \begin{cases} 1, & f \in \{u_0, u_1, \ldots, u_{k-1}\} \\ 0, & \text{otherwise} \end{cases}$$

It is straightforward to verify that h is indeed a homomorphism.

($\Longleftarrow$) : Assume that $G_0 \to G$ where

$$G_0 = \mathsf{CG}(n, k, u_1, \ldots, u_{k-1}, l_1, \ldots, l_{k-1})$$

is a corresponder graph and h is a homomorphism from G_0 to G. Because in graph G all paths given by the regular expression 1^* lead to c_i-nodes or null, we conclude that each C_j node is mapped to some c_i node. For $0 \leq j < k$ we define

$$t_j = i \quad \text{iff} \quad h(C_{2j}) = c_{2i}$$

From the properties of homomorphism we derive

$$t_j = i \quad \text{iff} \quad h(C_{2j+1}) = c_{2i+1}$$

We will show that t_j is a solution of the PCP instance. Let $u_0 = l_0 = 0$ and $u_{n+1} = l_{n+1} = n$. Let

$$\tau(a_i^{2j,0}) = \tau(a_i^{2j+1,\alpha}) = v_i^j$$

$$\tau(b_i^{2j,\alpha}) = \tau(b_i^{2j+1,0}) = w_i^j$$

and $h' = \tau \circ h$. By construction of s_2 in G we have

$$h'(U_{2j}) = h'(U_{2j+1}) = h'(L_{2j}) = h'(L_{2j+1})$$

for $0 \leq j < n$. To prove

$$v_{t_0} v_{t_1} \ldots v_{t_{k-1}} = w_{t_0} w_{t_1} \ldots w_{t_{k-1}}$$

it therefore suffices to show

$$v_{t_j} = h'(U_{2u_j}) h'(U_{2(u_j+1)}) \ldots h'(U_{2(u_{j+1}-1)}) \qquad (4)$$

$$w_{t_j} = h'(L_{2l_j+1}) h'(L_{2(l_j+1)+1}) \ldots h'(L_{2(l_{j+1}-1)+1}) \qquad (5)$$

for $0 \leq j < k$. Let $t_j = i$. Then $h(C_{2j}) = c_{2i}$. We have $\langle C_{2j}, U_{2u_j} \rangle \in s_2$ in the corresponder graph G_0. On the other hand, $\langle c_{2i}, a_i^{0,0} \rangle \in s_2$ is the only s_2-outgoing edge of c_{2i} in G. Therefore, $h(U_{2u_j}) = a_i^{0,0}$. From this, we first conclude so $h'(U_{2u_j}) = v_i^0$. Next, by construction of s_1 edges of G and G_0 we get

$$h(U_{2(u_j+1)}) = a_i^{2,0}$$

$$\ldots$$

$$h(U_{2(u_j+p_i-1)}) = a_i^{2(p_i-1),0}$$

To establish (4) it suffices to show $u_j + p_i = u_{j+1}$. To see that the equality holds, suppose first $u_j + p_i > u_{j+1}$. Then $h(U_{2u_{j+1}}) = a_i^{2f,0}$ where $f > 0$. Because h is a homomorphism, following s_2 edge once we conclude $h(L_{2u_{j+1}}) = b_g^{2r,0}$ for some g and following s_2 for the second time we obtain the contradiction because in corresponder graph $s_2(L_{2u_{j+1}}) = \mathsf{root}$ but in G we have $s_2(b_g^{2r,0}) = \mathsf{null}$. Similarly, suppose now $u_j + p_i < u_{j+1}$. Then $s_1(U_{2(u_j+p_i-1)}) \neq \mathsf{null}$, so let $U_{2(u_j+p_i)} = s_1(s_1(U_{2(u_j+p_i-1)}))$. Because $u_j + p_i \notin \{u_0, \ldots, u_{k-1}\}$, we get

$$s_2(s_2(U_{2(u_j+p_i)})) = \mathsf{null}$$

On the other hand, $h(U_{2(u_j+p_i)}) = a_f^{0,0}$ for some f, so

$$s_2(s_2(h(U_{2(u_j+p_i)}))) = \mathsf{root}$$

which is again a contradiction. Therefore $u_j + p_i = u_{j+1}$. Showing (5) is analogous. We conclude that $t_0, \ldots, t_{k-1}$ is a solution for PCP instance.

Our claim is therefore true and satisfiability over corresponder graphs is undecidable. ■

3.3 Defining Corresponder Graphs

In this section construct graphs P and Q such that

$$G_0 \to (P \wedge \neg Q) \qquad (6)$$

iff G_0 is a corresponder graph.

When presenting the graphs $P, Q_0, \ldots, Q_{16}$ we use the following conventions. We use the label r to denote the root of the graph. We label the edges of the relation s_1 relation by 1 and the edges of s_2 by 2. Note that if a node has no outgoing edges, it would be useless in the graph in terms of specifying a set of models G_0. Every graph node will therefore have at least one outgoing edge for every label. However, in order to make the graph sketches clearer, if a node x has an outgoing edge with label a to every node in the graph, we will simply omit all a edges of node x from the sketch. In particular, if a node has no outgoing edges in the graph sketch, it means that its outgoing edges are unconstrained. A double-headed arrow from node x to node y with label a denotes two single arrows, one from x to y and one from y to x, both labeled with a. We do not show the edge $\langle \mathsf{root}, \mathsf{null} \rangle \in s_2$ that is always present in an orable graph. We also do not show the edges originating from null. We will be free to display null several times in the same picture, all these occurrences denote to the unique root node in the graph.

The graph P in Figure 5 is our first approximation of a corresponder graph. Unfortunately, P allows some models that are not corresponder graphs, such as the example in Figure 6. This is why we introduce the graph Q. The graph Q appears in (6) negated and we design it to contain models of P that are not corresponder graphs.

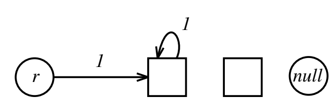

Figure 7: Graph Q_0

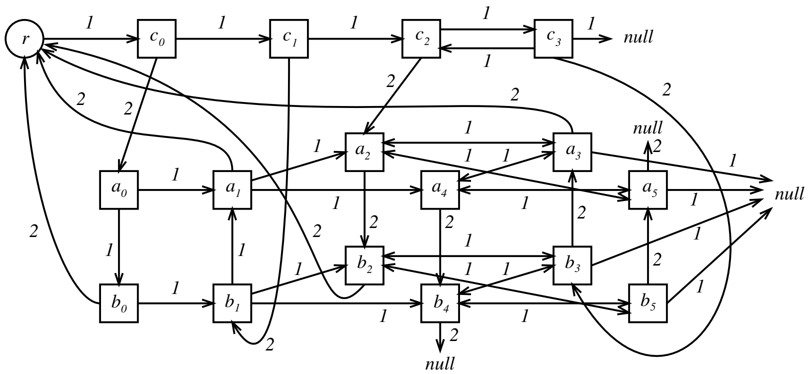

Figure 5: Graph P

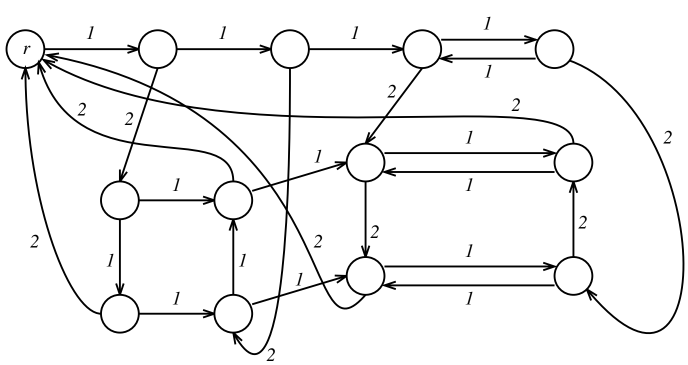

Figure 6: A model of P that is not a corresponder graph

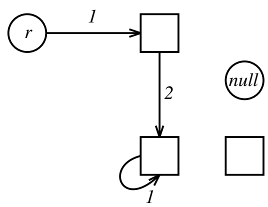

Figure 8: Graph Q_1

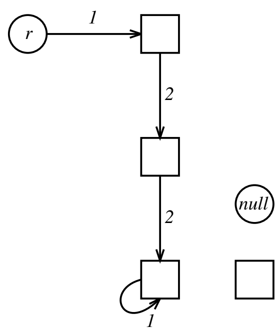

Figure 9: Graph Q_2

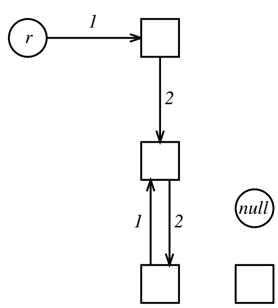

Figure 10: Graph Q_3

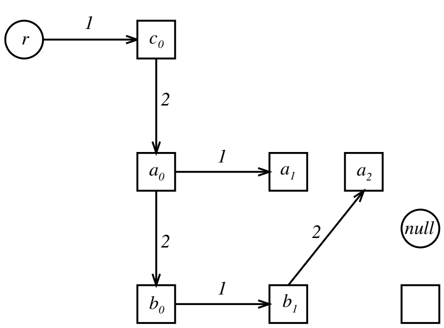

Figure 11: Graph Q_4

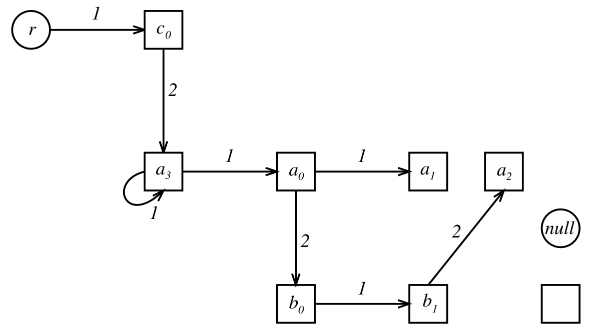

Figure 12: Graph Q_5

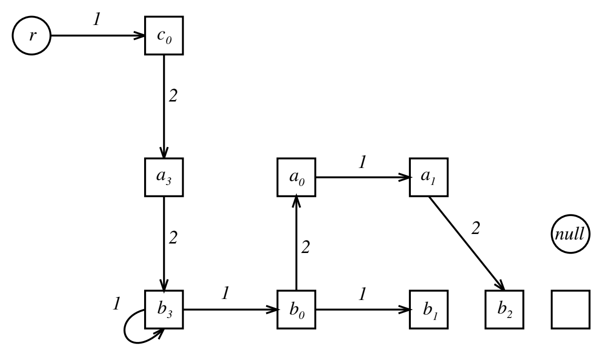

Figure 13: Graph Q_6

We construct Q as a sum of orable graphs:

$$Q = Q_0 + Q_1 + \cdots + Q_{16}$$

The idea behind the construction of these graphs comes from the proof of Proposition 37; we now give only an informal overview of the graphs. The graphs Q_0 (Figure 7) Q_1 (Figure 8), Q_2 (Figure 9), and Q_3 (Figure 10) eliminate certain cycles from the set of models of P. The graphs Q_4 (Figure 11), Q_5 (Figure 12), Q_6 (Figure 13), and Q_9 (Figure 16) ensure that different paths in the graph lead to the same object. The graphs Q_7 (Figure 14) and Q_8 (Figure 15) ensure that there is the same number of U and L-nodes in a model of P. The graphs Q_{10} (Figure 17) and Q_{11} (Figure 18) ensure that U or L nodes have an s_2 edge to root iff the U or L node in the same column has an s_2-edge from a C-node. The graphs Q_{12} (Figure 19) and Q_{13} (Figure 20) ensure that a C-node that is later in the C-list has an edge to a node that is later in the U or L list. Finally, graphs Q_{14} (Figure 21), Q_{15} (Figure 22) and Q_{16} (Figure 23) ensure that C-nodes have s_2 edges only to U and L-nodes, and that an L or U node can only have an edge to $\mathsf{root}, \mathsf{null}$, a U-node, or an L-node.

We can now show the key step in the undecidability proof for the implication of graph constraints.

Proposition 37

$$G_0 \to (P \wedge \neg Q) \tag{7}$$

iff G_0 is a corresponder graph.

Proof. $(\Longleftarrow)$: Let G_0 be a corresponder graph

$$G_0 = \mathsf{CG}(n, k, u_1, \ldots, u_{k-1}, l_1, \ldots, l_{k-1})$$

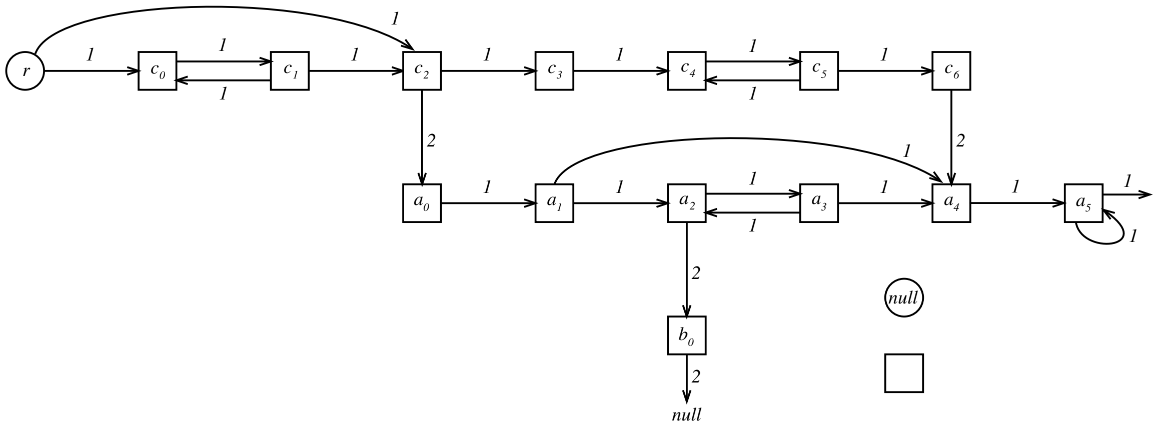

Figure 19: Graph Q_{12}

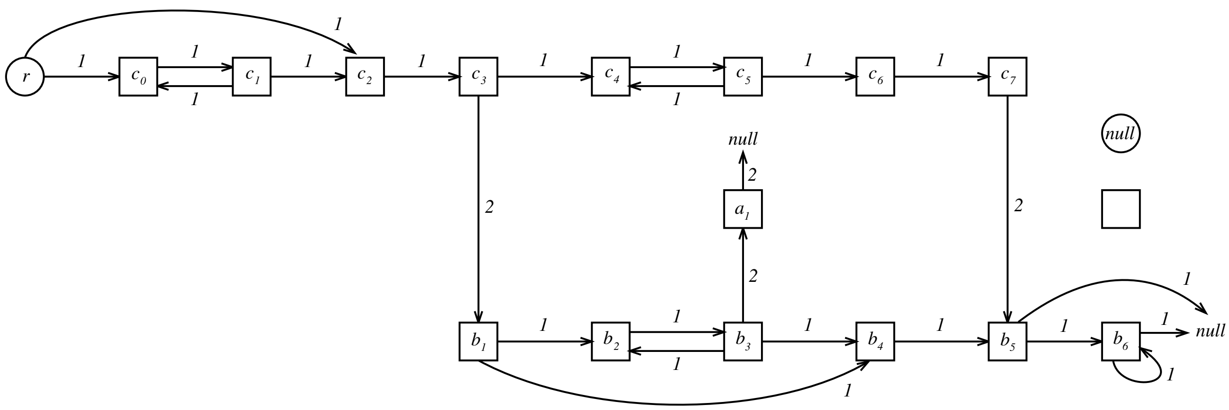

Figure 20: Graph Q_{13}

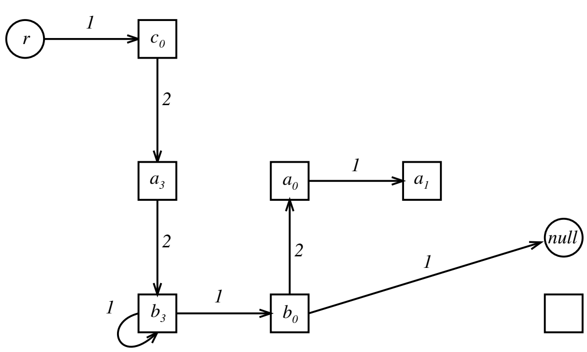

Figure 14: Graph Q_7

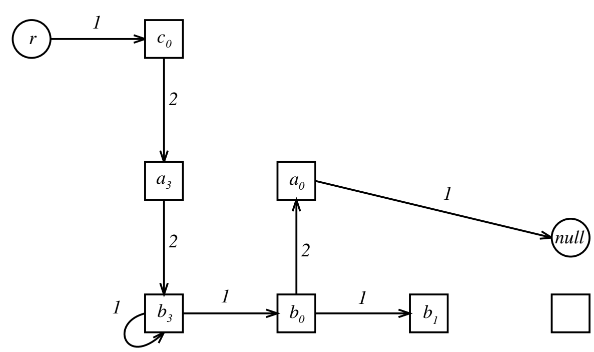

Figure 15: Graph Q_8

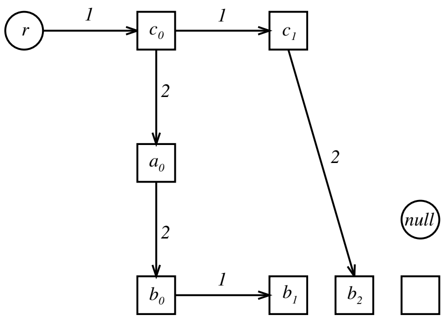

Figure 16: Graph Q_9

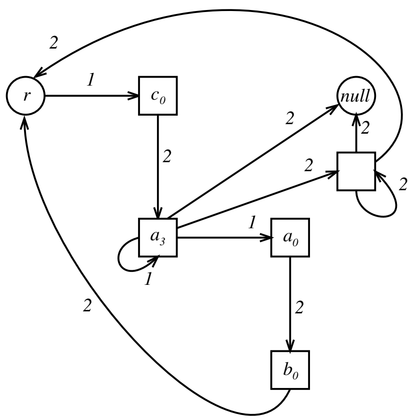

Figure 17: Graph Q_{10}

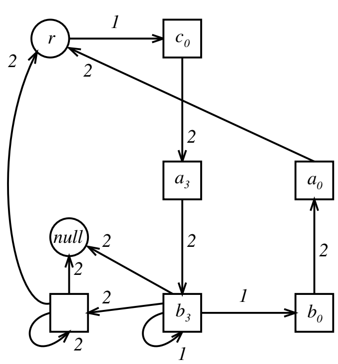

Figure 18: Graph Q_{11}

We show that $G_0 \to P$ and for all $0 \le i \le 16$, it is not the case that $G_0 \to Q_i$.

$(G_0 \to P)$: Define homomorphism h from G_0 to P as follows.

$$\begin{aligned}
h(C_0) &= c_0 \\
h(C_1) &= c_1 \\
h(C_{2j+2}) &= c_2, \qquad 0 \le j < k-1 \\
h(C_{2j+3}) &= c_3, \qquad 0 \le j < k-1 \\
h(U_0) &= a_0 \\
h(U_1) &= a_1 \\
h(U_{2j+2}) &= \begin{cases} a_2, & j+1 \in \{u_1, \ldots, u_{k-1}\} \\ a_4, & \text{otherwise} \end{cases} \\
h(U_{2j+3}) &= \begin{cases} a_3, & j+1 \in \{l_1, \ldots, l_{k-1}\} \\ a_5, & \text{otherwise} \end{cases} \\
h(L_0) &= b_0 \\
h(L_1) &= b_1 \\
h(L_{2j+2}) &= \begin{cases} b_2, & j+1 \in \{u_1, \ldots, u_{k-1}\} \\ b_4, & \text{otherwise} \end{cases} \\
h(L_{2j+3}) &= \begin{cases} b_3, & j+1 \in \{l_1, \ldots, l_{k-1}\} \\ b_5, & \text{otherwise} \end{cases}
\end{aligned}$$

It is straightforward to verify that h is indeed a homomorphism, so $G_0 \to P$.

$(\neg\, G_0 \to Q_0)$: Apply Proposition 18 with $e = 1^*$.

$(\neg\, G_0 \to Q_1)$: Apply Proposition 18 with $e = 121^*$.

$(\neg\, G_0 \to Q_2)$: Apply Proposition 18 with $e = 1221^*$.

$(\neg\, G_0 \to Q_3)$: Apply Proposition 18 with $e = 12(21)^*$.

$(\neg\, G_0 \to Q_4)$: Suppose h is a homomorphism from G_0 to Q_4. By mapping the path

$$\mathsf{root}, 1, C_0, 2, U_0, 1, U_1$$

we conclude $h(U_1) = a_1$. By mapping the path

$$\mathsf{root}, 1, C_0, 2, U_0, 2, L_0, 1, L_1, 2, U_1$$

we conclude $h(U_1) = a_2$, which is a contradiction.

$(\neg\, G_0 \to Q_5)$: Suppose h is a homomorphism from G_0 to Q_5. By mapping the slice in 121^* with h we conclude that there exists a node U_j in G_0 such that $h(U_j) = a_0$. Now as in the previous case we get that $h(U_{j+1}) = a_1$ and $h(U_{j+1}) = a_2$, which is a contradiction.

$(\neg\, G_0 \to Q_6)$: Similarly to the previous case, map the slice in 1221^* to conclude that for some node L_j we have $h(L_j) = b_0$ and then obtain a contradiction.

$(\neg\, G_0 \to Q_7)$: Similarly to the previous cases, suppose h is a homomorphism from G_0 to Q_7. By mapping the slice in 1221^* via h, we conclude that there exists a node L_j in G_0 such that $h(L_j) = b_0$. By construction of Q_7 it must be $s_1(L_j) = \mathsf{null}$. Furthermore, $h(U_j) = a_0$ and $s_1(U_j) \neq \mathsf{null}$. This is a contradiction with the fact that G_0 is a corresponder graph.

$(\neg\, G_0 \to Q_8)$: This fact is analogous to the previous one.

$(\neg\, G_0 \to Q_9)$: Suppose h is a homomorphism from G_0 to Q_9. As in the case $\neg\, G_0 \to Q_4$ we conclude $h(L_1) = b_1$ and $h(L_1) = b_2$, a contradiction.

$(\neg\, G_0 \to Q_{10})$: Suppose h is a homomorphism from G_0 to Q_{10}. As in case $\neg\, G_0 \to Q_5$, for some node U_j in

G_0 we have $h(U_j) = a_0$. Next it follows $h(L_j) = b_0$ and $s_2(L_j) = \mathsf{root}$, which implies $j = 2u_r$ where $0 \leq r < k$. Therefore $s_2(C_{2r}) = U_j$. But this is in contradiction with the fact that $h(U_j) = a_0$ and a_0 has no incoming s_2-edges in Q_{10}.

($\neg\ G_0 \to Q_{11}$) : This case is analogous to the previous one.

($\neg\ G_0 \to Q_{12}$) : Suppose h is a homomorphism from G_0 to Q_{10}. By mapping the slice in 1^* from G_0 to G we conclude that there exists a node C_{2j} in G_0 such that $h(C_{2j}) = c_2$ and a node C_{2i} where $i \geq j+2$ such that $h(C_{2i}) = c_6$. Since $s_2(c_2) = a_0$, we conclude $h(U_{2u_j}) = a_0$. By mapping the path

$$U_{2j}, 1, U_{2j+1}, 1, U_{2j+2}, 1, \ldots, 1, \mathsf{null}$$

with homomorphism h, we conclude that there exists be some U-node with even index that is mapped to a_4. So let $U_{2(u_j+t)}$ be such node with the least index. Then

$$h(U_{2(u_j+t)}) = a_4 \tag{8}$$

and

$$h(U_{2(u_j+r)}) = a_2$$

for $1 \leq r < t$. Let $1 \leq r < t$. Then $h(L_{2(u_j+r)}) = b_0$ so $s_2(L_{2(u_j+r)}) = \mathsf{null}$, which means that $u_j + r < u_{j+1}$ for all $1 \leq r < t$, so $u_j + t \leq u_{j+1} < u_i$. Therefore

$$s_2(C_{2i}) \neq U_{2(u_j+t)}$$

h is a homomorphism and $h(C_{2i}) = c_6$, so

$$h(U_{2u_i}) = a_4 \tag{9}$$

The corresponder graph G_0 contains a path in 1^* from $U_{2(u_j+t)}$ to U_{2u_i} where $U_{2(u_j+t)}$ and U_{2u_i} are distinct nodes because $u_j + t < u_i$. Because $h(U_{2(u_j+t)}) = h(U_{2u_i}) = a_4$, there exists a cyclic s_1-path from a_4 to a_4 in Q_{12}, a contradiction with the definition of Q_{12}.

($\neg\ G_0 \to Q_{13}$) : This case is analogous to the previous one.

($\neg\ G_0 \to Q_{14}$) : Suppose h is a homomorphism from G_0 to Q_{14}. Then $h(C_i) = c_2$ for some i. Let $S = s_2(C_i)$. Then $S = U_j$ or $S = L_j$ for some j and $h(S) = a_2$. On the other hand, by mapping the slices 121^* and 1221^*, we conclude that all U-nodes are mapped to a_0 and a_1 whereas all L-nodes are mapped to b_0 and b_1. This is a contradiction with $h(S) = a_2$.

($\neg\ G_0 \to Q_{15}$) : Suppose h is a homomorphism from G_0 to Q_{15}. Then for some node L_j we have $h(L_j) = b_2$ and therefore $h(U_j) = a_2$. On the other hand by mapping the slice 121^* we conclude that all U-nodes are mapped to a_0 and a_1, which is a contradiction.

($\neg\ G_0 \to Q_{16}$) : This case is analogous to the previous one.

($\Longrightarrow$) : Let

$$G_0 = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root} \rangle$$

Assume that $G_0 \to P$ and for all $0 \leq i \leq 16$ it is not the case that $G_0 \to Q_i$. We will show that G_0 is a corresponder graph. Let

$$\begin{aligned} C_0 &= s_1(\mathsf{root}) \\ C_1 &= s_1(C_0) \\ C_2 &= s_1(C_1) \\ &\ldots \end{aligned}$$

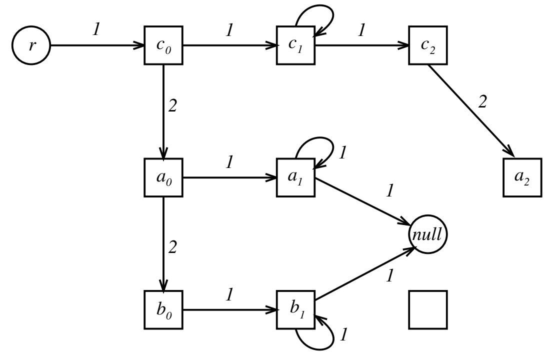

Figure 21: Graph Q_{14}

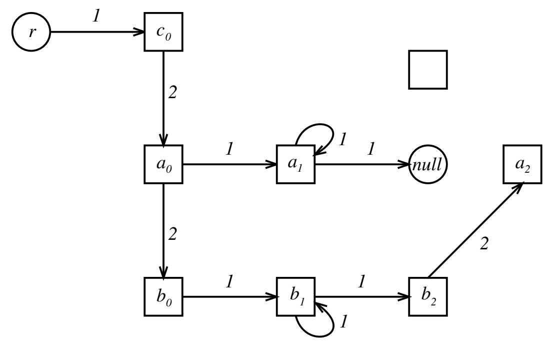

Figure 22: Graph Q_{15}

Then for all $C_i \geq 0$ we have $C_i \neq \mathsf{root}$, because $G_0 \to P$.

We claim that there must exist t such that $C_t = \mathsf{null}$ and $C_{t-1} \neq \mathsf{null}$. Suppose the claim is false. Because the graph is finite, the nodes C_i form a cycle with s_1 edges: there exist i_1, i_2 such that $0 < i_1 < i_2$, $C_{i_1} = C_{i_2}$ and $C_{i_1} \neq \mathsf{null}$. We can then show $G_0 \to Q_0$, a contradiction.

Let t be the smallest index such that $C_t = \mathsf{null}$. Then $t = 2k$ for some $k \geq 2$ because $G_0 \to P$. The nodes $\mathsf{root}, \mathsf{null}, C_0, \ldots, C_{2k-1}$ are all distinct.

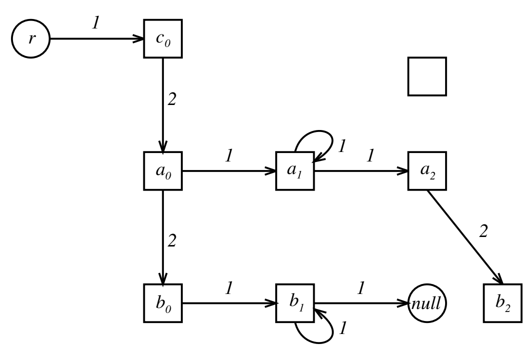

Figure 23: Graph Q_{16}

Next, consider the sequence

$$\begin{array}{rcl} U_0 & = & s_2(C_0) \\ U_1 & = & s_1(U_0) \\ U_2 & = & s_1(U_1) \\ & \dots & \end{array}$$

Because $\neg\ G_0 \to Q_1$ and $G_0 \to P$ there must exist some n_1 such that $U_{2n_1} = \mathsf{null}$ and

$$\begin{array}{rl} U_i \notin \{\mathsf{null}, \mathsf{root}\} & \cup \{C_0, \dots, C_{2k-1}\} \\ & \cup \{U_0, \dots, U_{i-1}\} \end{array}$$

for $0 \leq i < 2n_1$. Because $\neg G_0 \to Q_2$ and $G_0 \to P$, considering the sequence

$$\begin{array}{rcl} L_0 & = & s_2(U_0) \\ L_1 & = & s_1(L_0) \\ L_2 & = & s_1(L_1) \\ & \dots & \end{array}$$

we conclude there must exist n_2 such that $L_{2n_2} = \mathsf{null}$ and

$$\begin{array}{rl} L_i \notin \{\mathsf{null}, \mathsf{root}\} & \cup \{C_0, \dots, C_{2k-1}\} \\ & \cup \{U_0, \dots, U_{2n_1-1}\} \\ & \cup \{L_0, \dots, L_{i-1}\} \end{array}$$

for $0 \leq i < 2n_2$. Let

$$\begin{array}{rl} V_0 = \{\mathsf{null}, \mathsf{root}\} & \cup \{C_0, \dots, C_{2k-1}\} \\ & \cup \{U_0, \dots, U_{2n_1-1}\} \\ & \cup \{L_0, \dots, L_{2n_2-1}\} \end{array}$$

Figure 24 shows the shape of the portion of G_0 identified so far. By construction,

$$s_1[V_0] \subseteq V_0$$

In the sequel we will show that $s_2[V_0] \subseteq V_0$ holds as well. By definition of heap, all nodes in G_0 are reachable from root, which will imply $V \setminus V_0 = \emptyset$. We will also show that $n_1 = n_2$ and that G_0 satisfies the invariants that make it isomorphic to a corresponder graph.

We first observe that $s_2(C_1) = L_1$. Indeed, suppose that $s_2(C_1) \neq L_1$. From $G_0 \to P$ follows $s_2(C_1) \notin \{\mathsf{null}, \mathsf{root}\}$. We can then show $G_0 \to Q_9$, which is a contradiction.

We now show that the s_2 edges between U-nodes and L-nodes form a $(2 \times n)$-grid where $n = n_1 = n_2$. First we observe $s_2(L_1) = U_1$, otherwise we would have $G_0 \to Q_4$. Next we claim that every non-null s_2 edge originating from a U-node terminates at an L-node. Suppose $s_2(U_j)$ is not an L-node. It cannot be a C-node or a U-node because $G_0 \to P$. The only remaining possibility is that $s_2(U_j)$ is a node outside V_0. But then $G_0 \to Q_{16}$, a contradiction. Similarly, because $\neg\ G_0 \to Q_{15}$, every non-null s_2 edge of an L-node terminates at a U-node. Finally, we claim that for all $j \geq 0$, either $U_{2j} = L_{2j} = \mathsf{null}$, or all of the following holds:

- $\mathsf{null} \notin \{U_{2j}, L_{2j}, U_{2j+1}, L_{2j+1}\}$
- $s_2(U_{2j}) = L_{2j}$
- $s_2(L_{2j+1}) = U_{2j+1}$

We have already established the claim for $j = 0$. Suppose the claim does not hold for all j. Consider the least $j > 0$ for which the claim does not hold. Then one of the nodes U_{2j}, L_{2j} is not null. Assume $U_{2j} \neq \mathsf{null}$ and $L_{2j} = \mathsf{null}$. Then $G_0 \to Q_7$, a contradiction. Similarly, if $U_{2j} = \mathsf{null}$ and $L_{2j} \neq \mathsf{null}$, then $G_0 \to Q_8$, again a contradiction. So $U_{2j} \neq \mathsf{null}$ and $L_{2j} \neq \mathsf{null}$. Then from $G_0 \to P$ follows $U_{2j+1} \neq \mathsf{null}$ and $L_{2j} \neq \mathsf{null}$. From the previous discussion and $G_0 \to P$ we conclude

$$s_2(U_{2j}) = L_{2i}$$

for some $i \geq 0$. We want to show $i = j$. Suppose $i < j$. Then there is a cycle p starting at U_0 such that $\mathsf{word}(p) \in (21)^*$. But then $G_0 \to Q_3$, a contradiction. Now suppose $i > j$. Then $G_0 \to Q_6$, a contradiction. Therefore $i = j$ and $s_2(U_{2j}) = L_{2j}$. We similarly establish $s_2(L_{2j+1}) = U_{2j+1}$ using the fact $\neg\ G_0 \to Q_5$. This establishes our claim for all $j \geq 0$. We conclude that $n_1 = n_2$ and U and L-nodes are linked as in Figure 25.

We next consider s_2-edges of C-nodes and find the values $u_1, \dots, u_{k-1}$ and $l_1, \dots, l_{k-1}$. First we show that $s_2(C_{2j})$ is a U-node for $0 \leq j < k$. Suppose $s_2(C_{2j})$ is not a U-node. Because $G_0 \to P$, we conclude $s_2(C_{2j}) \notin V_0$. But then $G_0 \to Q_{14}$, a contradiction. We similarly establish from $G_0 \to P$ and $\neg G_0 \to Q_{14}$ that $s_2(C_{2j+1})$ is an L-node for $0 \leq j < k$. From $G_0 \to P$ it follows that $s_2(C_{2j}) = U_{2i}$ for some i and $s_2(C_{2j+1}) = L_{2f+1}$ for some f.

We can therefore define u_j and l_j such that

$$\begin{array}{rcl} s_2(C_{2j}) & = & U_{2u_j} \\ s_2(C_{2j+1}) & = & U_{2l_j+1} \end{array}$$

for $0 \leq j < k$.

We next show

$$s_2(L_{2j}) = \mathsf{root} \quad \text{iff} \quad \exists i\ s_2(C_{2i}) = U_{2j}$$

From $G_0 \to P$ we have $s_2(L_{2j}) \in \{\mathsf{null}, \mathsf{root}\}$. Moreover, if $s_2(C_{2i}) = U_{2j}$, then $s_2(L_{2j}) = \mathsf{null}$. It remains to show that $s_2(L_{2j}) = \mathsf{root}$ implies $s_2(C_{2i}) = U_{2j}$ for some i. Suppose that $s_2(L_{2j}) = \mathsf{root}$ but $\forall i.\ s_2(C_{2i}) \neq U_{2j}$. Then $G_0 \to Q_{10}$, a contradiction. We similarly establish

$$s_2(U_{2j+1}) = \mathsf{root} \quad \text{iff} \quad \exists i\ s_2(C_{2i+1}) = L_{2j+1}$$

using $\neg\ G_0 \to Q_{11}$.

We claim

$$u_j < u_{j+1}$$

for $0 \leq j < k-1$. Suppose the claim is false and let j be the smallest index for which $u_{j+1} \leq u_j$. Let i be such that u_i is the largest among $u_0, \dots, u_{k-1}$ with the property $u_i < u_{j+1}$. Clearly $0 \leq i \leq j-1$. Then there exists a homomorphism h from G_0 to Q_{12} such that $h(U_{2u_i}) = a_0$ and $h(U_{2u_{j+1}}) = a_4$. This is a contradiction with $\neg\ G_0 \to Q_{12}$. We similarly conclude

$$l_j < l_{j+1}$$

for $0 \leq j < k-1$, using $\neg\ Q_{13}$.

Finally we observe that we have identified all s_2-edges from V_0, so $s_2[V_0] \subseteq V_0$. Therefore $V \setminus V_0 = \emptyset$. We conclude that G_0 is isomorphic to

$$\mathsf{CG}(n_1, k, u_1, \dots, u_{k-1}, l_1, \dots, l_{k-1})$$

■

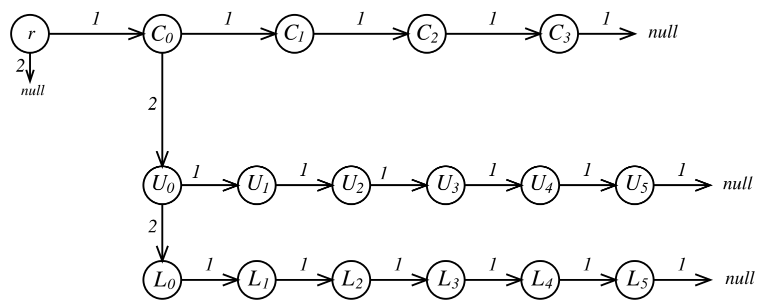

Figure 24: Graph G_0 after identifying the nodes

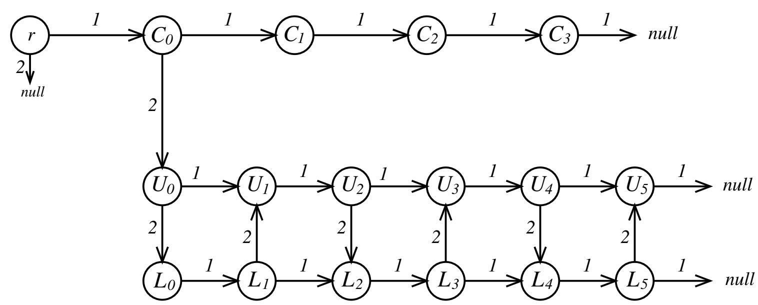

Figure 25: Graph G_0 after establishing s_2 edges between U and L-nodes

3.4 The Undecidability Result

Theorem 38 *The implication of graphs is undecidable over the class of heaps.*

Proof. We will reduce satisfiability of graphs over the class of corresponder graphs to the problem of finding a counterexample to an implication of graphs over the class of heaps. Given the reduction in Proposition 35, this will establish that the implication of graphs is Turing co-recognizable and undecidable.

Let G be a graph. Consider the implication

$$(G \times P) \leadsto_{\mathcal{H}} Q \tag{10}$$

We claim that G_0 is a counterexample for this implication iff G_0 is a corresponder graph such that $G_0 \to G$.

Assume that G_0 is a corresponder graph and $G_0 \to G$. By Proposition 37, we have $G_0 \to P$ and $\neg G_0 \to Q$. We then have $G_0 \to (G \times P)$. Since $\neg G_0 \to Q$, we conclude that G_0 is a counterexample for (10).

Assume now that G_0 is a counterexample for (10). Then $G_0 \to G \times P$ and $\neg G_0 \to Q$. Since $G_0 \to P$ and $\neg G_0 \to Q$, by Proposition 37 we conclude that G_0 is a corresponder graph. Furthermore, $G_0 \to G$. ∎

3.5 Discussion

In this section we give comments on our proof of the undecidability of implication and state some implications of this result for checking properties of programs.

3.5.1 Graph Equivalence and Negation

Definition 39 *We say that graphs G_1 and G_2 are equivalent over the class of graphs C and write*

$$G_1 \approx_C G_2$$

iff

$$G_0 \to G_1 \quad \textit{iff} \quad G_0 \to G_2$$

for every graph $G_0 \in C$.

Proposition 40 *Equivalence of graphs over the class of heaps is undecidable.*

Proof. From Proposition 25 we have

$$G_1 \leadsto_{\mathcal{H}} G_2$$

iff

$$G_1 \approx_{\mathcal{H}} G_1 \times G_2$$

The result then follows from Proposition 38. ∎

We also observe that regular graph constraints over heaps are not closed under the negation. Indeed, assume that for every graph G there exists a graph $\overline{G}$ such that the heap models of $\overline{G}$ are all heaps that are not models of G. Then finding a counterexample to an implication $P \leadsto_{\mathcal{H}} Q$ is reduced to satisfiability of the graph

$$P \times \overline{G}$$

This is a contradiction because Proposition 23 implies that satisfiability over heaps is decidable whereas Proposition 38 implies that finding a counterexample to $P \leadsto_{\mathcal{H}} Q$ is undecidable.

3.5.2 Implication of Acyclic Heaps

Corresponder graphs are a cyclic subclass of the class of heaps. The cyclicity, however, is not at all essential for our construction. We argue that implication of graphs is also undecidable over the class of acyclic heaps. We can define a minor variation of corresponder graphs where U and L nodes never point back to root. Instead, we introduce a special node different from null to indicate the difference between columns j for

$$j \in \{u_1, \ldots, u_{k-1}\} \cup \{l_1, \ldots, l_{k-1}\}$$

and the remaining columns. The resulting graphs are acyclic heaps. As a result, we have the following fact.

Proposition 41 *Implication of graphs is undecidable over the class of acyclic heaps.*

3.5.3 Alternative Proofs

An alternative way to prove undecidability would be to show that conjunction of regular graph constraints and their negations can express graphs similar to grids (instead of corresponder graphs). While the construction using grids may be possible, we have found the construction using corresponder graphs to be simpler. The reason is that corresponder graphs, unlike grids, are essentially one-dimensional structures.

Our proof of Proposition 37 could potentially be simplified by showing that a larger fragment of MSOL can be written in the form of negation of an implication of graphs. We consider formulas that can be reduced to checking negation of an implication between graphs. Let a *literal* be a formula constructed from an orable graph as in Section 2.5. Define a *homogeneous clause* as a disjunction of positive literals:

$$C_i = A_i^0 \vee \cdots \vee A_i^{n_i}$$

or a disjunction of negative literals:

$$D_i = (\neg B_i^0) \vee \cdots \vee (\neg B_i^{m_i})$$

Then any conjunction of positive and negative clauses

$$C_0 \wedge \cdots \wedge C_{n-1} \wedge D_0 \wedge \cdots \wedge D_{m-1}$$

is expressible as a negation of implication of graph constraints. This fragment appears quite expressive, but we have not been able to obtain a characterization of the fragment that allows a natural encoding a subclass like grids or corresponder graphs in a way simpler than in Proposition 37.

3.5.4 Consequences for Program Checking

Implication of graphs arises if procedure specifications are regular graph constraints.

Example 42 Consider a procedure p whose precondition is that the program heap is homomorphic to a graph G_1 and a procedure q whose precondition is that the program heap is homomorphic to a graph G_2 (Figure 26). If the first statement in the body of p is a call to $q()$, a program checker must ensure that implication $G_1 \rightsquigarrow G_2$ holds.

```
procedure p()
pre G1
{
    q();
}
procedure q()
pre G2
{
    ...
}
```

Figure 26: Program Checking Requires $G_1 \rightsquigarrow G_2$

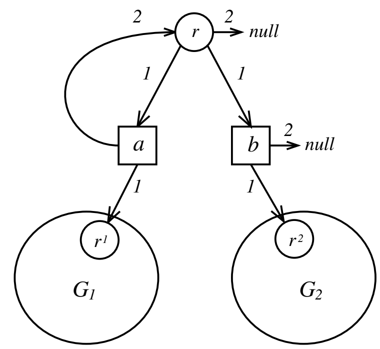

Figure 27: Ensuring an invariant requires implication

♦

We next show that the implication problem also arises when maintaining an invariant at every program point, if the invariant is a regular graph constraint. Let

$$G^1 = \langle V^1, s_1^1, s_2^1, \mathsf{null}, \mathsf{root}^1 \rangle$$

$$G^2 = \langle V^2, s_1^2, s_2^2, \mathsf{null}, \mathsf{root}^2 \rangle$$

be orable graphs such that there are no edges from nodes V^1 to root^1 and no edges from nodes V^2 to root^2. Construct the graph G (Figure 27) as

$$G = \langle V, s_1, s_2, \mathsf{null}, \mathsf{root} \rangle$$

where

$$\begin{array}{rcl} V & = & \{\mathsf{root}, a, b\} \cup V^1 \cup V^2 \\ s_1 & = & \{\langle \mathsf{root}, a \rangle, \langle \mathsf{root}, b \rangle, \langle a, \mathsf{root}^1 \rangle, \langle b, \mathsf{root}^2 \rangle\} \\ & \cup & s_1^1 \cup s_1^2 \\ s_2 & = & \{\langle \mathsf{root}, \mathsf{null} \rangle, \langle a, \mathsf{root} \rangle, \langle b, \mathsf{null} \rangle\} \\ & \cup & s_2^1 \cup s_2^2 \end{array}$$

and where

$$\{\mathsf{root}, a, b\} \cap (V^1 \cup V^2) = \emptyset$$

Suppose that we have a program checking system that verifies that a graph constraint is true after every statement. Consider the statement

$$\mathsf{root}.1.2 := \mathsf{null} \qquad (11)$$

Let

$$G^0 = \langle V^0, s_1^0, s_2^0, \mathsf{null}^0, \mathsf{root}^0 \rangle$$

be the graph before the statement. After the statement the resulting graph is

$$G^1 = \langle V^0, s_1^0, s_2^1, \mathsf{null}^0, \mathsf{root}^0 \rangle$$

where the value of 2-edge from x has changed so that it points to null:

$$s_2^1 = s_2^0[x := \mathsf{root}]$$

Our program checking system needs to verify that for all heaps G_0,

$$(G^0 \to G) \;\text{ implies }\; (G^1 \to G) \tag{12}$$

Let h a homomorphism from G^0 to G. Let $x = s_1^0(s_1^0(\mathsf{root}^0))$. Then

$$h(x) = \mathsf{root}^1$$

or

$$h(x) = \mathsf{root}^2$$

Moreover, root^1 and root^2 are reachable only through the path $\mathsf{root}^0, 1, a, 1$, so no nodes other than x may be mapped to root^1 or root^2. We can therefore show that the implication (12) is equivalent to

$$G_1 \rightsquigarrow G_2 \tag{13}$$

As explained in Section 3.5.2, we can modify the construction in the proof of Proposition 37 such that P and Q have no edges terminating at root. We then let $G^1 = P$ and $G^2 = Q$. From the undecidability of the implication of graphs over the domain of heaps it follows that maintaining an invariant expressed as a regular graph constraint is undecidable, even across a simple assignment statement such as (11).

4 Related Work

The idea of typestate as system for statically verifying changing properties of objects was proposed in [25] and extended in [24]. The original typestate system as well as the more recent work in the context object oriented programming [6] do not support constraints over dynamically allocated objects, which is the focus of our paper.

Several recent systems support tree-like dynamically allocated data structures [23, 28, 9, 18]. The restriction to tree-like data structures is in contrast to our notion of heap, which allows cycles. The presence of non-tree data structures is one of the key factors that make the implication of regular graph constraints undecidable.

The idea of representing properties of a statically unbounded number of heaps by homomorphically mapping them to a bounded family of graphs is pervasive in the work on shape analysis [16, 2, 11, 19, 20]. These analyses use abstractions that capture approximate properties of data structures even if they are not tree-like. This feature of shape analyses makes our results directly applicable. Our undecidability result implies inability to semantically check implication or equivalence of such abstractions.

Shape analysis techniques were applied to a typestate checking problem in role analysis [14]. The compositionality of the analysis and the presence of procedure specifications made the need for solving the implication of constraints in [14] explicit. The algorithm [14] uses "context matching" as a decidable approximation for the implication of constraints. In [13] it was suggested that the implication problem for role constraints is undecidable. The argument makes use of acyclicity constraints as well as the constraints on the number of incoming edges of a node. In the present paper, we have shown that undecidability holds even for the regular graph constraints, which cannot directly specify acyclicity or the number of incoming edges of a node. This makes the present undecidability result strictly stronger than the result in [13].

We were pleased to discover that the constraints derived as a simplification of role analysis constraints generalize the notions of tree automata [27, 3] and a whole family of equivalent systems over grids [12]. The remarkable fact that MSOL over trees is equivalent to tree automata inspired the question which classes of graphs have decidable MSOL theory [4]. In this paper we have introduced regular graph constraints which can be seen as a alternative to MSOL in generalizing projections of local properties over trees and grids. Although regular graph constraints are strictly weaker than MSOL (and in fact the satisfiability of regular graph constraints is decidable over heaps), we have shown that the implication for regular graph constraints over heaps is undecidable.

5 Conclusion

We have proposed regular graph constraints as an abstraction of mutually recursive properties of objects in potentially cyclic graphs. We presented some evidence that regular graph constraints are a natural generalization of the tree automata and domino systems. We have shown that satisfiability of regular graph constraints is decidable over the domain of heaps. As a main result, we have shown that the implication of regular graph constraints is undecidable. The consequence of this result is that verifying that procedure preconditions are satisfied as well as maintaining program invariants is undecidable if these properties are expressed as regular graph constraints.

We have seen that decidability of problems with regular constraints is sensitive to the choice of the class of graphs. In particular, a smaller class of graphs need not imply better decidability properties. This indicates that techniques for reasoning about different classes of graphs may be substantially different. We conclude that a good support for mechanized reasoning about data structures would likely contain a set of specialized reasoning techniques for different classes of graphs.

Acknowledgements We thank Chandrasekhar Boyapati, Yuri Gurevich, Patrick Lam and Andreas Podelski for useful discussions. We thank Chandrasekhar Boyapati and Patrick Lam for useful comments on a draft of this paper.

References

[1] Egon Börger, Erich Gräedel, and Yuri Gurevich. *The Classical Decision Problem.* Springer-Verlag, 1997.

[2] David R. Chase, Mark Wegman, and F. Kenneth Zadeck. Analysis of pointers and structures. In *Proc. ACM PLDI*, 1990.

[3] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997. release 1999.

[4] Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook of graph grammars and computing by graph transformations, Vol. 1 : Foundations*, chapter 5. World Scientific, 1997.

[5] Robert DeLine and Manuel Fähndrich. Enforcing high-level protocols in low-level software. In *Proc. ACM PLDI*, 2001.

[6] S. Drossopoulou, F. Damiani, M. Dezani-Ciancaglini, and P. Giannini. Fickle: Dynamic object reclassification. In *Proc. 15th European Conference on Object-Oriented Programming*, LNCS 2072, pages 130–149. Springer, 2001.

[7] Ronald Fagin, Larry J. Stockmeyer, and Moshe Y. Vardi. On monadic NP vs monadic co-NP. *Information and Computation*, 120(1), 1995.

[8] Cormac Flanagan, K. Rustan M. Leino, Mark Lilibridge, Greg Nelson, James B. Saxe, and Raymie Stata. Extended Static Checking for Java. In *Proc. ACM PLDI*, 2002.

[9] Pascal Fradet and Daniel Le Metayer. Shape types. In *Proc. 24th ACM POPL*, 1997.

[10] Ferenc Gecseg and Magnus Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages. Vol. III: Beyond Words*, chapter 1. Springer, 1997.

[11] Rakesh Ghiya and Laurie Hendren. Is it a tree, a DAG, or a cyclic graph? In *Proc. 23rd ACM POPL*, 1996.

[12] Dora Giammarresi and Antonio Restivo. Two-dimensional languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages Vol.3: Beyond Words*. Springer-Verlag, 1997.

[13] Viktor Kuncak. Designing an algorithm for role analysis. Master's thesis, MIT Laboratory for Computer Science, 2001.

[14] Viktor Kuncak, Patrick Lam, and Martin Rinard. Role analysis. In *Proc. 29th ACM POPL*, 2002.

[15] Viktor Kuncak and Martin Rinard. Reasoning about the heap in higher order logic. Technical report, MIT Laboratory for Computer Science, 2002.

[16] James R. Larus and Paul N. Hilfinger. Detecting conflicts between structure accesses. In *Proc. ACM PLDI*, Atlanta, GA, June 1988.

[17] K. Rustan M. Leino and Raymie Stata. Checking object invariants. Technical report, COMPAQ Systems Research Center, 1997.

[18] Anders Møller and Michael I. Schwartzbach. The Pointer Assertion Logic Engine. In *Proc. ACM PLDI*, 2001.

[19] Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Solving shape-analysis problems in languages with destructive updating. In *Proc. 23rd ACM POPL*, 1996.

[20] Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. In *Proc. 26th ACM POPL*, 1999.

[21] Thomas Schwentick and Klaus Barthelmann. Local normal forms for first-order logic with applications to games and automata. *Discrete Mathematics and Theoretical Computer Science*, 2001.

[22] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[23] F. Smith, D. Walker, and G. Morrisett. Alias types. In *Proc. 9th European Symposium on Programming*, Berlin, Germany, March 2000.

[24] Robert E. Strom and Daniel M. Yellin. Extending typestate checking using conditional liveness analysis. *IEEE Transactions on Software Engineering*, May 1993.

[25] Robert E. Strom and Shaula Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Transactions on Software Engineering*, January 1986.

[26] Wolfgang Thomas. On logics, tilings, and automata. In *Proc. 18th International Colloquium on Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*, 1991.

[27] Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages Vol.3: Beyond Words*. Springer-Verlag, 1997.

[28] David Walker and Greg Morrisett. Alias types for recursive data structures. In *Workshop on Types in Compilation*, 2000.