A SAT based algorithm for the matching problem in bigraphs with sharing

Michele Sevegnani, Chris Unsworth, and Muffy Calder

Department of Computing Science, University of Glasgow, UK
{michele,chrisu,muffy}@dcs.gla.ac.uk

Abstract. Bigraphical reactive systems are a formalism for modelling mobile computation. A bigraph consists of a place graph and a link graph; reaction rules define how place and link graphs evolve. Bigraphs originally supported only tree structures as place graphs, we have extended the formalism to *bigraphs with sharing*, which allow directed acyclic graphs as place graphs. A major challenge for bigraph tool support (with or without sharing) is defining bigraph matching and providing an efficient algorithm. In the case of bigraphs with sharing, place graph matching is a special case of the NP-complete, sub-graph isomorphism problem. We present the details of place graph matching and give a sound and complete, efficient algorithm in SAT.

1 Introduction

Bigraphical reactive systems (BRSs) are an emerging formalism for modelling mobile computation and pervasive systems. They were initially introduced by Milner [1] to provide an intuitive fully graphical model capable of representing both connectivity and locality. A BRS consists of a set of *bigraphs*, representing the state of the system, and a set of *reaction rules*, defining how the system can evolve.

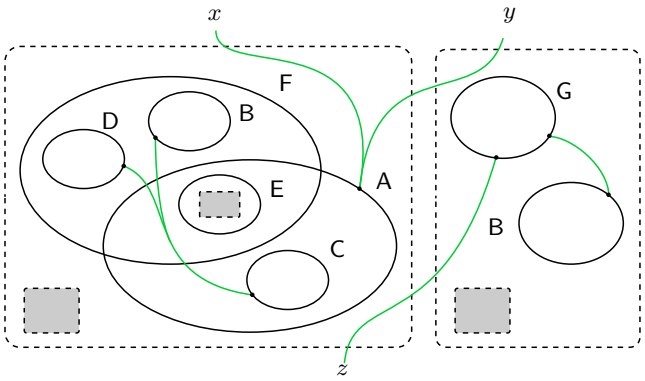
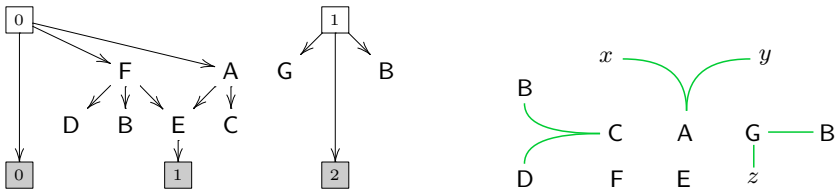


Fig. 1: A bigraph.

An example bigraph is shown in Figure 1. The two dashed rectangles, called *regions*, represent an association between the contained components such as a common location, organisation or network. The ovals and circles are *nodes*, which can represent physical or logical components within the system. Each node has a type, denoted here by the labels A to G. Nodes can be arbitrarily nested and may be situated in the intersection of one or more nodes. Each node can have zero, one or many *ports*, indicated by bullets, which represent possible connections. Actual connections are represented as *links*, depicted by green lines, which may connect two or more ports. The grey squares are *sites*, these represent parts of the model that have been abstracted away. The contents of a site can be depicted as another bigraph, where the region of the bigraph is inserted into the site. *Outer* and *inner names* can be used to indicate links (or potential links) to other bigraphs. In Figure 1 for instance, x and y are outer names and z is an inner name.



(a) Place graph (b) Link graph

Fig. 2: Place graph (a) and link graph (b)

A peculiarity of bigraphs is the complete independence of the linking and the placing of nodes, as suggested by the way that links cross boundaries in the diagram in Figure 1. This characteristic is formalized by defining bigraphs in terms of the constituent notions of *place graph* and *link graph*. Figure 2 shows the place and link graphs for the bigraph from Figure 1.

In [2] we have extended bigraphs to include place graphs with intersecting nodes, called bigraphs with sharing. This extension allows natural modelling of overlapping topologies such as signal ranges in wireless networks, offices belonging to more than one department, and overlapping biological zones and compartments. In bigraphs with sharing, a place graph is a directed acyclic graph (DAG), which may be disconnected, whose roots are the regions of the corresponding bigraph and leaves are its sites and atomic nodes. A node n_0 is a parent of a node n_1 only if n_0 contains n_1 in the original bigraph.

A link graph consists of a hyper-graph whose vertices are the names and nodes of the corresponding bigraph and hyper-edges are its links.

Reaction rules are used to model the dynamic properties of the system. *Reactions* define how a system reconfigures place and link graphs. A reaction consists of a pair (R, R'), where the *redex* R and the *reactum* R' are bigraphs that can be inserted into the same host bigraph. An example of a reaction rule for a BRS

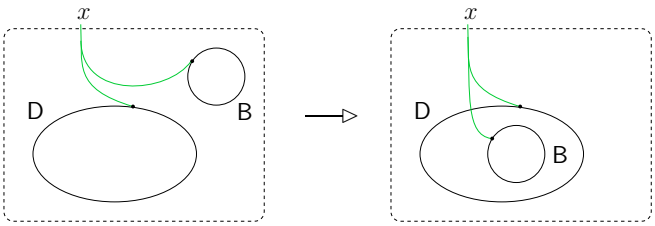


Fig. 3: An example of a reaction rule.

is represented in Figure 3. This rule states that if there are two nodes, of types D and B respectively, that share a link, then the node of type B can be moved inside the node of type D. For example, it can be seen that the redex in Figure 3 is a match for the nodes of type D and B in the node of type F in the bigraph in Figure 1. Therefore, the reaction can be applied. The resultant bigraph after performing this reaction is shown in Figure 4.

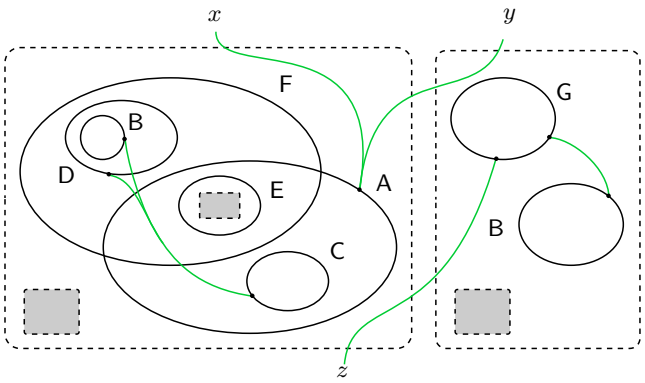


Fig. 4: The resultant bigraph after applying the reaction rule from Figure 3 to the bigraph shown in Figure 1.

For an automatic computation of the full transition system generated by a given BRS, it is of crucial importance to develop algorithms that detect efficiently whether a redex is a sub-part of a bigraph. This problem is known in the literature as the *bigraph matching problem* [3] and a prototypical implementation has been presented in [4].

To this end, we propose a sound and complete algorithm that, given a target and pattern place graph, will return a matching if one exists. It is then shown how an instance of the bigraph matching problem can be expressed in conjunctive

normal form, which will allow it to be solved by any standard SAT solver such as MiniSat [5].

- a definition of the bigraph matching problem for bigraphs with sharing,
- sound and complete algorithm for matching place graphs in bigraphs with sharing,
- a SAT encoding of the matching algorithm and empirical tests.

In the next section we give an overview of bigraphs with sharing and in Section 3 we define the bigraph matching problem. In Section 4 we discuss current matching algorithms (for the non-sharing case). Our algorithm is given in Section 5, with some examples, and in Section 6 we give the mapping to SAT and some empirical results. Conclusions are given Section 7, followed by our plans for future work.

2 Bigraphs with sharing

We now formally introduce bigraphs with sharing[1]. Let us fix some notational conventions (see [1, p. 14]). We write $S \uplus T$ for the union of two sets S and T known or assumed to be disjoint. If a function f has domain S and $S' \subseteq S$, then $f \restriction S'$ denotes the restriction of f to S'. Similarly, if f has codomain 2^T and $T' \subseteq T$, we write $f \downharpoonright T'$ to indicate $\{t \in 2^{T'} \mid \exists s \in S.\ t = (f(s) \cap T')\}$. For two functions f and g with disjoint domains S and T we write $f \uplus g$ for the function with domain $S \uplus T$ such that $(f \uplus g) \restriction S = f$ and $(f \uplus g) \restriction T = g$. By abuse of notation, for a function $f : S \to T$ we sometimes write $f(S')$ to indicate the set $\{f(i) \mid i \in S'\}$, where $S' \subseteq S$. Hence, in this case f has domain 2^S and codomain 2^T. In the following we assume that names, node-identifiers and edge-identifiers are drawn from three infinite sets, $\mathcal{X}$, $\mathcal{V}$ and $\mathcal{E}$, disjoint from each other. We denote the interfaces of bigraphs by I, J, K.

A bigraph (with sharing) is said to be *concrete* if nodes and edges have unique identifiers. It is *abstract* otherwise. When two concrete bigraphs differ only in the naming, they are said to be *support equivalent*.

The standard definition of place graphs (see [1, p. 15]) has to be extended in order to include sharing among places. This is achieved by representing place graphs as DAGs, instead of forests. Observe that links and names are unaffected by the introduction of overlapping places. Hence, the definition of link graphs remains unchanged.

Definition 1 ([1, Def. 2.1] concrete place graph with sharing). *A* concrete place graph with sharing

$$F = (V_F, ctrl_F, prnt_F) : m \to n$$

is a triple, having an inner face m and an outer face n, both finite ordinals. These index respectively the sites *and* roots *of the place graph. F has a finite*

[1] For further details refer to [2].

set $V_F \subset \mathcal{V}$ *of* nodes, *a* control map $ctrl_F : V_F \to \mathcal{K}$ *and a* parent map $prnt_F : m \uplus V_F \to 2^{V_F \uplus n}$ *which is acyclic. Parent map* $prnt_F$ *is said to be* acyclic *if* $v \in \alpha_F^i(\{v\})$ *for some* $v \in V_F$ *then* $i = 0$, *where* $\alpha_F : 2^{V_F} \to 2^{V_F}$ *is a function defined as follows:*

$$\alpha_F(W) \stackrel{def}{=} \bigcup_{v \in W} prnt_F \restriction V_F \downharpoonright V_F(v) \ .$$

Then, according to the new definition, it is possible to have $prnt_F(v) = \emptyset$ for some $v \in m \uplus V_F$. It is useful to define a function mapping nodes and roots to their children[2]:

$$prnt_F^{-1} : V_F \uplus n \to 2^{m \uplus V_F},$$

where $prnt_F^{-1}(v) = \{u \in m \uplus V_F \mid prnt_F(u) = v\}$. We also indicate the transitive closure of α_F as α_F^+. Any node or site having more than one parent is said to be shared.

A bigraph with sharing is simply the pair of its constituents, a place graph with sharing and a link graph.

Definition 2 (concrete bigraph). *A concrete bigraph*

$$F = (V_F, E_F, ctrl_F, prnt_F, link_F) : \langle k, X \rangle \to \langle m, Y \rangle$$

consists of a concrete place graph with sharing $F^{\mathsf{P}} = (V_F, ctrl_F, prnt_F) : k \to m$ *and a concrete link graph* $F^{\mathsf{L}} = (V_F, E_F, ctrl_F, link_F) : X \to Y$. *We write the concrete bigraph with sharing as* $F = \langle F^{\mathsf{P}}, F^{\mathsf{L}} \rangle$.

Some slight modifications have to be made to the definition of composition.

Definition 3 ([1, Def. 2.5] composition of place graphs with sharing). *If* $F : k \to m$ *and* $G : m \to n$ *are two place graphs with disjoint sets of node identifiers, their composite*

$$G \circ F = (V, ctrl, prnt) : k \to n$$

has nodes $V = V_F \uplus V_G$ *and control map* $ctrl = ctrl_F \uplus ctrl_G$. *Its parent map* $prnt : k \uplus V_F \uplus V_G \to 2^{V_F \uplus V_G \uplus n}$ *is defined as follows:*

$$prnt(w) \stackrel{def}{=} \begin{cases} prnt_F(w) & \text{if } w \in (k \uplus V_F) \text{ and } prnt_F(w) \subseteq V_F, \\ \bigcup_{j \in J} prnt_G(j) & \text{if } w \in (k \uplus V_F) \text{ and } prnt_F(w) = J \subseteq m, \\ prnt_F \downharpoonright V_F(w) \cup \bigcup_{j \in J} prnt_G(j) & \text{if } w \in (k \uplus V_F),\ prnt_F \downharpoonright m(w) = J \text{ and} \\ & prnt_F \downharpoonright V_F(w) \neq \emptyset, \\ prnt_G(w) & \text{if } w \in V_G. \end{cases}$$

Example 1. Let concrete place graphs $F : 2 \to 2$, $G : 2 \to 1$ and their composition $G \circ F$ be defined as in Figure 5. To show how the composition is performed, consider node v_2 in F. Its parent set is $prnt_F(v_2) = \{0, v_1\}$. The parent set of

[2] Despite the use of the $^{-1}$ notation, $prnt_F^{-1}$ is not the inverse of $prnt_F$.

site 0 in G is $prnt_G(0) = \{v'_0\}$. Then, according to Definition 3, v_2's parent set in the composite $G \circ F$ is computed as

$$\begin{aligned} prnt_{G\circ F}(v_2) &= prnt_F \restriction V_F(v_2) \cup \bigcup_{j\in J} prnt_G(j) \\ &= \{v_1\} \cup prnt_G(0) = \{v_1\} \cup \{v'_0\} \ , \end{aligned}$$

where $J = prnt_F(v_2) \cap \{0, 1\} = \{0\}$.

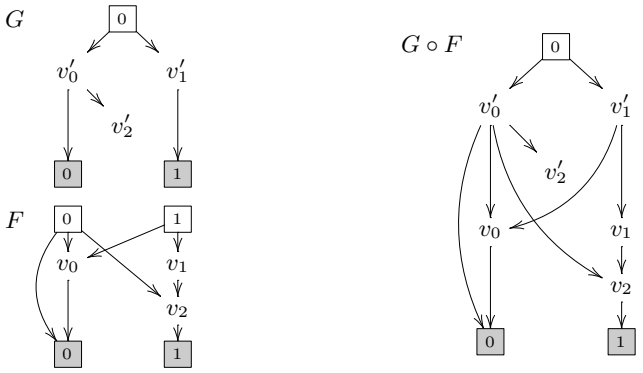


Fig. 5: Concrete place graphs with sharing $F : 2 \to 2$, $G : 2 \to 1$ and their composition $G \circ F : 2 \to 1$.

As expected, the composition of two bigraphs with sharing A and B is obtained by separately composing their place graphs and link graphs. Formally, $A \circ B = \langle A^{\mathsf{P}} \circ B^{\mathsf{P}}, A^{\mathsf{L}} \circ B^{\mathsf{L}} \rangle$. The same holds for the definition of tensor product, i.e. $A \otimes B = \langle A^{\mathsf{P}} \otimes B^{\mathsf{P}}, A^{\mathsf{L}} \otimes B^{\mathsf{L}} \rangle$.

So far, only concrete bigraphical structures have been considered. Informally, a procedure to turn a concrete bigraph into an abstract one, consists in discarding all its identifiers and idle edges (refer to [1] for a more detailed account). The inverse transformation is also possible.

Definition 4 (Concretion). *If G is an abstract bigraph, then a concrete bigraph $\hat{}G$, called a* concretion *of G, is obtained by assigning to each node a unique identifier $v \in \mathcal{V}$ and to each edge a unique identifier $e \in \mathcal{E}$. The* support *of $\hat{}G$ is given by $|\hat{}G| \stackrel{def}{=} V \uplus E$, where $V \subset \mathcal{V}$ and $E \subset \mathcal{E}$ are the identifiers used.*

Observe that any two concretions A, B of the same bigraph are always support equivalent, written $A \simeq B$. Moreover, there is a bijection $\rho : |A| \to |B|$ over identifiers, called *support translation*, that transforms A into B. This is expressed with the notation $B = \rho \cdot A$.

Finally, subcomponents of a bigraph can be specified as follows.

Definition 5 (occurrence). *A bigraph F* occurs *in a bigraph G if the equation $G = C_1 \circ (F \otimes \mathsf{id}_I) \circ C_0$ holds for some interface I and bigraphs C_0 and C_1.*

When considering concrete bigraphs, node renaming has to be taken into account.

Definition 6 (concrete occurrence). *A concrete bigraph F* occurs *in a concrete bigraph G if the equation $G = C_1 \circ (F' \otimes \mathsf{id}_I) \circ C_0$ holds for some interface I and for some concrete bigraphs F', C_0 and C_1. Moreover, $F \eqsim F'$, i.e. $F' = \rho \bullet F$ for some support translation ρ.*

An important property is that it is possible to determine an abstract occurrence starting from a concrete one. In other words, a bigraph F occurs in G only if an arbitrary concretion of F occurs in an arbitrary concretion of G. Since composition and tensor product are defined independently over the constituents of a bigraphs, the previous two definitions can be reformulated by using place graphs and link graphs only, i.e. F occurs in G only if F^{P} occurs in G^{P} and F^{L} occurs in G^{L}.

In the remainder of this document it is assumed that bigraphs allow sharing, unless stated otherwise.

3 The bigraph matching problem

The bigraph matching problem is a computational task in which a bigraph P, called pattern, and a bigraph T, called target, are given as input, and one must determine whether P occurs in T, as stated in Definition 5. Before presenting a general description of our algorithm we introduce useful notation and establish some propositions about the relationship between properties for place graphs and well-known graph theoretic properties.

One difference between place graphs and DAGs is the presence of roots and sites. As a consequence, the definition of degree of a vertex has to be extended.

Definition 7 (Degree). *Given a concrete place graph $B = (V_B, ctrl_B, prnt_B) : m \to n$, two functions $indeg : m \uplus V_B \to \mathbb{N} \times \{*, \bullet\}$ and $outdeg : V_B \uplus n \to \mathbb{N} \times \{*, \bullet\}$ may be defined as follows:*

$$indeg(v) \stackrel{def}{=} \begin{cases} (|P_v|, *) & \text{if } |P_v^*| > 0 \\ (|P_v|, \bullet) & \text{otherwise,} \end{cases}$$

$$outdeg(v) \stackrel{def}{=} \begin{cases} (|C_v|, *) & \text{if } |C_v^*| > 0 \\ (|C_v|, \bullet) & \text{otherwise.} \end{cases}$$

Sets $P_v \stackrel{def}{=} \{p \mid p \in (prnt_B \restriction V_B(v))\}$ and $P_v^* \stackrel{def}{=} \{p \mid p \in (prnt_B \restriction n(v))\}$ indicate the parents of v which are nodes and roots, respectively. Similarly, sets $C_v \stackrel{def}{=} \{c \mid c \in (prnt_B^{-1} \restriction V_B(v))\}$ and $C_v^* \stackrel{def}{=} \{c \mid c \in (prnt_B^{-1} \restriction m(v))\}$ are the children of v which are nodes and sites, respectively. We write $indeg(v)$ and $outdeg(v)$ to denote the indegree *and* outdegree *of a vertex v, respectively.*

Note that, by definition 1, roots have no parents and sites have no children. This can be expressed as follows:

$$indeg(v) = outdeg(u) = (0, \bullet) \qquad \text{for all } v \in n \text{ and } u \in m.$$

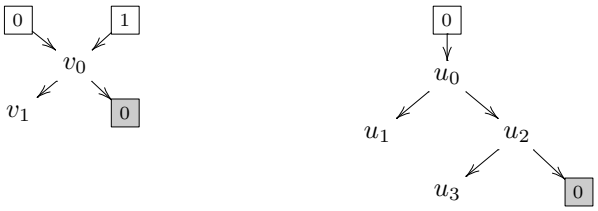
A first step toward an algorithm for matching is to introduce the notion of degree matching.

Definition 8 (Degree matching). *Given two nodes u and v, we say that u* matches (the degree of) *v if the following conditions hold:*

1. *if $outdeg(u) = (o, \bullet)$ then $outdeg(v) = (o, \bullet)$;*
2. *if $outdeg(u) = (o, *)$ then $outdeg(v) = (o', \_)$ with $o' \geq o$;*
3. *if $indeg(u) = (i, \bullet)$ then $indeg(v) = (i, \bullet)$;*
4. *if $indeg(u) = (i, *)$ then $indeg(v) = (i', \_)$ with $i' \geq i$.*

The previous definition states that a node with outdegree (indegree) $(x, \bullet)$ matches nodes with *exactly* x children (parents), while a node with outdegree (indegree) $(x, *)$ matches nodes with *at least* x children (parents). Note that nodes u and b may belong to different bigraphs. The following example gives an idea of how the procedure works on two example place graphs.

Example 2. Consider the concrete place graphs below:



Nodes u_0 and u_2 do not match node v_0, while v_0 matches both u_0 and u_1. This is shown by

$$\begin{aligned} indeg(v_0) = indeg(u_0) &= (0, *) & indeg(u_2) &= (1, \bullet) \\ outdeg(v_0) = outdeg(u_2) &= (1, *) & outdeg(u_0) &= (2, \bullet)\ . \end{aligned}$$

It is helpful for the definition of the matching algorithm to have an explicit transformation from concrete place graphs to DAGs. Such a procedure is straightforward and it can be defined as below. It consists of a map from nodes to vertices and a map from the parent map to the set of edges. Sites and roots are dropped.

Definition 9 (underlying graph). *Let $F = (V_F, ctrl_F, prnt_F) : m \to n$ be a concrete place graph. Directed graph $G_F = (V_F, E_F)$ with*

$$E_F \stackrel{def}{=} \{(u, v) \mid u \in prnt_F(v) \wedge u, v \in V_F\}$$

is called the underlying graph *of F.*

Note that the absence of cycles in parent map $prnt_F$ assures that G_F is a DAG. A crucial property for the definition of a matching algorithm in graph theoretic terms is that any control preserving graph isomorphism between two underlying graphs can trivially be extended to a support translation between two concrete place graphs. Informally, the extension is carried out by properly attaching sites and roots to the underlying graph obtained by applying the isomorphism.

Proposition 1. *Let $F^{\mathsf{P}}, G^{\mathsf{P}} : m \to n$ be two concrete place graphs and G_F, G_G their corresponding underlying graphs. If there exists a graph isomorphism $\iota : V_F \to V_G$ such that*

1. *the controls are preserved, i.e. $ctrl_F = ctrl_G \circ \iota$,*
2. *the parent map for sites is preserved, i.e. $prnt_G \restriction m = (\mathsf{id}_n \uplus \iota) \circ (prnt_F \restriction m)$,*
3. *the parent map for roots is preserved, i.e. $prnt_F \downarrow n = (prnt_G \downarrow n) \circ (\mathsf{id}_m \uplus \iota)$,*

then ι is a support translation from F^{P} to G^{P}, i.e. $G^{\mathsf{P}} = \iota \bullet F^{\mathsf{P}}$.

Proof. In order to prove that ι is a support translation from F^{P} to G^{P}, we have to prove that

1. ι preserves controls;
2. ι commutes with the structural maps as follows:

$$prnt_G \circ (\mathsf{id}_m \uplus \iota) = (\mathsf{id}_n \uplus \iota) \circ prnt_F. \tag{1}$$

The proof of (1) is immediate from the definition of ι. To prove (2), assume Equation (1) does not hold. This means that there exist a place v in F^{P} such that

$$prnt_G(\mathsf{id}_m(v)) \neq (\mathsf{id}_n \uplus \iota)(prnt_F(v)) \qquad \text{if } v \in m \tag{2}$$

$$prnt_G(\iota(v)) \neq (\mathsf{id}_n \uplus \iota)(prnt_F(v)) \qquad \text{otherwise.} \tag{3}$$

In Equation (2) the domains on both sides are restricted only to the set of sites. Therefore, it can be rewritten as $prnt_G \restriction m \neq (\mathsf{id}_n \uplus \iota) \circ (prnt_F \restriction m)$. However, this contradicts the hypothesis. Analogously, Equation (3) becomes

$$prnt_G \restriction V_G \circ \iota \neq (\mathsf{id}_n \uplus \iota) \circ prnt_F \restriction V_F. \tag{4}$$

By definition of of graph isomorphism, we know that

$$(\iota(u), \iota(v)) \in E_G \Leftrightarrow (u, v) \in E_F.$$

By construction of G_F and G_G, the following hold

$$\iota(u) \in prnt_G(\iota(v)) \qquad \iota(u), \iota(v) \in V_G \tag{5}$$

$$u \in prnt_F(v) \qquad u, v \in V_F \tag{6}$$

From (5) and (6), it follows that

$$prnt_G \restriction V_G \downarrow V_G \circ \iota = \iota \circ prnt_F \restriction V_F \downarrow V_F \tag{7}$$

By hypothesis, the parent map for roots is preserved. In particular, it is also preserved when restricting over nodes. Therefore, by merging this requirement with (7), we obtain

$$\begin{aligned}(prnt_G \restriction V_G \downharpoonright V_G \circ \iota) \uplus ((prnt_G \restriction V_G \downharpoonright n) \circ \iota) &= (\iota \circ prnt_F \restriction V_F \downharpoonright V_F) \uplus (prnt_F \restriction V_F \downharpoonright n) \\ prnt_G \restriction V_G \downharpoonright (V_G \uplus n) \circ \iota &= (\mathsf{id}_n \uplus \iota) \circ prnt_F \restriction V_F \downharpoonright (V_F \uplus n) \\ prnt_G \restriction V_G \circ \iota &= (\mathsf{id}_n \uplus \iota) \circ prnt_F \restriction V_F \qquad (8)\end{aligned}$$

But Equation (8) contradicts Equation (4). This concludes the proof.

We now give an outline of our matching algorithm. Since the novelty of sharing bigraphs resides in the definition of place graphs, we only present a procedure for the matching of place graphs. However, a similar approach can be used to implement a matching algorithm for link graphs.

The algorithm takes as input two concrete place graphs: one is the pattern and the other is the target. The output is **YES** when the pattern occurs in the target, **NO** otherwise. Recall from the previous section that this is also enough to determine whether there is a match between two abstract place graphs. The procedure consists of four different phases performed one after the other.

The first phase is an instance of the subgraph isomorphism problem, where the inputs are the underlying graphs of the pattern and the target. Intuitively, a candidate match is a subgraph of the target which is isomorphic with the pattern and respects the controls. If there are no subgraphs satisfying this condition, then the algorithm returns **NO**. The second phase is performed, otherwise. The degrees of the nodes in the isomorphic subgraphs obtained by the first phase of the algorithm are checked in the second phase. This takes care of sites and roots, as required by Definitions 6 and 3. For the same reason, shared sites and roots are checked in the third phase. Finally, in the fourth phase we check that the candidate match can be composed with a context to obtain exactly the target place graph. Informally, this consists of checking that once we leave a candidate match we never go back to it. This means that any node which is a children of a node in a candidate match P' and is outside P' does not have descendants in P' itself. The final output is **YES** when all the phases are executed correctly. It is **NO** otherwise.

4 Related work

The earliest formalisation of a matching algorithm was introduced in [3]. In this paper, the authors provide a matching system based on inference rules. These are defined formally starting from an inductive characterisation of occurrence, i.e. a pattern P can be proven to occur in a target T by induction on P and T. Proofs of soundness and completeness are also presented. The input to their matching algorithm is an abstract target and an abstract pattern. This solution does not support Bigraphs with sharing. A similar approach has been taken for the implementation of DBtk, a tool-kit for directed bigraphs [6].

We have chosen not to extend the inference approach of [3], from trees to DAGs for the following reason. Intuitively, such an approach would significantly increase the amount of unnecessary blind search in the inference process, which would adversely affect the performance of the algorithm. The additional blind search is a consequence of the complexity added by allowing sharing. This complexity is highlighted by the fact that the matching problem for bigraphs with sharing is a special case of the subgraph isomorphism problem, which is NP-complete and traditionally solved using some type of backtracking search [7]. Whereas, the matching problem without sharing is an instance of the sub-tree isomorphism problem, which can be efficiently solved in polynomial time.

Our approach relies heavily upon the extensive research and development efforts spent developing highly efficient SAT solvers. Similar approaches have been proven effective for solving several NP-complete problems by encoding them into efficient SAT models (e.g. graph colouring problem, bounded model checking,...). Since bigraphs are a special case of bigraphs with sharing, our algorithm can also be applied to solve the matching problem without sharing.

Another characteristic of our approach is that the algorithm is defined for concrete bigraphs. This allows the enumeration of all occurrences of the pattern in the target graph. This property is desirable because enumeration is required for any complete implementation of stochastic reactive systems [8]. A further benefit of our approach is that a normalisation step prior to the execution of the algorithm is not necessary.

Summarising, the two main reasons for adopting our approach are:

- the need for an efficient matching algorithm supporting bigraphs with sharing,
- the ability to count different occurrences of a match.

5 The algorithm

Following the outline we presented in the previous section, we give a formal definition of an algorithm for the matching of concrete place graphs. The algorithm takes the form of a function $\textsc{Match}(\_,\_)$. Its definition is described below.

Input: Concrete place graphs $T = (V_T, ctrl_T, prnt_T) : m \to n$ and $P = (V_P, ctrl_P, prnt_P) : m' \to n'$. T acts as target while P is the pattern we want to match in T. The algorithm is invoked by $\textsc{Match}(T, P)$.

Subgraph isomorphism: The underlying graphs of P and T are computed. We call them $G_P = (V_P, E_P)$ and $G_T = (V_T, E_T)$, respectively. If there exists a graph isomorphism $\iota : G_P \to \widetilde{G_T}$ such that $\widetilde{G_T} = (\widetilde{V_T}, \widetilde{E_T})$ is a subgraph of G_T and $ctrl_P = ctrl_T \circ \iota$, then go to the next step. Answer **NO** otherwise.

Degree check: Check whether all the nodes in V_P match the corresponding nodes in V_T. Namely, if v matches $\iota(v)$, for all $v \in V_P$, then go to the next step. Answer **NO** otherwise.

Shared roots and sites: Check that shared sites (roots) are matched properly, i.e. a shared site (root) can be matched only to nodes with a compatible parent set (set of children). To formally define this procedure, we need some definitions first. Set $M' \subseteq m'$ defined as

$$M' = \{i \in m' \mid |prnt_P \lfloor V_P(i)| > 1\} \ ,$$

is the set of shared sites in P. Function $f : M' \to 2^{V_T}$ maps the parent set of each $i \in M'$ to T. It is obtained by composing a parent map and an isomorphism $\iota : G_P \to \widetilde{G_T}$ as follows:

$$f \stackrel{\text{def}}{=} \iota \circ prnt_P \restriction M' \lfloor V_P \ .$$

We indicate by $C_i \subseteq V_T$ the set of nodes in T having a parent in common with a shared site $i \in M'$. It is constructed as

$$C_i \stackrel{\text{def}}{=} \bigcup_{j \in f(i)} prnt_T^{-1} \restriction V_T \lfloor V_T(j) \ .$$

Observe that C_i can contain elements belonging to $\iota(V_P)$. Finally, $\widehat{C_i}$ is the subset of C_i which members cannot be matched to shared site i:

$$\widehat{C_i} \stackrel{\text{def}}{=} \{j \in C_i \mid prnt_T \lfloor V_T(j) \neq \iota(prnt_P \lfloor V_P(i))\} \ .$$

The procedure is:
Input: set $\widehat{C_i}$.
Procedure: If for all $j \in \widehat{C_i}$ there exist a set of sites $M'' \subseteq m'$ such that

$$\iota\left(\bigcup_{h \in M''} prnt_P \restriction M'' \lfloor V_P(h)\right) = prnt_T \restriction V_T \lfloor \iota(V_P)(j)$$

holds, then return **YES**. Return **NO** otherwise.
If the procedure returns **YES** for all $\widehat{C_i}$ with $i \in M'$, then go to the next step. Return **NO** otherwise. The case for shared roots is symmetric.

Transitive closure: Compute set $C \subseteq V_T$, defined as follows:

$$C = \{v \notin \iota(V_P) \mid (u, v) \in E_T \wedge u \in \iota(V_P)\} \ .$$

Compute the transitive closure $G_T^+ = (V_T, E_T^+)$ of G_T. If all the nodes in C are not connected (via relation E_T^+) to a node belonging to $\iota(V_P)$, namely

$$\{(u, v) \in E_T^+ \mid u \in C \wedge v \in \iota(V_P)\} = \emptyset,$$

then answer **YES**. Answer **NO** otherwise.

We now prove that the algorithm is sound and complete.

Proposition 2 (completeness). *Let $S : m \to n$ and $R : m' \to n'$ be two concrete place graphs. If R occurs in S, then* $\textsc{Match}(S, R) = \textbf{YES}$.

Proof. By Definition 6, there are concrete bigraphs C_1, C_0, R', an interface p and a support translation ρ such that

$$S = C_1 \circ (R' \otimes \mathsf{id}_p) \circ C_0 \tag{9}$$
$$R' = \rho \cdot R \tag{10}$$

Assume (9) and (10) hold; we prove that $\textsc{Match}(S, R) = \textbf{YES}$. By construction, Algorithm Match returns **YES** only if all its four constituent phases do not return **NO**. Hence, we prove individually that each phase does not return **NO**.

1. Assume the hypothesis holds and the output of the first phase of the algorithm is **NO**. By construction, this means that one of the following conditions is not satisfied:
   (a) There exists an isomorphism $\iota : G_R \to \widetilde{G_S}$.
   (b) $\widetilde{G_S} = (\widetilde{V_S}, \widetilde{E_S})$ is a subgraph of G_S.
   (c) $ctrl_R(v) = ctrl_S(\iota(v))$ for all $v \in V_R$.
   By hypothesis, we know there exists a place graph $R' : m' \to n'$ such that

   $$prnt_{R'} \circ (\mathsf{id}_{m'} \uplus \rho) = (\mathsf{id}_{n'} \uplus \rho) \circ prnt_R \tag{11}$$
   $$ctrl_{R'} \circ \rho = ctrl_R \ . \tag{12}$$

   Equation (11) can be restricted when considering only nodes as shown below:

   $$prnt_{R'} \restriction V_{R'} \downharpoonright V_{R'} \circ \rho = \rho \circ prnt_R \restriction V_R \downharpoonright V_R \ . \tag{13}$$

   By Definition 9 and Equation (13), ρ is an isomorphism from G_R to $G_{R'}$. Moreover, by Definitions 3, 9 and Equation (9), $G_{R'}$ is a subgraph of G_S. From this and Equation (12) it also follows that ρ preserves the controls. Therefore, if we pick ι and $\widetilde{G_S}$ such that

   $$\widetilde{G_S} = G_{R'} \qquad \text{and} \qquad \iota = \rho \ ,$$

   then all the three conditions are satisfied and the first phase of the algorithm does not return **NO**. This is a contradiction.
2. Assume the hypothesis holds, and the second phase returns **NO**. This means that there exits a node v in R such that the degree of $\rho(v)$ in S is not matched by v. In the following we assume the indegree is not matched. The case for outdegree is symmetric. By Definition 8 we have two cases:
   (a) $\overline{indeg}(v) = (o, \bullet)$ and $\overline{indeg}(\rho(v)) \neq (o, \bullet)$ and
   (b) $\overline{indeg}(v) = (o, *)$ and $\overline{indeg}(\rho(v)) = (o', \_)$ with $o' < o$.
   We prove them separately.
   (2a)
   In the first case, for any $v \in V_R$ such that $\overline{indeg}(v) = (o, \bullet)$, it holds that

   $$|prnt_R(v)| = o \qquad \text{and} \qquad prnt_R(v) \subset V_R \ . \tag{14}$$

   By Equation (10), it also holds that

   $$|prnt_{R'}(\rho(v))| = o \qquad \text{and} \qquad prnt_{R'}(\rho(v)) \subset V_{R'} \ . \tag{15}$$

By Definition 3 and by Equations (9), (15) we have that

$$prnt_S \restriction V_{R'} \downharpoonright V_{R'} = prnt_{R'} \restriction V_R$$

However, this implies that

$$|prnt_S(\rho(v))| = o \qquad \text{and} \qquad prnt_S(\rho(v)) \subset V_S \ . \tag{16}$$

Therefore, $\overline{indeg}(\rho(v)) = (o, \bullet)$. This is a contradiction.
(2b)
Similarly, in the second case by hypothesis and Equation (10), the following hold

$$|prnt_{R'} \downharpoonright V_{R'}(\rho(v))| = o \qquad \text{and} \qquad prnt_{R'} \downharpoonright n'(\rho(v)) \neq \emptyset \ , \tag{17}$$

for any $v \in V_R$ such that $\overline{indeg}(v) = (o, *)$. By Definition 3 and Equation (9) we have that

$$prnt_S(\rho(v)) = prnt_{S_0} \downharpoonright V_{S_0}(\rho(v)) \cup \bigcup_{j \in J} prnt_{C_1}(j) \ , \tag{18}$$

where $S_0 = (R' \otimes \mathsf{id}_p) \circ C_0$ and $J = prnt_{S_0} \downharpoonright (n' + p)(\rho(v))$. By Definition 3, we know that none of the parents of a node in R' can be in C_0. Therefore, Equation (18) becomes

$$prnt_S(\rho(v)) = prnt_{R'} \downharpoonright V_{R'}(\rho(v)) \cup \bigcup_{j \in J'} prnt_{C_1}(j) \ , \tag{19}$$

with $J' = prnt_{R'} \downharpoonright n'(\rho(v))$. By Equations (17) and (19), we can write

$$\begin{aligned} |prnt_S(\rho(v))| &= |prnt_{R'} \downharpoonright V_{R'}(\rho(v))| + \left| \bigcup_{j \in J'} prnt_{C_1}(j) \right| \\ &= o + \left| \bigcup_{j \in J'} prnt_{C_1}(j) \right| \\ &= o' \geq o \ . \end{aligned}$$

This implies, $\overline{indeg}(\rho(v)) = (o', \_)$ with $o' \geq o$. This contradicts the hypothesis.

3. Assume the hypothesis holds, and the third phase returns **NO**. This means that there exists no set of sites M'' in R such that

$$prnt_S \restriction V_S \downharpoonright \iota(V_R)(j) = \iota \left(\bigcup_{h \in M''} prnt_R \restriction M'' \downharpoonright V_R(h) \right) \ ,$$

for some node j in S and some isomorphism ι from R to S. The case with shared roots is treated symmetrically, therefore we prove only the case for

sites. By definitions of composition, and Equation (9), we know that for any $w \in (V_{C_0} \uplus m)$ such that $prnt_S(w) \not\subseteq V_{C_0}$, the following holds

$$prnt_S(w) = prnt_{C_0} \lfloor V_{C_0}(w) \cup \bigcup_{i \in J} prnt_{S_1}(i)$$

where $S_1 = C_1 \circ (R' \otimes \mathsf{id}_p)$ and $J = prnt_{C_0} \lfloor (m' + p)(w)$. By restricting the domain, we obtain

$$prnt_S \restriction V_S(w) = prnt_{C_0} \restriction V_{C_0} \lfloor V_{C_0}(w) \cup \bigcup_{i \in J'} prnt_{S_1}(i) \ , \tag{20}$$

where $J' = prnt_{C_0} \restriction V_{C_0} \lfloor (m' + p)(w)$. By further restricting the codomain, Equation (20) can be rewritten as

$$\begin{aligned} prnt_S \restriction V_S \lfloor V_{R'}(w) &= \bigcup_{i \in J''} prnt_{S_1} \lfloor V_{R'}(i) \\ &= \bigcup_{i \in J''} prnt_{R'} \restriction J'' \lfloor V_{R'}(i) \ , \end{aligned} \tag{21}$$

where $J'' = prnt_{C_0} \restriction V_{C_0} \lfloor m'(w)$ is a set of sites in R'. Finally, by Equation (10), Equation (21) becomes

$$prnt_S \restriction V_S \lfloor \rho(V_R)(w) = \rho \left(\bigcup_{i \in J''} prnt_R \restriction J'' \lfloor V_R(i) \right) \ .$$

By choosing $j = w$, $\iota = \rho$ and $M'' = J''$ we obtain a contradiction.

4. Assume the hypothesis holds, and the last phase of the algorithm returns **NO**. Then,
$$\{(u, v) \in E_S^+ \mid u \in C \wedge v \in \iota(V_R)\} \neq \emptyset \tag{22}$$
must hold, with C as in Equation (9) and E_S^+ the transitive closure of the edge relation of G_S. By Equations (9), (10) and Definition 9, each element u belonging to set C is a node in C_0, i.e. $u \in V_{C_0}$. We know that any descendant v of u is also an element of V_{C_0} and not of $V_{R'} = \rho(V_R)$. This implies
$$\{(u, v) \in E_S^+ \mid u \in C \wedge v \in \rho(V_R)\} = \emptyset$$
where $\rho = \iota$. This is a contradiction.

This concludes the proof.

Proposition 3 (soundness). *Let $S : m \to n$ and $R : m' \to n'$ be two concrete place graphs. If* $\textsc{Match}(S, R) = \mathbf{YES}$*, then R occurs in S.*

Proof. In the following we prove that, whenever $\textsc{Match}(S, R) = \mathbf{YES}$, with $S : m \to n$ and $R : m' \to n'$, then it is possible to construct bigraphs $C_1, C_0, R', \mathsf{id}_p$ and a support translation ρ such that Equations (9), (10) hold. Let us begin with the construction of ρ and R'.

In the first phase of the algorithm a graph ismorphism $\iota : G_R \to \widetilde{G_S}$ is computed. By Definition 9, $\widetilde{G_S}$ can be the underlying graph of any place graph having $\widetilde{V_S} = \iota(V_R)$ as node set. In particular, it is the underlying graph of place graph $R' : m' \to n'$ constructed as follows:

$$\boxed{\begin{aligned} V_{R'} &\stackrel{\text{def}}{=} \iota(V_R) \\ ctrl_{R'} &\stackrel{\text{def}}{=} ctrl_R \circ \iota^{-1} \\ prnt_{R'} \restriction V_{R'} \downharpoonright V_{R'} &\stackrel{\text{def}}{=} prnt_R \restriction V_R \downharpoonright V_R \\ prnt_{R'} \restriction m' &\stackrel{\text{def}}{=} (\mathsf{id}_{n'} \uplus \iota) \circ prnt_R \restriction m' \\ prnt_{R'}^{-1} \restriction n' &\stackrel{\text{def}}{=} (\mathsf{id}_{m'} \uplus \iota) \circ prnt_R^{-1} \restriction n' \ . \end{aligned}} \tag{23}$$

Then, by Proposition 1, $\iota = \rho$ is a support translation and R' is support equivalent to the input place graph R.

We now describe the construction of C_0, C_1 and id_p. By definition of identity, and by construction of R', we know that $V_{C_0} \cup V_{C_1} = V_S \setminus V_{R'}$. We also know that there exists a set of nodes $V_X \subseteq V_S \setminus V_{R'}$ whose elements can be either nodes in C_0 and C_1. In this construction we arbitrarily choose to have $V_X \subseteq V_{C_1}$, i.e. the construction specifies the decomposition of S with the maximum context C_1. Therefore, set V_{C_0} contains all and only the nodes being descendant of a node in R'. It is defined as

$$V_{C_0} \stackrel{\text{def}}{=} \{v \in V_S \mid \overline{prnt}_S^+(\{v\}) = U \wedge U \cap V_{R'} \neq \emptyset\} \ .$$

This implies that $V_{C_1} \stackrel{\text{def}}{=} V_S \setminus (V_{R'} \cup V_{C_0})$. Interface d is computed as the number of different parents (in C_1) of places in C_0. It can be computed as

$$d \stackrel{\text{def}}{=} |P_0| \quad \text{with} \quad P_0 = \{v \in (V_{C_1} \uplus n) \mid v \in prnt_S(u) \wedge u \in (V_{C_0} \uplus m)\} \ .$$

Note that a bijection f between elements of P_0 and d (i.e. sites of C_1) can be constructed. Moreover, it is possible to specify a function $h : 2^{V_{R'}} \to 2^{m'}$ that associates a set of sites M to a set of nodes of $W \subseteq V_{R'}$, where M is the smallest subset of m' such that

$$W = \bigcup_{i \in M} prnt_{R'}(i) \ .$$

Informally, $h(W)$ indicates a set of non-redundant sites such that its parent set in bigraph R' is the same as the parent set of W in bigraph S. Observe that the previous definition is sound with respect to phases 2 and 3 of the algorithm. At this point we have all the elements to define place graph $C_0 : m \to m' + p$:

$$\boxed{\begin{aligned} ctrl_{C_0} &\stackrel{\text{def}}{=} ctrl_S \restriction V_{C_0} \\ prnt_{C_0}(w) &\stackrel{\text{def}}{=} prnt_S \downharpoonright V_{C_0}(w) \cup f(prnt_S \downharpoonright P_0(w)) \\ &\quad \cup h(prnt_S \downharpoonright V_{R'}(w)) \ . \end{aligned}} \tag{24}$$

The construction of place graph $C_1 : n' + d \to m$ is dual to the procedure described above. However, some additional definitions are required. Set P_R of places in C_1 having a child in R' is given by

$$P_R = \{v \in (V_{C_1} \uplus n) \mid v \in prnt_S(u) \wedge u \in V_{R'}\} \ ,$$

while function $k : P_R \to 2^{V_{R'}}$ is just a shorthand for

$$k = prnt_S^{-1} \restriction P_R \downharpoonright V_{R'} \ .$$

For each $w \in P_R$, we then define set M'_w as the smallest subset of n' such that the following holds

$$M'_w = \bigcup_{i \in k(w)} prnt_{R'} \downharpoonright n'(i) \ . \tag{25}$$

Finally, place graph C_1 is specified as follows:

$$\boxed{\begin{array}{rll} ctrl_{C_1} \stackrel{\text{def}}{=} ctrl_S \restriction V_{C_1} & & \\ prnt_{C_1} \restriction V_{C_1} \stackrel{\text{def}}{=} prnt_S \restriction V_{C_1} & & \\ prnt_{C_1}^{-1} \downharpoonright n'(w) \stackrel{\text{def}}{=} M'_w & & w \in P_R \\ prnt_{C_1} \restriction d \stackrel{\text{def}}{=} f^{-1} \ . & & \end{array}} \tag{26}$$

Note that the existence of sets M and M'_w is assured by phases 2 and 3 in the algorithm. This completes the proof.

5.1 Examples

The various phases of algorithm MATCH are illustrated with some examples.

Example 3. Consider place graphs T and P_0 drawn in Figure 6. The matching instance is given by MATCH(T, P_0). The first operation performed by the algorithm is the computation of the underlying graphs G_{P_0} and G_T. In the first phase five possible control preserving isomorphisms are detected:

$$\begin{array}{ll} \iota_0 = \{(v_0, u_0), (v_1, u_1)\} & \iota_1 = \{(v_0, u_0), (v_1, u_2)\} \\ \iota_2 = \{(v_0, u_3), (v_1, u_6)\} & \iota_3 = \{(v_0, u_4), (v_1, u_7)\} \\ \iota_4 = \{(v_0, u_5), (v_1, u_7)\} \ . & \end{array}$$

Therefore, it is possible to continue with the degree check in the second phase of the algorithm. The degrees of the nodes involved in the isomorphisms are:

$$\begin{array}{rr} indeg(a) = (0, *) & outdeg(b) = (0, *) \\ indeg(c) = (1, \bullet) & outdeg(d) = (1, \bullet) \\ indeg(u_7) = (2, \bullet) & outdeg(u_0) = (3, \bullet) \ , \end{array}$$

where a ranges over nodes v_0, u_0, $b = v_1, u_6, u_7$, $c = v_1, u_1, \ldots, u_6$ and $d = v_0, u_1, \ldots, u_5$. Since node v_0 cannot match the outer degree of u_0, isomorphisms ι_0 and ι_1 have to be discarded. Similarly, ι_3 and ι_4 are not accepted because v_1 does not match the inner degree of u_7. Thus, it follows that ι_2 is the only isomorphism in which all the degrees are matched. Since there are no shared roots and sites in the pattern P_0, the third phase of the algorithm is skipped and the last phase is executed instead. The subgraph of G_T induced by isomorphism ι_2 is $\widetilde{G_T} = (\{u_3, u_6\}, \{(u_3, u_6)\})$. Since both its vertices do not have children not in $\widetilde{G_T}$, set C computed in the fourth phase is empty. Therefore, the computation of the transitive closure is not necessary. Finally, the output of $\textsc{Match}(T, P_0)$ is **YES**, and the matching is given by ι_2.

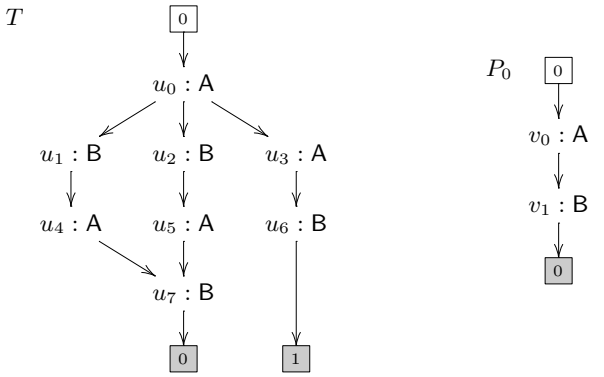


Fig. 6: Example 3: $\textsc{Match}(T, P_0) =$ **YES**, i.e. P_0 occurs in T. The match is given by $\{(v_0, u_3), (v_1, u_6)\}$.

Example 4. Consider concrete place graphs T and P_1 depicted in Figure 8. When $\textsc{Match}(T, P_1)$ is invoked, the following steps can be described. In the first phase of the algorithm two control preserving isomorphisms are found:

$$\iota_0 = \{(v_0, u_0), (v_1, u_1), (v_2, u_4)\}$$
$$\iota_1 = \{(v_0, u_0), (v_1, u_2), (v_2, u_5)\} \ .$$

In the second phase the degrees are checked. The degrees of the nodes belonging to T are reported in Example 3. For the nodes in P_1 the degrees are:

$$indeg(v_0) = outdeg(v_2) = (0, *) \qquad outdeg(v_0) = (1, *)$$
$$indeg(v_1) = indeg(v_2) = (1, \bullet) \qquad outdeg(v_1) = (1, \bullet) \ .$$

Since all the degrees are matched, with both the isomorphisms, the algorithm proceeds to the next step of the algorithm. As in the previous example, the third

phase is not executed because there are neither shared roots nor sites in P_1. The fourth phase checks that all paths leaving the matched pattern $\widetilde{G_T}$ are not going back into it. In order to do that, the algorithm computes the set C of children of nodes in $\widetilde{G_T}$ that are not in $\widetilde{G_T}$. In this case, there are two sets $C_0 = \{u_2, u_3, u_7\}$ and $C_1 = \{u_1, u_3, u_7\}$, obtained by considering the match given by isomorphisms ι_0 and ι_1, respectively. Analogously, we have two different matched patterns:

$$\widetilde{G_T^0} = (\overbrace{\{u_0, u_1, u_4\}}^{\widetilde{V_T^0}}, \{(u_0, u_1), (u_1, u_4)\})$$
$$\widetilde{G_T^1} = (\underbrace{\{u_0, u_2, u_5\}}_{\widetilde{V_T^1}}, \{(u_0, u_2), (u_2, u_5)\}).$$

Then, the transitive closure of G_T is computed as in Figure 7. As can be seen,

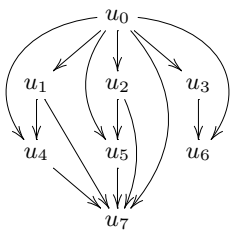


Fig. 7: Example 4: Transitive closure of underlying graph G_T.

there are no edges connecting elements of C_0 with vertices belonging to set $\widetilde{V_T^0}$. This means that isomorphism ι_0 gives a valid match. The same holds for ι_1 when considering sets C_1 and $\widetilde{V_T^1}$. Therefore, the final output of the algorithm is **YES**. Note that to perform the check in the fourth phase, the computation of the full transitive closure is not always necessary. In this case for instance, edges starting from node u_0 are not required to be added since u_0 cannot be reached by any path starting with an element in C_0 or C_1. However, in the worst case all vertices in the target are reachable starting from set C and the entire transitive closure has to be computed.

Example 5. Take place graphs T and P_2 as in Figure 9. We now describe the execution of MATCH(T, P_2). During the first phase of the algorithm two control preserving isomorphisms are found:

$$\iota_0 = \{(v_0, u_0), (v_1, u_1), (v_2, u_4), (v_3, u_7)\}$$
$$\iota_1 = \{(v_0, u_0), (v_1, u_2), (v_2, u_5), (v_3, u_7)\} \ .$$

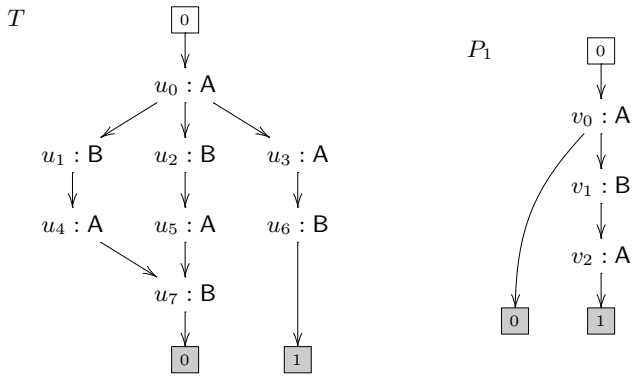


Fig. 8: Example 4: MATCH(T, P_1) = **YES**. There are two matches: $\{(v_0, u_0), (v_1, u_1), (v_2, u_4)\}$ and $\{(v_0, u_0), (v_1, u_2), (v_2, u_5)\}$.

These give rise to the corresponding candidate matches:

$$\widetilde{G_T^0} = (\{u_0, u_1, u_4, u_7\}, \{(u_0, u_1), (u_1, u_4), (u_4, u_7)\})$$
$$\widetilde{G_T^1} = (\{u_0, u_2, u_5, u_7\}, \{(u_0, u_2), (u_2, u_5), (u_5, u_7)\}) \ .$$

In the second phase the degrees are checked. Degrees of vertices in G_T are listed in Example 3, while the degrees of nodes belonging to G_{P_2} are:

$$\begin{aligned} &indeg(v_0) = outdeg(v_3) = (0, *) \\ &outdeg(v_0) = indeg(v_3) = (1, *) \\ &indeg(v_i) = outdeg(v_i) = (1, \bullet) \qquad\qquad i = 1, 2. \end{aligned}$$

Since all the nodes in G_T are matched by the correspondent nodes in G_{P_2}, the algorithm moves to the fourth phase (third phase is skipped because there are neither shared roots nor shared sites in the pattern). In the fourth phase the transitive closure of graph G_T is computed as reported in Figure 7. Moreover, the sets of children are $C_0 = \{u_2, u_3\}$ and $C_1 = \{u_1, u_3\}$. At this point, the algorithm discards isomorphism ι_0 because edge (u_2, u_7) connects C_0 to the a vertex in candidate match $\widetilde{G_T^0}$. Similarly, ι_1 is not accepted because edges (u_1, u_7) connects C_1 to $\widetilde{G_T^1}$. Therefore, the algorithm returns **NO**.

Example 6. Take as input for the matching algorithm place graphs T' and P_3. They are the target and the pattern, respectively. A graphical representation is given in Figure 10. When invoking MATCH(T', P_3), the first phase of the matching algorithm is performed. As a result, isomorphism $\iota : \{(v_0, u_0), (v_1, u_1), (v_2, u_2)\}$ is computed. In the second phase the degree matching is performed. The degrees

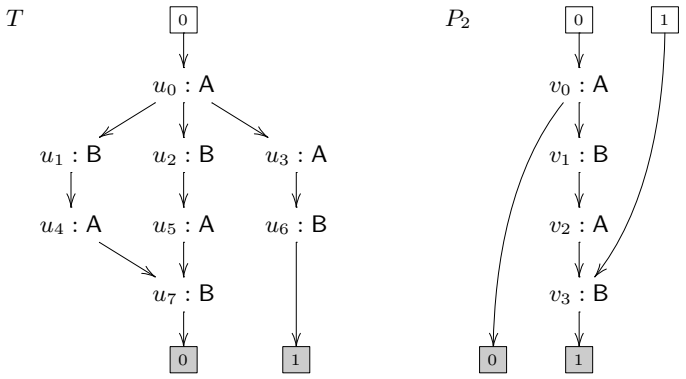


Fig. 9: Example 5: $\textsc{Match}(T, P_2) = \mathbf{NO}$.

are:

$$\begin{aligned} indeg(a) = outdeg(b) &= (0, *) && a = v_0, u_0 \; b = v_1, v_2 \\ indeg(c) &= (1, \bullet) && c = v_1, v_2, u_1, u_2 \\ outdeg(d) &= (2, \bullet) && d = v_0, u_0, u_1, u_2 \; . \end{aligned}$$

Since all the degrees are matched (in particular v_1 matches u_1 and v_2 matches u_2), the third phase can be executed. In this example, site 0 in the pattern is shared between nodes v_1 and v_2. Therefore, the algorithm computes the set of nodes in T' that cannot be assigned to site 0 as follows: $\widehat{C_0} = \{u_3, u_5\}$. Then, the procedure defined in the third phase is executed using u_3 and u_5 as inputs. In the first case, the procedure fails to find a set of sites in the pattern being children only of node v_1. In the second case, the procedure again returns **NO** because there are no sites which parent set is $\{v_2\}$. Note that site 0 do not satisfy the requirement in both cases because its parent set is $\{v_1, v_2\}$. At this point the algorithm interrupts its computation and returns **NO**. This means that P_3 does not occur in T'.

6 A SAT encoding

In this section we show how the place graph matching problem can be modelled and solved efficiently with a specialised SAT solver [5]. SAT solvers check satisfiability of propositional formulae (usually written in conjunctive normal form). Though in general NP-complete, SAT solvers are highly efficient for many practical applications.

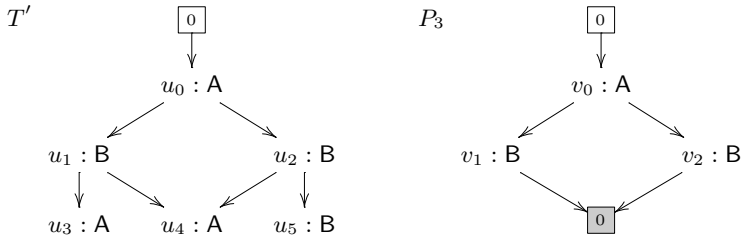


Fig. 10: Example 6: $\textsc{Match}(T', P_3) = \mathbf{NO}$.

6.1 Representing place graph matching as SAT

The problem of place graph matching (ignoring roots and sites for now) can be expressed as a sub-graph isomorphism problem with the additional constraints of vertex labels and degree matching. We now show how the sub-graph isomorphism problem can be expressed as a SAT problem and show how additional clauses can be added to handle vertex labels and degree matching. It is then shown how this model can be extended to include roots and sites.

The problem is assumed to be expressed as two directed acyclic graphs (DAGs), the pattern graph $\mathcal{P}$ and a target graph $\mathcal{T}$. $\mathcal{P}v$ is the set of n vertices associated with $\mathcal{P}$. $\mathcal{P}a$ is an $n \times n$ adjacency matrix, which holds data on the edges between the n vertices in $\mathcal{P}$, where $\mathcal{P}a_{ij} = 1$ iff there is an edge starting at vertex i and ending at vertex j. $\mathcal{P}l$ is an array of length n that holds the vertex labels, where $\mathcal{P}l_i$ is the label for vertex i.

6.2 Sub-graph isomorphism

Given a pattern graph $\mathcal{P}$ and a target graph $\mathcal{T}$, where $\mathcal{P}$ and $\mathcal{T}$ are directed acyclic graphs, find a matching $\mathcal{M} = \{(\mathcal{P}v_1, \mathcal{T}v_k), (\mathcal{P}v_2, \mathcal{T}v_l), \ldots, (\mathcal{P}v_n, \mathcal{T}v_p)\}$. Such that, for all $(\mathcal{P}v_i, \mathcal{T}v_k) \in \mathcal{M}$ and $(\mathcal{P}v_j, \mathcal{T}v_l) \in \mathcal{M}$ the appropriate edges in $\mathcal{P}$ and $\mathcal{T}$ match, meaning, $\mathcal{P}a_{ij} = \mathcal{T}a_{kl}$

To represent this as a SAT instance, a literal L_{ij} is used to represent each possible $(\mathcal{P}v_i, \mathcal{T}v_j)$ pair in $\mathcal{M}$. If there are n vertices in $\mathcal{P}$ and m vertices in $\mathcal{T}$ then there will be $n \times m$ literals in the model. A summary of the clauses is shown in Figure 11. To ensure that each $\mathcal{P}v_i$ is matched to at least one $\mathcal{T}v$, the clause $L_{i1} \vee L_{i2} \vee \cdots \vee L_{im}$ (Clause 1) is added. The clause $\neg L_{ik} \vee \neg L_{il}$ (Clause 2) is then added for each $\mathcal{P}v_i$ and each pair $(\mathcal{T}v_k, \mathcal{T}v_i)$ to ensure that $\mathcal{P}v_i$ is matched to at most one $\mathcal{T}v$. For all $\mathcal{T}v_k$ the clause $\neg L_{ik} \vee \neg L_{jk}$ (Clause 3) is added for each pair $(\mathcal{P}v_k, \mathcal{P}v_i)$, so $\mathcal{T}v_k$ is matched to at most one vertex in $\mathcal{P}$. Finally, for each potential pair of assignments $(\mathcal{P}v_i, \mathcal{T}v_k) \in \mathcal{M}$ and $(\mathcal{P}v_j, \mathcal{T}v_l) \in \mathcal{M}$, such that $\mathcal{P}a_{ij} \neq \mathcal{T}a_{kl}$, the clause $\neg L_{ik} \vee \neg L_{jl}$ is added.

1. for all $\mathcal{P}v_i$	$L_{i1} \vee L_{i2} \vee \cdots \vee L_{im}$
2. for all $\mathcal{P}v_i$, $\mathcal{T}v_k$, $\mathcal{T}v_l$ s.t. $k < l$	$\neg L_{ik} \vee \neg L_{il}$
3. for all $\mathcal{P}v_i$, $\mathcal{P}v_j$, $\mathcal{T}v_k$ s.t. $i \neq j$	$\neg L_{ik} \vee \neg L_{jk}$
4. for all $\mathcal{P}v_i, \mathcal{P}v_j, \mathcal{T}v_k, \mathcal{T}v_l$ s.t. $\mathcal{P}a_{ij} \neq \mathcal{T}a_{kl}$	$\neg L_{ik} \vee \neg L_{jl}$

Fig. 11: Summary of clauses for a SAT model for sub-graph isomorphism.

6.3 Vertex labels and degree matching

In a matching $\mathcal{M}$ a vertex $\mathcal{P}v_i$ can only be matched to sum vertex $\mathcal{T}v_k$ if the labels and the degrees match. The labels match if $\mathcal{P}l_i = \mathcal{T}l_k$ and the degrees match if both:

$$\text{In degree } \sum_{j=1}^{n} \mathcal{P}a_{ij} = \sum_{l=1}^{m} \mathcal{T}a_{kl}$$

and

$$\text{Out degree } \sum_{j=1}^{n} \mathcal{P}a_{ji} = \sum_{l=1}^{m} \mathcal{T}a_{lk}$$

If a pair of vertices $(\mathcal{P}v_i, \mathcal{T}v_k)$ do not match then a single literal clause $\neg L_{ik}$ is added. Both roots and sites can be matched to 0, 1 or many vertices. Therefore, if a vertex $\mathcal{P}v_i$ has a parent which is a root, $\mathcal{P}v_i$ can be matched to a vertex in $\mathcal{T}$ that does not have a matching in degree. Similarly, if a vertex $\mathcal{P}v_i$ has a child which is a site, $\mathcal{P}v_i$ can be matched to a vertex in $\mathcal{T}$ that does not have a matching out degree. The degree matching of such vertices will be enforced by clauses detailed in the following section.

6.4 Roots and sites

A root is modelled as a specialised vertex, which is allowed to be matched to 0 or more vertices. A vertex $\mathcal{P}v_i$ is a root if its associated label $\mathcal{P}l_i =$ "*root*". A root $\mathcal{P}v_i$ is represented by m literals in the same way as normal vertices, where $L_{ik} = 1$ indicates that the vertex $\mathcal{T}v_k$ is matched to the root $\mathcal{P}v_i$. A summary of the clauses for roots is shown in Figure 12. If a vertex $\mathcal{T}v_k$ is matched to a root then all its ancestors must also be matched to the same root. This is enforced by the clause $L_{ik} \vee \neg L_{il}$ (Clause 1), which is posted for each pair $(\mathcal{T}v_k, \mathcal{T}v_l)$ such that there is an edge $\mathcal{T}a_{kl}$.

If a vertex $\mathcal{P}v_i$ is a root, vertex $\mathcal{P}v_j$ is a child of $\mathcal{P}v_i$, $\mathcal{P}v_j$ is matched to a vertex $\mathcal{T}v_k$ and $\mathcal{T}v_l$ is a parent of $\mathcal{T}v_k$ then $\mathcal{T}v_l$ must be matched to a parent of $\mathcal{P}v_j$. This relationship is enforced by the clause $\neg L_{jk} \vee_{\mathcal{P}v_{i'} \in P} L_{i'l}$ (Clause 2), where P is the set of all parents of $\mathcal{P}v_j$. This clause will be posted for all $\mathcal{P}v_j$, such that $\mathcal{P}a_{ij} = 1$ and $\mathcal{P}l_i =$ "*root*", for all $\mathcal{T}v_k$ and for all $\mathcal{T}v_l$, such that

$\mathcal{P}a_{lk} = 1$. Note that is relation is the same if $\mathcal{P}v_i$ were not a root. However, in the none root case the relationship is enforced by the degree matching clauses and Clause 4 from Figure 11.

If a vertex is matched to a root then it must not be matched to any other vertex[3]. This is enforced by the clause $\neg L_{ik} \vee \neg L_{jk}$ (Clause 3), which will be posted for all $\mathcal{P}l_i =$ "*root*", for all $\mathcal{P}l_j \neq$ "*root*" and for all $\mathcal{T}v_k$.

If a root in the pattern graph has 2 children, then any vertex matched to that root must either have children that are matched to the children of the root or be an ancestor of such a vertex. This is enforced by the clause $\neg L_{j^1k^1} \vee \neg L_{j^2k^2} \vee \neg L_{il}$ (Clause 4), which is posted for all roots $\mathcal{P}v_i$ with 2 children $\mathcal{P}v_{j^1}$ and $\mathcal{P}v_{j^2}$ and for all vertex $\mathcal{T}v_l$ with a child $\mathcal{T}v_{k^1}$ and all vertices $\mathcal{T}v_{k^2}$ which are not children of $\mathcal{T}v_l$. Note that a similar relation needs to be enforced for roots with more than 2 children.

1.	for all $\mathcal{P}l_i =$ "*root*", $\mathcal{T}v_k, \mathcal{T}v_l$ s.t. $\mathcal{T}a_{kl} = 1$	$L_{ik} \vee \neg L_{il}$
2.	for all $\mathcal{P}l_i =$ "*root*", $\mathcal{T}v_k$ s.t. $\mathcal{T}a_{kl} = 1, \mathcal{P}a_{ij} = 1,$ $P = \{\mathcal{P}a_{i'j} \vert \mathcal{P}v_{i'}\}$	$\neg L_{jk} \vee_{\mathcal{P}v_{i'} \in P} L_{i'l}$
3.	for all $\mathcal{P}l_i =$ "*root*", $\mathcal{P}l_j \neq$ "*root*", $\mathcal{T}v_k$	$\neg L_{ik} \vee \neg L_{jk}$
4.	for all $\mathcal{P}l_i =$ "*root*", $\mathcal{P}a_{ij^1} = 1,$ $\mathcal{P}a_{ij^2} = 1, \mathcal{T}a_{lk^1} = 1, \mathcal{T}a_{lk^2} = 0$	$\neg L_{j^1k^1} \vee \neg L_{j^2k^2}$ $\vee \neg L_{il}$

Fig. 12: Summary of clauses for adding roots to the sub-graph isomorphism SAT problem.

Similar to a root, a site is modelled as a specialised vertex, which is allowed to be matched to 0 or more vertices. A site differs from a root by the fact that a site is a leaf node. A vertex $\mathcal{P}v_i$ is a site if its associated label $\mathcal{P}l_i =$ "*site*". A site $\mathcal{P}v_i$ is represented by m literals in the same way as vertices, where $L_{ik} = 1$ indicates that the vertex $\mathcal{T}v_k$ is matched to the site $\mathcal{P}v_i$. A summary of the clauses for sites is shown in Figure 13. If a vertex $\mathcal{T}v_k$ is matched to a site then all its descendants must also be matched to the same site. This is enforced by the clause $\neg L_{ik} \vee L_{il}$ (Clause 1), which is posted for each pair $(\mathcal{T}v_k, \mathcal{T}v_l)$ such that there is an edge $\mathcal{T}a_{kl}$.

If a vertex $\mathcal{P}v_i$ is a site, vertex $\mathcal{P}v_j$ is a parent of $\mathcal{P}v_i$, $\mathcal{P}v_j$ is matched to a vertex $\mathcal{T}v_k$ and $\mathcal{T}v_l$ is a child of $\mathcal{T}v_k$ then $\mathcal{T}v_l$ must be matched to a child of

[3] A vertex may be matched to more than one root.

$\mathcal{P}v_j$. This relationship is enforced by the clause $\neg L_{jk} \vee_{\mathcal{P}v_{i'} \in C} L_{i'l}$ (Clause 2), where C is the set of all children of $\mathcal{P}v_j$. This clause will be posted for all $\mathcal{P}v_j$, such that $\mathcal{P}a_{ji} = 1$ and $\mathcal{P}l_i =$ "*site*", for all $\mathcal{T}v_k$ and for all $\mathcal{T}v_l$, such that $\mathcal{P}a_{kl} = 1$. Note that is relation is the same if $\mathcal{P}v_i$ were not a site. However, for in the none site case the relationship is enforced by the degree matching clauses and Clause 4 from Figure 11.

If a vertex is matched to a site then it must not be matched to any other vertex[4]. This is enforced by the clause $\neg L_{ik} \vee \neg L_{jk}$ (Clause 3), which will be posted for all $\mathcal{P}l_i =$ "*site*", for all $\mathcal{P}l_j \neq$ "*site*" and for all $\mathcal{T}v_k$.

If a site in the pattern graph has 2 parents, then any vertex matched to that site must either have parents that are matched to the parents of the site or be a descendant of such a vertex. This is enforced by the clause $\neg L_{j^1k^1} \vee \neg L_{j^2k^2} \vee \neg L_{il}$ (Clause 4), which is posted for all sites $\mathcal{P}v_i$ with 2 parents $\mathcal{P}v_{j^1}$ and $\mathcal{P}v_{j^2}$ and for all vertex $\mathcal{T}v_l$ with a parent $\mathcal{T}v_{k^1}$ and all vertices $\mathcal{T}v_{k^2}$ which are not parents of $\mathcal{T}v_l$. Note that a similar relation needs to be enforced for sites with more than 2 parents.

1.	for all $\mathcal{P}l_i =$ "*site*", $\mathcal{T}v_k, \mathcal{T}v_l$ s.t. $\mathcal{T}a_{kl} = 1$	$\neg L_{ik} \vee L_{il}$
2.	for all $\mathcal{P}l_i =$ "*site*", $\mathcal{T}v_k$ s.t. $\mathcal{P}a_{ji} = 1, \mathcal{T}a_{kl} = 1,$ $C = \{\mathcal{P}a_{ji'} \mid \mathcal{P}v_{i'}\}$	$\neg L_{jk} \vee_{\mathcal{P}v_{i'} \in C} L_{i'l}$
3.	for all $\mathcal{P}l_i =$ "*site*", $\mathcal{P}l_j \neq$ "*site*", $\mathcal{T}v_k$	$\neg L_{ik} \vee \neg L_{jk}$
4.	for all $\mathcal{P}l_i =$ "*site*", $\mathcal{P}a_{j^1i} = 1$ $\mathcal{P}a_{j^2i} = 1, \mathcal{T}a_{k^1l} = 1, \mathcal{T}a_{k^2l} = 0$	$\neg L_{j^1k^1} \vee \neg L_{j^2k^2}$ $\vee \neg L_{il}$

Fig. 13: summary of clauses for adding sites to the sub-graph isomorphism SAT problem.

6.5 Avoiding multiple matches

The clauses described in Section 6.4 to handle roots and sites enforce the relationship $L_{ik} \Rightarrow L_{il}$ where $\mathcal{P}l_i =$ "*site*" and $\mathcal{T}v_l$ is an ancestor of $\mathcal{T}v_k$ and a similar version for sites. The relationship should be $L_{ik} \Leftrightarrow L_{il}$. Fixing this in the model will require potentially lengthy clauses. The result of the relaxation will be that some vertices may be erroneously matched to a site or root.

[4] A vertex may be matched to more than one site.

However, the correct enforcement of all the other relationships means that the resultant match will be sound. The potential problem with this is that the usual technique for finding all solutions to a SAT problem[5] may result in multiple equivalent matches being returned, which will artificially increase the number of matches and thus waste effort. To avoid this, when attempting to find additional matches, instead of posting a clause which is a negation of the entire solution, a clause will be posted which will force only the significant literals to be different. This clause would be a disjunction of the negated literals that represent the matchings of the concrete vertices only. For example, if the solution was $\{(\mathcal{P}v_1, \mathcal{T}v_3), (\mathcal{P}v_2, \mathcal{T}v_4), (\mathcal{P}v_3, \mathcal{T}v_5), (\mathcal{P}v_4, \mathcal{T}v_8),$
$(\mathcal{P}v_5, \mathcal{T}v_9)\}$, where $\mathcal{P}v_1$ and $\mathcal{P}v_5$ are roots and sites respectively, then the clause $\neg L_{2,4} \vee \neg L_{3,5} \vee \neg L_{4,8}$ would be posted.

This solution has been tested with all the examples in this paper. All the correct matchings were returned in less than a millisecond. Larger place graphs were then randomly generated. It was found that instances with 500 nodes in the target graph and between 10 and 100 nodes in the pattern graph could be solved in 20 to 30 seconds. The solution was also tested on the a pair of graphs taken from a model of the 802.11 protocol [2] shown in Figure 14. A single match was correctly returned in less than a millisecond. This is sufficient as a proof of concept. However, a more efficient encoding should yield better performance.

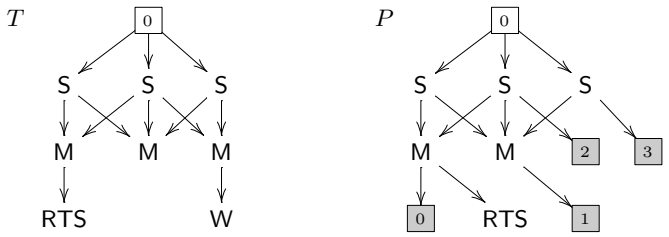


Fig. 14: Example instance: 802.11 protocol [2]

7 Conclusions

We have proposed a sound and complete algorithm for finding matchings in place graphs for bigraphs with sharing. It has been shown how the matching problem can be mapped into a SAT instance. As a special case of the sub-graph isomorphism problem, we strongly suspect that the bigraph matching problem with sharing is NP-complete. Therefore, as with many other such problems, a

[5] After a solution is found, the negation of that solution is posted as a clause and the search is restarted.

SAT based solution is likely to provide an efficient solution. Initial empirical results imply that this is the case.

8 Future work

The next step is to apply this approach, for place graphs, to link graphs. This will allow the two solutions to be combined to form a matching algorithm for a full bigraph. We are particularly interested in a recent generalisation of link graphs called *link graph with aliases* which has been proposed in [9]. In this new definition, a link can have any number of outer names. Examples of bigraphs making use of aliases are given in Figures 1 and 4, where the port attached to the node of control A is connected to both names x and y. We plan to use a similar SAT based approach since the matching problem for link graphs is related to the sub-hypergraph isomorphism problem, which is NP-complete. An algorithm for binding bigraphs will also be considered. It would then be interesting to compare empirically our solution with the inference rules based approach proposed for non-sharing bigraphs.

Finally, we will develop an automated tool to perform simple reachability analysis over BRSs.

References

1. Milner, R.: The Space and Motion of Communicating Agents. Cambridge University Press (2009)
2. Sevegnani, M., Calder, M.: Bigraphs with sharing. Technical Report TR-2010-310, University of Glasgow (2010)
3. Birkedal, L., Damgaard, T.C., Glenstrup, A.J., Milner, R.: Matching of bigraphs. Electr. Notes Theor. Comput. Sci. **175**(4) (2007) 3–19
4. Birkedal, L., Bundgaard, M., Damgaard, T.C., Debois, S., Elsborg, E., Glenstrup, A.J., Hildebrandt, T.T., Milner, R., Niss, H.: Bigraphical programming languages for pervasive computing. In: Proceedings of Pervasive 2006 International Workshop on Combining Theory and Systems Building in Pervasive Computing. (May 2006) 653–658
5. Eén, N., Sörensson, N.: An extensible sat-solver. In: SAT. (2003) 502–518
6. Bacci, G., Grohmann, D., Miculan, M.: Dbtk: A toolkit for directed bigraphs. In: CALCO. (2009) 413–422
7. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM **23**(1) (1976) 31–42
8. Krivine, J., Milner, R., Troina, A.: Stochastic bigraphs. Electr. Notes Theor. Comput. Sci. **218** (2008) 73–96
9. R., M.: Seminar notes on development in bigraphs. http://www.cl.cam.ac.uk/~rm135/Bigraphs-Seminars.pdf (17 March 2010)