Finding a Forest in a Tree

The matching problem for wide reactive systems*

Giorgio Bacci[1], Marino Miculan[2], and Romeo Rizzi[3]

[1] Dept. of Computer Science, Aalborg University, Denmark. grbacci@cs.aau.dk
[2] Dept. of Mathematics and Computer Science, University of Udine, Italy. marino.miculan@uniud.it
[3] Dept. of Computer Science, University of Verona, Italy. romeo.rizzi@univr.it

Abstract. *Wide reactive systems* are rewriting systems specified by *wide* reaction rules, where redex and reactum are lists of terms (*forests*), i.e. rules of the form $\langle l_1(\boldsymbol{x}_1), \ldots, l_n(\boldsymbol{x}_n)\rangle \Rightarrow \langle r_1(\boldsymbol{y}_1), \ldots, r_n(\boldsymbol{y}_n)\rangle$ such that $\cup_i \boldsymbol{y}_i \subseteq \cup_i \boldsymbol{x}_i$. Wide reaction rules are particularly useful for process calculi for mobile and global computations, because they allow one to connect processes which can be *at different places* in the system, possibly crossing boundaries and firewalls. For instances, *remote procedure calls* can be modeled as a process in place i activating a reaction in a different place j; *code mobility* can be modeled by instantiating variables in $\boldsymbol{y}_i$ with terms using variables from $\boldsymbol{x}_j$, for a different j; etc.
In order to apply a wide reaction rule, we have to find a matching of the rule redex within the global state. This problem can be restated as follows: how to match a given forest (the redex) inside an unordered tree (the system), possibly finding the subtrees to be grafted at the forest's leaves (i.e., instantiating the variables)? We show that, although the problem is NP-complete in general, the exponential explosion depends only on the number n of roots of the forest (the *width* of the redex), and not on the size of the global tree (the system state). In most practical cases, the width is constant and small (i.e., ≤ 3), hence our results show that the wide reaction systems can be actually used for process calculi.

1 Introduction

It is common to present the dynamics of agents of process calculi by means of a *reduction semantics*, that is a relation of the form $a \twoheadrightarrow a'$, where a and a' are agents. This relation is usually defined by means of a set $\mathcal{R}$ of *(parametric) reaction (or reduction) rules* of the form $l(x_1, \ldots, x_n) \Rightarrow r(x_1, \ldots, x_n)$ where x_i are (agent) variables, and $l(\boldsymbol{x})$, $r(\boldsymbol{x})$ are terms called *redex* and *reactum*, respectively. Reaction rules induce reactions relations according the usual (non-deterministic) term rewriting: rule $l(\boldsymbol{x}) \Rightarrow r(\boldsymbol{x})$ can be applied to an agent (i.e., a closed term) a if $l(\boldsymbol{x})$ *matches* some subterm of a, that is, if $a = C[l(\boldsymbol{x})\sigma]$ for some context $C[\_]$ (indicating the position of the rewriting) and substitution σ. The result term of this rule application is then $a' = C[r(\boldsymbol{x})\sigma]$. Many general frameworks with efficient implementations have been developed around this concept, such as *term rewriting systems* [2] and *rewriting logics* [20].

* This work is partially supported by MIUR PRIN project 2010LHT4KM, *CINA*.

However, the approach presented above considers just one subterm replacement at once (or, at most, parallel execution of independent reductions). In fact, we can easily generalize reactive systems to consider *wide* reaction rules, as in Milner's *reactive systems* [19]. A wide rule has the form $\boldsymbol{l}(\boldsymbol{x}) \Rightarrow \boldsymbol{r}(\boldsymbol{y})$, where $\boldsymbol{l}(\boldsymbol{x}) = \langle l_1(\boldsymbol{x}_1), \ldots, l_n(\boldsymbol{x}_n)\rangle$ and $\boldsymbol{r}(\boldsymbol{y}) = \langle r_1(\boldsymbol{y}_1), \ldots, r_n(\boldsymbol{y}_n)\rangle$ are lists of terms, i.e., *forests*; n is called their *width*. The variables in the reactum must be a subset of those in the redex (i.e., $\cup_i \boldsymbol{y}_i \subseteq \cup_i \boldsymbol{x}_i$). Applying such a rule to an agent a means 1) finding a context with n holes $C[X_1, \ldots, X_n]$ and a substitution σ for the variables $\boldsymbol{x}$ such that $a = C[\boldsymbol{l}(\boldsymbol{x})\sigma]$; 2) replace each $l_i(\boldsymbol{d}_i)$ with the corresponding $r_i(\boldsymbol{y}_i)[\sigma]$. This can be summarized in the following rule:

$$\frac{\boldsymbol{l}(\boldsymbol{x}) \Rightarrow \boldsymbol{r}(\boldsymbol{y}) \in \mathcal{R} \qquad a = C[\boldsymbol{l}(\boldsymbol{x})\sigma] \qquad a' = C[\boldsymbol{r}(\boldsymbol{y})\sigma]}{a \twoheadrightarrow a'} \tag{1}$$

Notice that subtrees $l_i(\boldsymbol{d})$ can occur in any order in the agent a, but not one under another. Traditional reactive systems are the particular case when $n = 1$.

Although not very widespread in practice, wide rules and wide reactive systems are quite useful and expressive, especially for process calculi for mobile and global computations. The key aspect is that in a wide rule we can deal with processes which can be *everywhere* in the system, not only "in the same place". For example, a *remote procedure call* can be modeled as a process in a place (i-th tree of the forest) activating a reaction in a different position (i.e., a rewriting of the j-th tree); *code mobility* can be modeled by instantiating some variables in $\boldsymbol{y}_i$ in place i with terms using variables from $\boldsymbol{x}_j$, for a different j; etc.. This kind of reductions can be represented also using non-wide reaction rules, but at the expense of a more cumbersome and less natural reduction semantics.

As an example, let us consider CaSPiS [5], a session-centered calculus developed within the SENSORIA project as a core language for Service Oriented Computing programming. CaSPiS allows processes to synchronize even if they are far apart (as long as they share a channel). To define this behaviour using a standard reduction semantics, we would need an infinite set of rules of the form

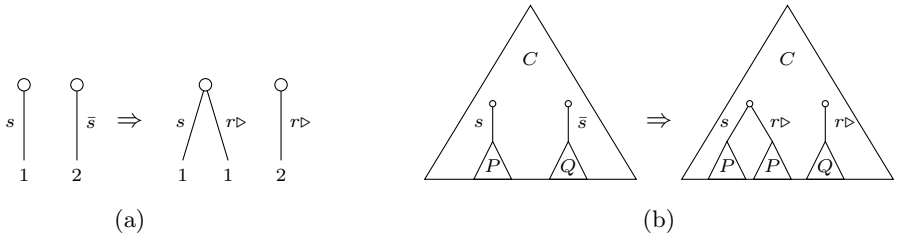
$$C[s.P, \bar{s}.Q] \Rightarrow C[s.P | r \triangleright P, r \triangleright Q] \qquad (r \text{ fresh}) \qquad \text{(ServiceSync)}$$

one for each suitable context $C[\__1, \__2]$ with two holes. Actually, the same semantics can be readily defined using a single wide reaction rule:

$$\langle s.x_1, \bar{s}.x_2\rangle \Rightarrow \langle s.x_1 | r \triangleright x_1, r \triangleright x_2\rangle \qquad (r \text{ fresh}) \qquad \text{(WideServiceSync)}$$

whilst $C[\__1, \__2]$ is the context where the redex $\langle s.x_1, \bar{s}.x_2\rangle$ is found and P, Q are the parameters the redex is instantiated to. In fact, the redex and the reactum are forests of two trees each, and the rule is rendered graphically as in Figure 1(a).

However, this added expressivity comes at a price: the problem of matching a wide redex within an agent is not as easy as the usual term matching. In this paper, we address precisely this problem, which we call the *forest pattern matching*: given a forest $\boldsymbol{l}(\boldsymbol{x})$ (the *pattern*) and an unordered tree a (the *target*), how to match each of the trees of the pattern within the target (with no overlaps), singling out the subtrees $\boldsymbol{d}$ that instantiate the parameters for the pattern?



(a) (b)

Fig. 1. A parametric rule (a), and its application as forest pattern matching (b).

As one may expect, forest pattern matching is NP-complete, as we will prove by means of a reduction from 3-SAT. However, this reduction points out the real source of time-complexity: the requirement that pattern trees are not overlapping in the target. The good news is that this combinatorial explosion does not depend on the size of the target tree, but only on the pattern width! As a consequence, once we fix a reactive system the exponential part becomes a constant factor, and the cost of each reduction step is polynomial on the size of the agent. Moreover, in most cases, rule width is small: e.g., for Ambients, CaSPiS, etc, it is ≤ 3. Therefore, our results prove that wide reaction rules are feasible for many calculi and models for global computation, and the implementation of general abstract machines for calculi for distributed systems, like [16], is viable.

The rest of this paper is structured as follows. In Section 2 we define formally the forest matching problem, which we show to be NP-complete in Section 3. In Section 4 we address this issue using Downey and Fellows' parameterized complexity theory [10] and indeed in Section 5 we provide an algorithm for it. As a side result, we introduce the new *rainbow antichain* problem, which is NP-complete but fixed-parameter tractable. Final remarks are in Section 6.

2 Labeled Trees, Forest Patterns, and Matches

In this section we define the forest pattern matching problem with no overlaps. As a first step, we define edge-labeled unordered trees, adopting the syntax of ambient calculus without actions [6], and extending it to (linear) context trees.

Let m, n range over an enumerable set Λ of labels, and x, y, z over an enumerable set Ξ of variables. Finite sets of variables are ranged over by X, Y, Z. The set of terms is the set of labeled context trees, finitely branching and of finite depth, where variables are interpreted as leaves where other trees can be grafted. We denote by $T(X), S(X)$ trees whose variables are in X. The syntax of these trees is defined by the following grammar.

Syntax of context trees

$T(X) ::=$	$\mathbf{0}$	empty tree
	x	leaf, $x \in X$
	$m[\,T(X)\,]$	labeled tree
	$T(Y) \mid T'(Z)$	siblings, where $X = Y \uplus Z$

We often abbreviate $m[\,\mathbf{0}\,]$ as $m[\,]$, and $T(X)$ as T. We assume that "|" associates to the right, i.e. $T \mid T' \mid T''$ is read $T \mid (T' \mid T'')$. Let $lab(T) \subset \Lambda$ be the set of node labels in T, and $vars(T) \subset \Xi$ be the set of the variables occurring in T (obviously, $vars(T(X)) \subseteq X$). We say that a term T is *ground* when $vars(T) = \emptyset$.

The intuitive interpretation of terms T as *unordered* trees induces an equivalence $T \equiv T'$ which is the minimal congruence that includes the commutative monoidal laws for | and $\mathbf{0}$. This relation, similar to ambient calculus congruence, can be axiomatized as follows.

Structural congruence on context trees

$$\frac{}{T \equiv T}\ (\mathsf{refl}) \qquad \frac{T \equiv T'}{T' \equiv T}\ (\mathsf{symm}) \qquad \frac{T \equiv T' \qquad T' \equiv T''}{T \equiv T''}\ (\mathsf{trans})$$

$$\frac{T \equiv T'}{T \mid T'' \equiv T' \mid T''}\ (\mathsf{sibl}) \qquad \frac{T \equiv T'}{m[T] \equiv m[T']}\ (\mathsf{rooting})$$

$$\frac{}{T \mid T' \equiv T' \mid T}\ (\mathsf{comm}) \qquad \frac{}{T \mid (T' \mid T'') \equiv (T \mid T') \mid T''}\ (\mathsf{assoc}) \qquad \frac{}{T \mid \mathbf{0} \equiv T}\ (\mathsf{nil})$$

The axiomatization of structural congruence is adequate with respect to the semantic for unordered trees: $T \equiv T'$ iff T and T' represent the same tree structure (obviously, where siblings are not ordered). Moreover, if $T \equiv T'$ then $lab(T) = lab(T')$ and $vars(T) = vars(T')$.

Given two tree terms $T(X)$, $S(Y)$ with X, Y disjoint, we define term substitution, written $T\{S/x\}$, as usual: the occurrence x in T is replaced by the term S. For $x \in vars(T)$, $vars(T\{S/x\}) = (vars(T) \setminus \{x\}) \cup vars(S)$. Simultaneous substitution $T\{S_1/x_1, \ldots, S_k/x_k\}$ is defined by the substitution composition $T\{S_1/x_1\} \cdots \{S_k/x_k\}$, where $x_1, \ldots, x_n$ are supposed to be pairwise distinct; we denote it by $T\{\boldsymbol{S}/\boldsymbol{x}\}$.

Lemma 1. *If $S_i \equiv S_i'$ for $i \in \{1, \ldots, k\}$, then $T\{\boldsymbol{S}/\boldsymbol{x}\} \equiv T\{\boldsymbol{S'}/\boldsymbol{x}\}$.*

Intuitively, given a tree list $S = S_1, \ldots, S_n$, called a "pattern", searching for a (sub-)match of S in a tree T means to find an occurrence of each $S_1, \ldots, S_n$ within T, without overlaps and possibly by instantiating variables in S_i. This means that we have to decompose T in a subtree C where all S_i can be grafted, and a list of subtrees to be grafted to the leaves of S_i. Formally:

Definition 2. *A* forest matching instance*, denoted by $T \succeq \boldsymbol{S}$, is given by a tree T (*target*), and a list of trees $\boldsymbol{S}(X) = S_1(X_1), \ldots, S_n(X_n)$ (*pattern*) where X_i are all disjoint and $X = \cup_{i=1}^{n} X_i$. We say that $\boldsymbol{S}(X)$* matches in *$T(Y)$ if for some* context *$C(Z)$ and* parameters *$\boldsymbol{D} = D_1, \ldots, D_m$,*

$$T \equiv \big(C\{\boldsymbol{S}/\boldsymbol{z}\}\big)\{\boldsymbol{D}/\boldsymbol{x}\}\,, \qquad \text{where } z_1, \ldots, z_n \in Z \text{ and } x_1, \ldots, x_m \in X\,.$$

A match for $T \succeq \boldsymbol{S}$ is denoted by $C, \boldsymbol{D} \models T \succeq \boldsymbol{S}$, and we write $\models T \succeq \boldsymbol{S}$ if $C, \boldsymbol{D} \models T \succeq \boldsymbol{S}$ for some C, $\boldsymbol{D}$.

Proposition 3. *If* $\models T \succeq \boldsymbol{S}$ *and* $\models S_i \succeq \boldsymbol{Q}$*, for some* $1 \leq i \leq n$ *and* $n = |\boldsymbol{S}|$*, then* $\models T \succeq \boldsymbol{Q}$*; and in particular* $\models T \succeq S_1, \ldots, S_{i-1}, \boldsymbol{Q}, S_{i+1}, \ldots, S_n$*.*

Proof. We have to prove that if $\models T \succeq \boldsymbol{S}$ and $\models S_i \succeq \boldsymbol{Q}$, for some $1 \leq i \leq n$ and $n = |\boldsymbol{S}|$, then $\models T \succeq S_1, \ldots, S_{i-1}, \boldsymbol{Q}, S_{i+1}, \ldots, S_n$.

Since $\models T \succeq \boldsymbol{S}$ and $\models S_i \succeq \boldsymbol{Q}$ there exist contexts C, C' and parameters $\boldsymbol{D}$, and $\boldsymbol{D'}$ such that

$$\begin{aligned} T &\equiv (C\{S_1/z_1, \ldots, S_n/z_n\})\{\boldsymbol{D}/\boldsymbol{x}\} && \text{for some } z_1, \ldots, z_n \in \mathit{vars}(C) \\ S_i &\equiv (C'\{\boldsymbol{Q}/\boldsymbol{z'}\})\{\boldsymbol{D'}/\boldsymbol{x'}\} && \text{for some } z'_1, \ldots, z'_m \in \mathit{vars}(C') \end{aligned}$$

Without loss of generality, suppose $\{z_1, \ldots, z_n\}$ disjoint from $\{z'_1, \ldots, z'_m\}$, otherwise a variable renaming can be applied. Now, by an easy replacement of S_i and some rearrangements on the context and parameters, we obtain

$$\begin{aligned} T &\equiv (C\{S_1/z_1, \ldots, S_i/z_i, \ldots, S_n/z_n\})\{\boldsymbol{D}/\boldsymbol{x}\} \\ &\equiv (C\{S_1/z_1, \ldots, (C'\{\boldsymbol{Q}/\boldsymbol{z'}\})\{\boldsymbol{D'}/\boldsymbol{x'}\}/z_i, \ldots, S_n/z_n\})\{\boldsymbol{D}/\boldsymbol{z}\} \\ &\equiv ((C\{C'/z_i\})\{S_1/z_1, \ldots, \boldsymbol{Q}/\boldsymbol{z'}, \ldots, S_n/z_n\})\{\boldsymbol{D}, \boldsymbol{D'}/\boldsymbol{x}, \boldsymbol{x'}\}\,. \end{aligned}$$

This means that $(C\{C'/z_i\}, (\boldsymbol{D}, \boldsymbol{D'}))$ is a match for $S_1, \ldots, S_{i-1}, \boldsymbol{Q}, S_{i+1}, \ldots, S_n$ in T, that is, $\models T \succeq S_1, \ldots, S_{i-1}, \boldsymbol{Q}, S_{i+1}, \ldots, S_n$.

Similarly, we can prove that $(C\{S_i/z_1, \ldots, C'/z_i, \ldots, S_n/z_n\}, \boldsymbol{D'})$ is a match for $\boldsymbol{Q}$ in T, hence $\models T \succeq \boldsymbol{Q}$ holds too. □

Many tree term patterns give rise to trivial or redundant forest matchings. For example, the empty tree matches any target T as $T \equiv (T \mid x)\{\mathbf{0}/x\}$. Another example of trivial matching instance is when the tree pattern is a variable. Indeed, a variable x can be matched in any tree T (in fact, x has as many matches as the number of sub-terms in T). A typical situation of redundant matching instances occurs when the pattern has "unguarded" variables at the top level, e.g. it is of the form $x \mid R$. Intuitively, this pattern matches an occurrence of R "beside anything, possibly nothing". As an example, let $T = m[\,\mathbf{0}\,] \mid n[\,k[\,\mathbf{0}\,]\,]$ be the target and $S = x \mid m[\,\mathbf{0}\,]$ be the pattern. Then, we have three different matches, namely $(y, n[\,k[\,\mathbf{0}\,]\,])$, $(n[\,y\,], k[\,\mathbf{0}\,])$, and $(n[\,k[\,\mathbf{0}\,]\,], \mathbf{0})$), despite $m[\,\mathbf{0}\,]$ occurs only once in T. A similar situation happens also when sibling variables $x \mid y$ occurs in the pattern (note that they can replaced by a single variable z, because $(x \mid y)\{D_1/x, D_2/y\} = (z)\{D_1 \mid D_2/z\}$).

Interestingly, redundant matches can be avoided by restricting our attention to a particular class of patterns, which we call *solid* after [15].

Definition 4. *A pattern* $\boldsymbol{S}(X) = S_1(X_1), \ldots, S_n(X_n)$ *is* solid *if for* $1 \leq i \leq n$*:* $S_i \not\equiv \mathbf{0}$*, for no* $x \in X$ *and* S' *it is* $S_i \equiv x \mid S'$*, and no two variables* $x, y \in X_i$ *are siblings, that is,* $x \mid y$ *cannot occur in* S_i *(up to* $\equiv$*).*

We can prove that any matching instance can be reduced to a matching instance whose pattern is solid. To this end let we define the function *solid* over forest patterns (i.e., tree lists), which drops empty trees and unguarded variables, and collapses sibling variables in one:

Transformation into solid patterns

$$solid(\epsilon) = \epsilon \qquad solid(T, \boldsymbol{S}) = \begin{cases} solid(\boldsymbol{S}) & \text{if } T \equiv \mathbf{0} \\ solid(Q, \boldsymbol{S}) & \text{if } T \equiv x \mid Q \\ solid(sld(T), \boldsymbol{S}) & \text{otherwise} \end{cases}$$

$$sld(\mathbf{0}) = \mathbf{0} \qquad del(\mathbf{0}) = \mathbf{0}$$
$$sld(x \mid T) = x \mid del(T) \qquad del(x \mid T) = del(T)$$
$$sld(m[T] \mid S) = m[sld(T)] \mid sld(S) \qquad del(m[T] \mid S) = m[sld(T)] \mid del(S)$$

Solid patterns enjoy the following properties.

Proposition 5. *The following statements hold:*

(a) ***no empty trees:*** $\models T \succeq \mathbf{0}, \boldsymbol{S} \iff \models T \succeq \boldsymbol{S}$;
(b) ***no sibling variables:*** $\models T \succeq x \mid y \iff \models T \succeq x$;
(c) ***no unguarded variables:*** $\models T \succeq x \mid S \iff \models T \succeq S$.

Due to the above, solid patterns suffice for checking match existence.

Lemma 6. $\models T \succeq solid(T)$ *if and only if* $\models solid(T) \succeq T$.

Proof. It is an easy application of proposition 5 and proposition 3. In fact, proposition 3 ensures that it suffices to check $\models \boldsymbol{T} \succeq \boldsymbol{T'} \iff \models \boldsymbol{T'} \succeq \boldsymbol{T}$ for each equation $\boldsymbol{T} = \boldsymbol{T'}$ defining *solid*. This is just a straightforward application of (a), (b), (c) of Proposition 5. □

Theorem 7. $\models T \succeq solid(\boldsymbol{S})$ *if and only if* $\models T \succeq \boldsymbol{S}$.

Proof. It follows directly from lemma 6 and proposition 3. □

Actually, all matches against a pattern $\boldsymbol{S}$ can be obtained from matches against $solid(\boldsymbol{S})$.

3 NP-completeness of Forest Pattern Matching

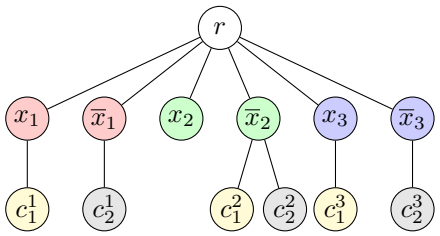
The main result of this section is that the problem of finding a pattern matching of a list of patterns $\boldsymbol{S} = S_1, \ldots, S_n$ in a target tree T is NP-complete. We show this by a reduction from 3-SAT [7]. Although the reduction can be done directly, we do it in two steps by introducing an intermediate problem, called RAINBOWANTICHAIN, that points out the actual source of time-complexity hardness.

An instance of RAINBOWANTICHAIN is a tree $\mathcal{T}(\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V}$ and edges $\mathcal{E}$, and a finite set $\mathcal{P}$ of colors, said *palette*. Some of the nodes in $\mathcal{T}$ have been colored with colors taken from the palette $\mathcal{P}$. Note that, the same color can be associated with different nodes, and each node can be associated with more than one color. RAINBOWANTICHAIN asks whether there exists a *rainbow antichain* $\mathcal{R} \subseteq \mathcal{V}$ in $\mathcal{T}$, i.e., a subset of nodes such that for no pair $u, v \in \mathcal{R}$ of distinct nodes u is an ancestor of v (hence, an *antichain*) and where each color $c \in \mathcal{P}$ has exactly one representative in $\mathcal{R}$ (hence, *colorful* w.r.t. $\mathcal{P}$).

Theorem 8. RainbowAntichain *is NP-complete.*

Proof. RainbowAntichain is in NP, since, given a set of nodes $\mathcal{R}$, checking whether $\mathcal{R}$ is a rainbow antichain for $\mathcal{T}$ can be done in polynomial time by a breadth-first visit of $\mathcal{T}$, and for each $v \in \mathcal{R}$ found, first increase the node counter nc, then the color counter $p[i]$ $(1 \leq i \leq |\mathcal{P}|)$ if v has color $c_i \in \mathcal{P}$. The check fails whether $nc > |\mathcal{P}|$ or $p[j] = 0$ for some $1 \leq j \leq |\mathcal{P}|$; it succeeds otherwise.

Let $C = \{c_1, \ldots, c_m\}$ be an instance of 3-Sat on variables $\{x_1, \ldots, x_n\}$. From C we define a colored tree $\mathcal{T}$ as follows. Let r be the root node which is left uncolored. For each variable x_i let x_i and $\overline{x}_i$ be child nodes of r, and color them with a fresh color c_{x_i}, distinct for each variable. For each clause $c_j \in C$, let c_j^1, c_j^2, c_j^3 be children nodes of l_i in T if c_j contains l_i as negated, and assign to each of them a fresh color c_{c_j}, distinct for each clause. An example of construction for $c_1 = (\overline{x}_1 \vee x_2 \vee \overline{x}_3)$, $c_2 = (x_1 \vee x_2 \vee x_3)$ is shown below.



Let φ be a truth assignment satisfying the formula C. By construction, selecting only literal nodes l_i which are satisfied by φ, we obtain a rainbow antichain $\mathcal{R}'$ in $\mathcal{T}$ for the palette $\{c_{x_i} : 1 \leq i \leq n\}$. Now, we extend $\mathcal{R}'$ to $\mathcal{R}$ adding all clause nodes which are not children of a element in $\mathcal{R}'$. Such $\mathcal{R}$ is clearly an antichain for $\mathcal{T}$, but we must ensure that is colorful and no more than one representative per color is taken. To do this, it suffices to prove that $\mathcal{R}$ is colorful, indeed if a color occurs more than once in $\mathcal{R}$ we remove the others. By hypothesis, each clause c_j is satisfied by φ, hence c_j has at least one literal l_i such that $\varphi(l_i) = \mathbf{T}$. By construction of $\mathcal{T}$, there exist a node c_j^k $(1 \leq k \leq 3)$ child of $\overline{l_i}$, hence already in $\mathcal{R}$. This holds for all clauses c_j, hence $\mathcal{R}$ is colorful.

Conversely, let $\mathcal{R}$ be a rainbow antichain for $\mathcal{T}$. Let $\varphi\colon \{x_1, \ldots, x_n\} \to \mathit{Bool}$ be defined by $\varphi(x_i) = \mathbf{T}$ if x_i is a node in $\mathcal{R}$, and $\varphi(x_i) = \mathbf{F}$ if x_i is a node not in $\mathcal{R}$. Since $\mathcal{R}$ has exactly one representative per color, no opposite literals are in $\mathcal{R}$, hence φ is a truth assignment for C. By colorfulness of $\mathcal{R}$, for all colors c_{c_j} $(1 \leq j \leq m)$ there exists a node $c_j^k \in \mathcal{R}$ $(1 \leq k \leq 3)$ such that c_j^k has color c_{c_j}. By construction of $\mathcal{T}$, each $c_j^k \in \mathcal{R}$ is a children of a literal node $l_i \notin \mathcal{R}$, and moreover the clause c_j contains $\overline{l_i}$. Since $l_i \notin \mathcal{R}$, by definition $\varphi(\overline{l_i}) = \mathbf{T}$, hence $\varphi(c_j) = \mathbf{T}$. This holds for all $1 \leq j \leq m$, hence φ satisfies C. □

It is easy to see that an instance $\mathcal{T}, \mathcal{P} = \{c_1, \ldots, c_n\}$ of RainbowAntichain can be reduced to a forest pattern matching problem, namely, the one that solves $\models T \succeq (c_1[\,x_1\,], \ldots, c_n[\,x_n\,])$, for a suitable tree term T defined upon $\mathcal{T}$. Thus:

Theorem 9. *The forest pattern matching problem is NP-complete.*

Theorem 8 states that the complexity hardness is merely due to finding a rainbow antichain in the given target, which corresponds to locate the list of trees of the pattern so that they are not in overlap in the target tree.

4 Tree Representation of Forest Pattern Matching

Despite the NP-completeness result from Theorem 9, in the next section we give a tractability result for the forest pattern matching problem, when the number of trees in the matching pattern is bounded by a (relatively small) constant h and their roots have at most k children, for some (relatively small) constant k. We propose a parameterized algorithm whose running time is $f(h,k) + O(n_s \cdot n_t^{3/2})$, for n_t and n_s the number of nodes in the target and pattern, respectively. This proves that the forest pattern matching is a fixed-parameter tractable problem (FPT) (we refer to [10] for the formal definition of this complexity class).

In presenting the algorithm we switch from edge-labeled tree terms to a more convenient node-labeled tree representations of them. This translation eases the description of the proposed algorithm and provides a closer connection between the concept of (labeled) subtree isomorphism and tree pattern matching.

Formally, a (rooted) node-labeled tree $\mathcal{T}(\mathcal{V}, \mathcal{E}, \mathit{label})$ is a triple, where $\mathcal{V}$ is the node set, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ the set of (oriented) edges, and $\mathit{label}\colon \mathcal{V} \to \Lambda^+ \times \{\mathit{op}, \mathit{cl}\}$ is a function associating with each node a label $m \in \Lambda^+ = \Lambda \uplus \{*\}$, and a flag *op* or *cl*. In the following we often abbreviate $\mathcal{T}(\mathcal{V}, \mathcal{E}, \mathit{label})$ with $\mathcal{T}$ and if $\mathit{label}(v) = (m, t)$ we say that v is m-labeled and *open* (resp. *closed*) if $t = \mathit{op}$ (resp. $t = \mathit{cl}$); $\mathit{root}(\mathcal{T})$ denotes the root node; $Ch(v)$ denotes the set of children of v; and $\mathcal{T}{\restriction}v$ denotes the subtree of $\mathcal{T}$ rooted at a node $v \in \mathcal{V}$. In the following, when $\mathcal{T}$ is the empty tree (i.e., $\mathcal{V} = \emptyset$), we assume $Ch(\mathit{root}(\mathcal{T})) = \emptyset$.

Definition 10. *A node-labeled tree $\mathcal{T}(\mathcal{V}, \mathcal{E}, \mathit{label})$ is said the* graphical representation *of an edge-labeled tree term $T \equiv m_1[T_1] \mid \cdots \mid m_n[T_n] \mid x_1 \mid \cdots \mid x_k$, for $n, k \geq 0$, if the following conditions hold:*

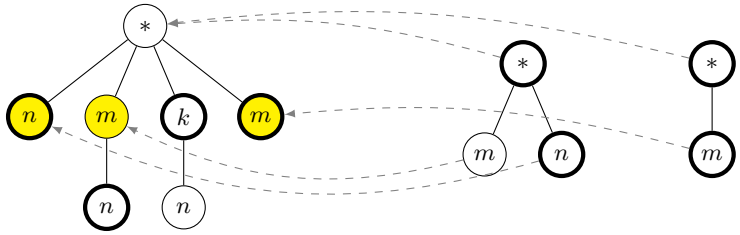
1. *if $n = 0$, then $\mathcal{T}$ is the empty tree ;*
2. *if $n > 0$, then*
   (a) *$\mathcal{V} = \{r, v_1, \ldots, v_n\} \cup \mathcal{V}'$, where $\{r, v_1, \ldots, v_n\} \cap \mathcal{V}' = \emptyset$;*
   (b) *$\mathcal{E} = \bigcup_i (\{(r, v_i)\} \cup \{(v_i, w) \mid w \in Ch(\mathit{root}(\mathcal{T}_i))\}) \cup \mathcal{E}'$;*
   (c) *$\mathit{label}\colon \mathcal{V} \to \Lambda^+ \times \{\mathit{op}, \mathit{cl}\}$ is such that, for all $v \in \mathcal{V}$*

$$\mathit{label}(v) = \begin{cases} (*, \mathit{op}) & \text{if } v = r \text{ and } k = 0 \\ (*, \mathit{cl}) & \text{if } v = r \text{ and } k > 0 \\ (m_i, t) & \text{if } v = v_i \text{ and } \mathit{label}(\mathit{root}(\mathcal{T}_i)) = (m, t) \\ \mathit{label}_i(v) & \text{if } v \in \mathcal{V}_i \end{cases}$$

where $\mathcal{T}_i(\mathcal{V}_i, \mathcal{E}_i, \mathit{label}_i)$ are the graphical tree representation of T_i ($1 \leq i \leq n$) with pairwise disjoint node sets, $\mathcal{V}' = \bigcup_i (\mathcal{V}_i \setminus \{\mathit{root}(\mathcal{T}_i)\})$, and $\mathcal{E}' = (\mathcal{V}' \times \mathcal{V}') \cap \bigcup_i \mathcal{E}_i$.

The graphical representation of a tree term is always rooted at a $*$-labeled node and converts m-labeled edges into m-labeled nodes, discarding variables. Note, however, that nodes in $\mathcal{T}$ are open iff they have a variable as a child in its tree term representation. In Figure 2 it is shown an example of translation into the graphical representation.

The following proposition relates the subtree isomorphism to the notion of tree pattern matching on terms, when the pattern is supposed to be solid.



$T = n[\,\mathbf{0}\,] \mid m[\,y_1 \mid n[\,\mathbf{0}\,]\,] \mid k[\,n[\,y_2\,]\,] \mid m[\,\mathbf{0}\,] \mid y_3 \qquad S_1 = m[\,x\,] \mid n[\,\mathbf{0}\,] \qquad S_2 = m[\,\mathbf{0}\,]$

Fig. 2. The forest pattern $\boldsymbol{S} = S_1, S_2$ has a match in T: for $C = z_1 \mid k[\,n[\,y_2\,]\,] \mid z_2 \mid y_3$ and $D = y_1 \mid n[\,\mathbf{0}\,]$, $T \equiv (C\{S_1/z_1, S_2/z_2\})\{D/x\}$. Bold-circled nodes are closed.

Proposition 11. *For a term T and a solid one T', where $\mathcal{T}(\mathcal{V}, \mathcal{E}, label)$ and $\mathcal{T}'(\mathcal{V}', \mathcal{E}', label')$ are their tree representations, respectively, then $\models T \succeq T'$ if and only if there exists $\mathcal{V}'' \subseteq \mathcal{V}$, where $|\mathcal{V}'| = |\mathcal{V}''|$, and $\rho\colon \mathcal{V}' \to \mathcal{V}''$ a one-to-one function such that*

1. *$(u, v) \in \mathcal{E}'$ iff $(\rho(u), \rho(v)) \in \mathcal{E}$;*
2. *if v is m-labeled then $\rho(v)$ is m-labeled, for $m \in \Lambda$;*
3. *if $v \in \mathcal{V}' \setminus \{root(\mathcal{T}')\}$ is closed then $\rho(v)$ is closed and $|Ch(v)| = |Ch(\rho(v))|$.*

Conditions (1–2) in Proposition 11 correspond the subtree isomorphism of $\mathcal{T}'$ in $\mathcal{T}$ with respect to Λ^+-labeled nodes (note that, $*$ acts as a wildcard label). Condition (3) is required in situations like the one that follows. Let $T = m[\,n[\,\mathbf{0}\,]\,]$ and $T' = m[\,\mathbf{0}\,]$, then T' has no match in T even though, considering their graphical tree representations, there exists ρ satisfying conditions (1) and (2).

Proposition 11 induces the definition of the following relation: $\rho \models \mathcal{T} \succeq \mathcal{T}'$ iff there exists ρ satisfying conditions (1–3). Obviously $\models T \succeq T'$ iff $\rho \models \mathcal{T} \succeq \mathcal{T}'$ and $\mathcal{T}$, $\mathcal{T}'$ are graphical representations for T, T', respectively.

Now, let us consider the forest pattern matching problem, that is, when the pattern is a list of arbitrary length $h \geq 0$.

Proposition 12. *Given a term T and a solid (forest) pattern $\boldsymbol{S} = S_1, \ldots, S_h$, where $\mathcal{T}$ and $\boldsymbol{\mathcal{S}} = \mathcal{S}_1, \ldots, \mathcal{S}_h$ are their tree representations (with disjoint node sets), then $\models T \succeq \boldsymbol{S}$ if and only if*

1. *$\rho_i \models \mathcal{T} \succeq \mathcal{S}_i$, for $1 \leq i \leq h$;*
2. *$\mathcal{R} = \{\rho_i(v) \mid v \in Ch(root(\mathcal{S}_i)), 1 \leq i \leq h\}$ is an antichain in $\mathcal{T}$.*

Condition (1) is obvious, and follows by Proposition 11. Condition (2) states that the children of each $\mathcal{S}_i$-root must be mapped by ρ_i to form an antichain in $\mathcal{T}$. This ensures that the mapping of trees in the pattern are not overlapping in $\mathcal{T}$. Note that, different roots of the pattern can be mapped to the same target node, and that the antichain condition must be satisfied by the roots' children nodes only (see Figure 2 for an example).

5 A Parameterized Algorithm

In this section, we provide a parameterized algorithm to solve the forest pattern matching problem. Let T be a term, $\boldsymbol{S} = S_1, \ldots, S_h$ a solid (forest) pattern, and $\mathcal{T} = (\mathcal{V}, \mathcal{E}, label)$, $\mathcal{S}_i = (\mathcal{V}_i, \mathcal{E}_i, label_i)$ be their respective tree representations, for $1 \leq i \leq h$, with pairwise disjoint node sets; according to the alternative characterization of Proposition 12, our algorithm will solve $\models \mathcal{T} \succeq \boldsymbol{\mathcal{S}}$, where the chosen parameters are $h = |\boldsymbol{\mathcal{S}}|$ and $k = \max_i |Ch(root(\mathcal{S}_i))|$.

The key idea is to find all possible matches of each $\mathcal{S}_i$ separately, identifying them by coloring the nodes in $\mathcal{T}$, and finally to search for a rainbow antichain. The proposed algorithm uses the so called *reduction to kernel size* technique [10] and works in three steps:

1. for each $\mathcal{S}_i$ in the pattern, we identify all possible mappings ρ_i satisfying $\rho_i \models \mathcal{T} \succeq \mathcal{S}_i$. These mappings correspond to tree matches and we identify them by coloring the nodes in $\mathcal{T}$: each $\mathcal{S}_i$ is associated with a color $f \in \mathcal{F}$, and nodes in $Ch(root(\mathcal{S}_i))$ with colors from the palette $\mathcal{P}_i$ (a color for each node). Palettes are supposed to be disjoint.
2. we reduce the size of the colored target tree $\mathcal{T}$, yielding a *reduced kernel* of size depending only on the parameters h and k.
3. we perform an exhaustive search for a rainbow antichain on palette $\bigcup_i \mathcal{P}_i$.

Coloring the target tree: By Proposition 11 we know that this corresponds to solving the subtree isomorphism problem for each $\mathcal{S}_i$ in the pattern and ensuring that the closedness property holds (i.e., condition (3) in Proposition 11). It is not hard to see that the Matula's algorithm [17] for the subtree isomorphism can be adapted to our aims. Let M be a Boolean matrix of size $n_s \times n_t$, where n_t and n_s are respectively the number of nodes of the target tree and of the pattern (the summation of each node set of the whole tree list). By dynamic programming on $\mathcal{T}$ and $\boldsymbol{\mathcal{S}}$ we can fill M as follows: for each node u in $\boldsymbol{\mathcal{S}}$ and node v in $\mathcal{T}$, $M[u, v] = \mathbf{T}$ if there exists an embedding (respecting node labeling and the closedness property) of $\boldsymbol{\mathcal{S}}\restriction u$ in $\mathcal{T}$ rooted at v, otherwise $M[u, v] = \mathbf{F}$ (see Matula [17] for details on how the matrix M is obtained).

From the matrix M we define the coloring functions for $\mathcal{T}$. Let $\mathcal{F}$ and $\mathcal{P}_i$, for $1 \leq i \leq h$, be disjoint palettes such that $|\mathcal{F}| = h$ and $|\mathcal{P}_i| = |Ch(root(\mathcal{S}_i))| \leq k$, and $\alpha\colon \{\mathcal{S}_1, \ldots, \mathcal{S}_h\} \to \mathcal{F}$ and $\beta_i\colon Ch(root(\mathcal{S}_i)) \to \mathcal{P}_i$ be bijections associating a color $f \in \mathcal{F}$ with each $\mathcal{S}_i$ in the pattern, and a color $p \in \mathcal{P}_i$ with each children of $root(\mathcal{S}_i)$. For each $v \in \mathcal{V}$, we define the sets $color_R(v) \subseteq \mathcal{F}$ and $color_i(v) \subseteq \mathcal{P}_i$, for $1 \leq i \leq h$ as follows:

$$\alpha(\mathcal{S}_i) \in color_R(v) \iff M[root(\mathcal{S}_i), v] \qquad \beta_i(u) \in color_i(v) \iff M[u, v]$$

Note that nodes may take color from different palettes, indeed a subtree of the target may have a match with more than one tree in the pattern.

Proposition 13. *If* $\alpha(\mathcal{S}_i) \in color_R(v)$ *then* $\bigcup_{u \in Ch(v)} color_i(u) = \mathcal{P}_i$.

The above implies that, if a node v in $\mathcal{T}$ is $\alpha(\mathcal{S}_i)$-colored, i.e., $\mathcal{S}_i$ has a match rooted at v, then there must exist $C \subseteq Ch(v)$ such that $|C| = |Ch(root(\mathcal{S}_i))|$ and, for all $u \in Ch(root(\mathcal{S}_i))$, $\mathcal{S}_i\restriction u$ has a match rooted at a node in C.

Reduction to kernel size: The reduction of $\mathcal{T}$ to kernel size consists in a decoloring procedure that aims at leaving as much nodes as possible completely uncolored in order to remove them from $\mathcal{T}$. Indeed, uncolored nodes have no influence in the detection of a possible rainbow antichain in $\mathcal{T}$.

Before starting with the description of the reduction, we need some technical definitions and notations. We say that a node is c-decolored if we remove c from all its color sets (note that $color_R$ and $color_i$ are disjoint, hence the set deletion of c influences only the corresponding color palette). By $\mathcal{T} \setminus v$ we denote the tree obtained from $\mathcal{T}$ removing the node v and such that the children of u are adopted by its parent (if u is the root node we just decolor it).

Definition 14. *Let $\mathcal{T}$ be a tree and u a node. We denote by* $\mathsf{fout}(u)$ *the* fan-out of u, *defined as* $\mathsf{fout}(u) = \sum_{v \in an(u)} |Ch(v)| - 1$, *where $an(v)$ is the set of all ancestors of v; and by* $\mathsf{fout}(\mathcal{T}) = \max_{v \in \mathcal{V}} \mathsf{fout}(v)$ *the maximal fan-out in $\mathcal{T}$.*

Intuitively, $\mathsf{fout}(u)$ is the out-degree of the whole path from u to the root of $\mathcal{T}$.

Lemma 15. *If v is uncolored and $\mathcal{T}$ admits a $\mathcal{P}$-rainbow antichain, then also $\mathcal{T} \setminus v$ has $\mathcal{P}$-rainbow antichain.*

Lemma 16. *If $\mathcal{T}$ has a $\mathcal{P}$-rainbow antichain, then it has one also when u is c-decolored, for color $c \in \mathcal{P}$, if one of the following conditions hold:*

(a) u is a (strict) ancestor of v, and both u, v are c-colored;
(b) $\mathcal{T}$ has only c-colored leaves and u is a leaf such that $\mathsf{fout}(u) \geq |\mathcal{P}|$.

Proof. (a) Let $\mathcal{R}$ be a rainbow antichain for $\mathcal{T}$ such that $u \in \mathcal{R}$. Since u belongs to $\mathcal{R}$, for some color $c_{\mathcal{R}} \in \mathcal{P}$ assigned to u, $\mathcal{R}$ must be rainbow on the palette $\mathcal{P}$. If we decolor u by c, there are two cases. If $c \neq c_{\mathcal{R}}$, $\mathcal{R}$ continues to be a rainbow antichain for $\mathcal{T}$, conversely, if $c = c_{\mathcal{R}}$, $\mathcal{R}$ is no more colorful on $\mathcal{P}$, since one of the representative of $\mathcal{P}$ lacks (i.e., c). By hypothesis, u has a c-colored descendant v. It is easy to see that $\mathcal{R}' = (\mathcal{R} \setminus \{u\}) \cup \{v\}$ is still an antichain and moreover it is colorful for $\mathcal{P}$.

(b) Let $\mathcal{T}$ be a colored tree on palette $\mathcal{P}$ such that, all its leaves are colored by $c \in \mathcal{P}$, and v is a leaf in $\mathcal{T}$ for which $\mathsf{fout}(v) \geq |\mathcal{P}|$. We want to prove that if $\mathcal{T}$ has a rainbow antichain, it continues to have one also if we c-decolor v. Let P be the path from the leaf v to the root of $\mathcal{T}$. To each outer-neighbour n_i ($1 \leq i \leq \mathsf{fout}(v)$) of P corresponds a subtree $\mathcal{T}\restriction n_i$ with all leaves colored by c, since $\mathcal{T}$ has only c-colored leaves. It is worth noting that all $\mathcal{T}\restriction n_i$ are not overlapping with each other, since $\bigcup_i \{n_i\}$ is an antichain for $\mathcal{T}$.

Suppose $\mathcal{R}$ be a rainbow antichain for $\mathcal{T}$ such that $v \in \mathcal{R}$. Since $v \in \mathcal{R}$, for some color $c_{\mathcal{R}} \in \mathcal{P}$ assigned to v, $\mathcal{R}$ must be rainbow on the palette $\mathcal{P}$. If we c-decolor v, there are two cases. If $c \neq c_{\mathcal{R}}$, $\mathcal{R}$ continues to be a rainbow antichain for $\mathcal{T}$, conversely, if $c = c_{\mathcal{R}}$, $\mathcal{R}$ is no more rainbow on $\mathcal{P}$, since one of the representative of $\mathcal{P}$ lacks. Note that $\mathcal{R}$, apart v, must reside in $\bigcup_i \mathcal{T}\restriction n_i$. Since $\mathsf{fout}(v) \geq |\mathcal{P}|$, there are more than $|\mathcal{P}|$ subtrees $\mathcal{T}\restriction n_i$ ($1 \leq i \leq \mathsf{fout}(v)$), hence there is no way to choose $|\mathcal{P}|$ distinct nodes from $\bigcup_i \mathcal{T}\restriction n_i$ such that each $\mathcal{T}\restriction n_i$ as at lest one of these nodes. Therefore, since each $\mathcal{T}\restriction n_i$ contains at least

one node colored by c (all leaves are c-colored!), we can substitute the node $v \in R$ with one of the leaf node in the "untouched" $\mathcal{T}\restriction n_i$, thus obtaining a new antichain where v is not choosen (hence v can be safely decolored). □

Applying (a) we c-decolor all nodes that have a c-colored descendant, and by Lemma 15 we remove all the nodes that are left uncolored. Note that, this procedure can be applied both on palette $\mathcal{F}$ and on palette $\mathcal{P}_i$, for $1 \leq i \leq h$. This reduction returns a tree where all paths do not have color repetitions, hence, by Proposition 13 its height is at most $2h$. Condition (b) induces another decoloring procedure. In fact, once the previous reduction is applied, nodes with the same color must form an antichain and, in particular for each $f \in \mathcal{F}$ we can apply (b) just ignoring paths from a leaf up to a f-colored node. Note that this time we do not apply the reduction on palettes $\mathcal{P}_i$'s.

Proposition 17. *If* $\mathsf{fout}(\mathcal{T}) \leq m$*, then* $\mathcal{T}$ *has at most* 2^m *leaves.*

Proof. The proof is by induction on $m \geq 0$. If $m = 0$, then $\mathsf{fout}(\mathcal{T}) = 0$, hence $\mathcal{T}$ must be a single path, hence it has exactly one leaf. Let $m > 0$, and $\mathcal{T}$ be a tree with $t > 0$ children under its root (the case when $t = 0$ is trivial). By inductive hypothesis, each subtree rooted at a child of the root have at most 2^{k-t+1} leaves, since their fan-out is at most $k - (t-1)$. Since there are t of those subtrees, the number of the leaves in $\mathcal{T}$ is at most $t \cdot 2^{k-t+1}$. We have $t \cdot 2^{k-t+1} = 2 \cdot \frac{t}{2^t} \cdot 2^k \leq 2^k$, since, for all $t > 0$, $\frac{t}{2^t} \leq \frac{1}{2}$. □

By Proposition 17, the reduced target tree have at most $2^{|\mathcal{F}|}$ (hence, 2^h) f-colored nodes, for each $f \in \mathcal{F}$. Note, however, that we do not have a bound on the total number of nodes in the reduced tree, indeed the reduction induced by Lemma 16 (b) is not applied on c-colored nodes, for $c \in \bigcup_i \mathcal{P}_i$. This problem is overcome just checking that for each color $f \in \mathcal{F}$, all f-colored nodes have no more than $|\bigcup_i \mathcal{P}_i|$ c-colored children, for $c \in \bigcup_i \mathcal{P}_i$. Since $|\bigcup_i \mathcal{P}_i| \leq h \cdot k$, we obtain a reduced tree $\mathcal{T}_{red}$ with at most $h\,(k+1) \cdot 2^h$ nodes.

Looking for rainbow antichains: What we actually need is the following for each node v in the reduced target tree: for each $X \subseteq \mathcal{F}$, determine whether the pattern trees corresponding to color in X can be mapped *simultaneously* in the subtree $\mathcal{T}_{red}\restriction v$. To compute this, we determine all the possible tuples $t = (c_1, \ldots, c_{|Ch(v)|})$ of colors associated to each child of v, then we check that for each $\alpha(S_i) \in X$, the tuple t contains $\mathcal{P}_i$. Since both $Ch(v)$ and $\bigcup_i \mathcal{P}_i$ have at most $h \cdot k$ elements, for each node v and subset X we need to check at most $(h \cdot k)^2$ tuples at a cost of $h \cdot k$ per tuple. We denote this by the predicate $N(v, X)$.

In order to determine whether there exists a rainbow antichain in the reduced target tree $\mathcal{T}$, we need to check that $A(\mathcal{T}_{red}, \mathcal{F})$ hold, where the predicate $A(\mathcal{T}, X)$, for $\mathcal{T}$, subtree of $\mathcal{T}_{red}$, and $X \subseteq \mathcal{F}$, is defined as follows:

$$A(\mathcal{T}, X) = N(v, X) \vee \bigvee_{Y \subseteq X} \Big(A(\mathcal{T}', Y) \wedge A(\mathcal{T}'', Y \setminus X) \Big),$$

where, $v = \mathit{root}(\mathcal{T})$, $\mathcal{T}' = \mathcal{T}\restriction u_1$ and $\mathcal{T}''$ is the tree obtained by collecting all $\mathcal{T}\restriction u_j$ under a fresh copy of the node v, for $Ch(v) = \{u_1, \ldots, u_m\}$ and $2 \leq j \leq m$.

Now, $A(\mathcal{T}, X)$ holds if and only if $\mathcal{T}$ admits a rainbow antichain $\mathcal{R}$ for the palette $\bigcup_{\alpha(\mathcal{S}_i)\in X} \mathcal{P}_i$. Indeed, the antichain is either a subset of the immediate children of root (in this case $N(root(\mathcal{T}), X)$ holds), or it is split in the subtrees of $\mathcal{T}$ (in this case the right part of the formula holds). A formal argument for this fact can be provided by a straightforward induction on the height of $\mathcal{T}$.

In order to calculate $A(\mathcal{T}, X)$ we must solve a *subset convolution* problem for each node in the reduced target tree. Each subset convolution can be calculated in time $O(h^2 \cdot 2^h)$, by means of the fast subset convolution algorithm of [3], hence we can check $A(\mathcal{T}_{red}, \mathcal{F})$ in time $O(h \cdot k)^3 + O(h^3(k+1) \cdot 2^{2h})$ using a dynamic programming algorithm working bottom-up on the structure of $\mathcal{T}_{red}$.

Complexity analysis of the algorithm: The coloring phase costs $O(n_s \cdot n_t^{3/2})$ where n_t and n_s are the number of nodes in the target and pattern, respectively, [17]. Note that while coloring the nodes from leaves up to the root, it can be easily performed the first decoloring step, just do not coloring nodes by colors already assigned to some descendant.

The second decoloring phase must be performed after the previous decoloring. This is both necessary for the correctness of the reduction, and useful to increase the node fan-outs. The decoloring, for each $f \in \mathcal{F}$, first calculates the fan-out of each f-colored node just performing a simple depth-first visit of the tree, then it decolors the nodes by other h depth-first visits, one for each color in $\mathcal{F}$. The overall cost of the reduction is linear in n_t.

The cost for checking the existence of rainbow antichains in $\mathcal{T}_{red}$ has been already shown to be in $O(h \cdot k)^3 + O(h^3(k+1) \cdot 2^{2h})$.

Concluding, the overall cost of the algorithm is $O(h^3(k+1) \cdot 2^{2h}) + O(n_s \cdot n_t^{3/2})$.

Notice that the proposed algorithm proves also that the forest pattern matching problem is fixed-parameter tractable even if we choose as parameter simply $K = |\bigcup_i Ch(\mathcal{S}_i)|$; indeed, in this case the upper bound would be $O(K^3 \cdot 2^{2K}) + O(n_s \cdot n_t^{3/2})$. We have preferred to consider the two parameters h, k, instead of the single K, because this yields a lower and more precise upper-bound.

6 Conclusions

In this paper we have considered the problem of matching a forest within an unordered tree, with no overlaps. This problem arises in many situations dealing with hierarchical structures; in particular, it covers the problem of matching the redex of wide reaction rules within processes. This case is particularly interesting in the definition and the implementation of the semantics of calculi for distributed and global computation.

We have shown that, although the problem is NP-complete in general, the combinatorial explosion depends only on the forest width. This parameter is usually fixed once we have chosen the reduction system (i.e., reaction rules do not change) and it is small (≤ 3), thus the exponential explosion is actually bounded and the problem is feasible. In fact, we have provided an algorithm that solves this problem, respecting these complexity bounds. Therefore, our results prove that wide reaction rules are feasible for many calculi for global computation.

As a side result of our proof techniques, we have singled out the new *rainbow antichain* problem, which is NP-complete but fixed-parameter tractable; we think that this problem can be useful also for the analysis and reductions of other problems about trees and forests.

Related work. Wide reaction systems can be seen as a particular case of *graph rewriting systems*. It is well-known that in general graph rewriting is NP-complete [9], as each step requires to solve a subgraph isomorphism problem. In this paper we have improved this situation in the case of wide reaction systems, by showing that subgraph isomorphism can be avoided in favour of the rainbow antichain problem—thus yielding a polynomial algorithm where the exponential part becomes a constant factor once we have fixed the set of reaction rules.

The problem of forest matching arises also in the case of *Bigraphical Reactive Systems*, i.e. wide reaction systems whose states are *bigraphs* [14,19]. This case is quite interesting because BRSs are intended to be a general meta-model for global computations. Several algorithms for bigraph matching have been proposed [8,21,18], but these solutions do not take advantage from the fact that combinatorial explosion depends only on redex widths, as we have shown here.

Future work. First, we plan to apply the results and algorithm presented in this paper to real calculi and frameworks. The cases of Bigraphical Reactive Systems, BioBigraphs [1] and Synchronized Hyperedge Replacement [11] are of particular interest. In these cases, we have to integrate forest pattern matching with sub(hyper)graph isomorphisms (needed to match e.g. the link part of bigraphs). We hope that the tractability results given in this paper will help to tame the complexity of subgraph isomorphism.

In [20], equational rewriting logics have been proposed as a framework for operational semantics. Comparing our algorithm with tools implementing rewriting logic (e.g., Maude) is not immediate, since rewriting logics do not directly support wide rewritings. A possible workaround is to encode wide reaction rules into standard ones by extending the syntax with a suitable "tensor product"; however, we expect our direct approach to outperform this indirect one. On the other hand, our results about wide reaction rules could be incorporated into tools like Maude, leading to a new and more expressive *wide rewriting logics*.

An interesting question is whether there are other possible reductions to be applied in the target tree in order to yield a smaller kernel instance. A positive result in this direction would provide a significant improvement of both time and space complexity upper bounds. At the moment, we know only that our problem does not fulfill the criteria in [4] that would imply the nonexistence of a polynomial-bounded kernel, so there is still hope.

Another interesting situation is that of wide reaction systems with *reaction rates*, like Stochastic Bigraphs [15]. These cases are of great interest in quantitative models of networks, biological systems, etc. Here, we are interested to pick out a single match among many possible matches of different rules, but still respecting rates and stochastic distributions. We think that it should be possible to derive a suitable *counting* algorithm from the one presented in this paper.

References

1. G. Bacci, D. Grohmann, and M. Miculan. Bigraphical models for protein and membrane interactions. In G. Ciobanu, editor, *Proc. MeCBIC*, volume 11 of *Electronic Proceedings in Theoretical Computer Science*, pages 3–18, 2009.
2. M. Bezem, J. W. Klop, and R. de Vrijer. *Term rewriting systems*. CUP, 2003.
3. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proc. STOC '07*, pages 67–74, 2007. ACM.
4. H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels (extended abstract). In *Proc. ICALP*, volume 5125 of *Lecture Notes in Computer Science*, pages 563–574. Springer, 2008.
5. M. Boreale, R. Bruni, R. De Nicola, and M. Loreti. Sessions and pipelines for structured service programming. In *Proc. FMOODS*, volume 5051 of *Lecture Notes in Computer Science*, pages 19–38. Springer, 2008.
6. L. Cardelli and A. D. Gordon. Mobile ambients. In *Proc FOSSACS '98*, pages 140–155. Springer, 1998.
7. S. A. Cook. The complexity of theorem-proving procedures. In *Proc. STOC '71*, pages 151–158, 1971. ACM.
8. T. C. Damgaard, A. J. Glenstrup, L. Birkedal, and R. Milner. An inductive characterization of matching in binding bigraphs. *Formal Aspects of Computing*, 25(2):257–288, 2013.
9. H. Dörr. *Efficient graph rewriting and its implementation*, volume 922 of *Lecture Notes in Computer Science*. Springer, 1995.
10. R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.
11. G. L. Ferrari, D. Hirsch, I. Lanese, U. Montanari, and E. Tuosto. Synchronised hyperedge replacement as a model for service oriented computing. In *Proc. FMCO*, volume 4111 of *Lecture Notes in Computer Science*, pages 22–43. Springer, 2005.
12. J. E. Hopcroft and R. E. Tarjan. A v^2 algorithm for determining isomorphism of planar graphs. *Inf. Process. Lett.*, 1(1):32–34, 1971.
13. J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proc. STOC '74*, pages 172–184, 1974. ACM.
14. O. H. Jensen and R. Milner. Bigraphs and transitions. In *Proc. POPL*, pages 38–49. ACM, 2003.
15. J. Krivine, R. Milner, and A. Troina. Stochastic bigraphs. In *Proc. MFPS*, volume 218 of *Electronic Notes in Theoretical Computer Science*, pages 73–96, 2008.
16. A. Mansutti, M. Miculan, and M. Peressotti. Towards distributed bigraphical reactive systems. In R. Echahed, A. Habel, and M. Mosbah, editors, *Proc. GCM'14*, page 45, 2014.
17. D. W. Matula. Subtree isomorphism in $O(n^{5/2})$. *Annals of Discrete Mathematics*, 2:91–106, 1978.
18. M. Miculan and M. Peressotti. A CSP implementation of the bigraph embedding problem. In *Proc. 1st International Workshop on Meta Models for Process Languages (MeMo)*, 2014.
19. R. Milner. *The Space and Motion of Communicating Agents*. CUP, 2009.
20. T.-F. Serbanuta, G. Rosu, and J. Meseguer. A rewriting logic approach to operational semantics. *Inf. Comput.*, 207(2):305–340, 2009.
21. M. Sevegnani, C. Unsworth, and M. Calder. A SAT based algorithm for the matching problem in bigraphs with sharing. Technical Report TR-2010-311, Department of Computer Science, University of Glasgow, 2010.

A Proofs of Technical Results

Proof (of Proposition 5). We prove each point separately.

(a, $\Longrightarrow$) Since $\models T \succeq \mathbf{0}, \boldsymbol{S}$, there exist a context C and parameters $\boldsymbol{D}$ such that $T \equiv (C\{\mathbf{0}/z, \boldsymbol{S}/\boldsymbol{z}\})\{\boldsymbol{D}/\boldsymbol{x}\}$ for some $\boldsymbol{z} = (z_1, \ldots, z_n)$ and z in $vars(C)$. It is simple to prove that $C\{\mathbf{0}/z, \boldsymbol{S}/\boldsymbol{z}\} \equiv C\{\mathbf{0}/z\}\{\boldsymbol{S}/\boldsymbol{z}\}$, hence, by associativity of substitution composition, $T \equiv ((C\{\mathbf{0}/z\})\{\boldsymbol{S}/\boldsymbol{z}\})\{\boldsymbol{D}/\boldsymbol{x}\}$, that is, $\models T \succeq \boldsymbol{S}$.
(a, $\Longleftarrow$) Since $\models T \succeq \boldsymbol{S}$, there exist a context C and parameters $\boldsymbol{D}$ such that $T \equiv (C\{\boldsymbol{S}/\boldsymbol{z}\})\{\boldsymbol{D}/\boldsymbol{x}\}$ for some $\boldsymbol{z} = (z_1, \ldots, z_n)$ in $vars(C)$. Observe that $C \equiv C \mid \mathbf{0} \equiv (C \mid z)\{\mathbf{0}/z\}$ for some $z \notin \{z_1, \ldots, z_n\}$. From this we have that $T \equiv ((C \mid z)\{z/\mathbf{0}, \boldsymbol{S}/\boldsymbol{z}\})\{\boldsymbol{D}/\boldsymbol{x}\}$, that is, $\models T \succeq \mathbf{0}, \boldsymbol{S}$.
(b, $\Longrightarrow$) Since $\models T \succeq x \mid y$, there exist a context C and parameters $\boldsymbol{D}$ such that $T \equiv (C\{x \mid y/z\})\{\boldsymbol{D}/\boldsymbol{x}\}$ for some $z \in vars(C)$. Since $C\{x \mid y/z\} \equiv C\{y \mid w/z\}\{w/x\}$ (for w fresh), by associativity of substitution composition, we get $T \equiv ((C\{y \mid w/z\})\{x/w\})\{\boldsymbol{D}/\boldsymbol{x}\}$, that is, $\models T \succeq x$.
(b, $\Longleftarrow$) Since $\models T \succeq x$, there exist a context C and parameters $\boldsymbol{D}$ such that $T \equiv (C\{x/z\})\{\boldsymbol{D}/\boldsymbol{x}\}$ for some $z \in vars(C)$. It is easy to prove that $C\{x/z\} \equiv C\{x \mid \mathbf{0}/z\} \equiv C\{x \mid y/z\}\{\mathbf{0}/y\}$ for y fresh. Now by associativity and from the freshness of y, we obtain $T \equiv (C\{x \mid y/z\})\{\mathbf{0}/y, \boldsymbol{D}/\boldsymbol{x}\}$, that is, $\models T \succeq x \mid y$.
(c) has the same proof of (b), just replace x in (b) with S. □

Proof (of Theorem 9). Given a match $(C, \boldsymbol{D})$ for $T \succeq \boldsymbol{S}$, checking that $T \equiv (C\{\boldsymbol{S}/\boldsymbol{z}\})\{\boldsymbol{D}/\boldsymbol{x}\}$ corresponds to a tree isomorphism test, which is in P [12,13].

Let a colored tree $\mathcal{T}$ and a palette $\mathcal{P} = \{c_1, \ldots, c_n\}$ be and instance of RainbowAntichain. Let us transform $\mathcal{T}$ into a tree term T as follows. If $\mathcal{T}$ is a single node v (a leaf) T is $m[\,\mathbf{0}\,]$, where $m = c$ if v has color c, otherwise $m = *$, a fresh name not in $\mathcal{P}$ denoting an uncolored node. If $\mathcal{T}$ has root r and $\mathcal{T}_1, \ldots, \mathcal{T}_k$ are the (children) subtrees of r, T is $m[\,T_1 \mid \cdots \mid T_k\,]$, where m is as above for r, and $T_1, \ldots, T_k$ are transformed trees of $\mathcal{T}_1, \ldots, \mathcal{T}_k$.

Suppose $(C, \boldsymbol{D})$ be a match for $T \succeq (c_1[\,x_1\,], \ldots, c_k[\,x_n\,])$. In C, each $c_i[\,x_i\,]$ is grafted into a variable $z_i \in vars(C)$. Since variables can appear in terms only as leaves, in the transformation $\mathcal{T}$ of T, we have found a rainbow antichain for $\mathcal{P}$, since the matching pattern has all the colors in $\mathcal{P}$ exacty once.

Assume that $\mathcal{T}$ has a rainbow antichain $\mathcal{R}$. In order to recover context C and parameters $\boldsymbol{D}$, which are a match for $T \succeq (c_1[\,x_1\,], \ldots, c_k[\,x_n\,])$, it suffices to apply the construction explained above with some adjustments: we obtain C applying the transformation from the root of T, but if a node in $\mathcal{R}$ is reached it is transformed by a fresh variable z_i ($1 \leq i \leq n$) one for each element in $\mathcal{R}$; D_j's are recovered applying the original transformation starting from the subtrees rooted at the children of nodes in $\mathcal{R}$. It is straightforward to prove that $T \equiv (C\{c_1[\,x_1\,]/z_1, \ldots, c_k[\,x_n\,]/z_n\})\{\boldsymbol{D}/\boldsymbol{x}\}$, for $\boldsymbol{x} = x_1, \ldots, x_n$. □