Deciding the Non-Emptiness of Attack Trees

Maxime Audinot*, Sophie Pinchinat, François Schwarzentruber, and Florence Wacheux

[1] Univ. Rennes/IRISA/CNRS
[2] ENS Rennes
[3] {maxime.audinot, sophie.pinchinat, florence.wacheux}@irisa.fr
[4] francois.schwarzentruber@ens-rennes.fr

Abstract. We define and study the decision problem of the *non-emptiness* of an attack tree. This decision problem reflects the natural question of knowing whether some attack scenario described by the tree can be realized in (a given model of) the system to defend. We establish accurate complexity bounds, ranging from NP-completeness for arbitrary trees down to NLOGSPACE-completeness for trees with no occurrence of the AND operator. Additionally, if the input system to defend has a succinct description, the non-emptiness problem becomes PSPACE-complete.

1 Introduction

Attack trees are one of the most prominent graphical models for security, originally proposed by [20]. They are intuitive and provide a readable description of the (possibly many) ways of attacking a critical system, thus enabling efficient communication between security experts and decision makers.

For about a decade, formal methods have been deployed to tame these models, with the perspective to develop all kinds of assistant tools for attack trees. The formal approaches range among attack tree quantitative analysis [12,1], system-based approaches to assist experts in their design [2,3], and automated generation of attack trees [15,9,17]. All of these approaches rely on solid semantics. To cite a few, there are the multi-set semantics [14], the series-parallel graph semantics [11], the linear logic semantics [8], and the path/trace semantics [2,3].

It is important to notice that the path semantics of attack trees provides a natural way of interpreting the tree as a set of attacking scenarios in the system to defend. Such semantics therefore relies not only on the description of the tree but also on a formal definition of the system. This formal definition should reflect the evolution of the system when attacked, in other words its operational semantics. For example, in the ATSyRA tool [18] or in the Treemaker tool [10], the experts specify a system in some Domain Specific Language, then this specification is compiled into a transition system whose states denote the system configurations and whose transitions describe the ability for an attacker to act on the system, hence to modify the current configuration.

Although this is not made formal here, we claim that most of the existing semantics of attack trees in the literature intrinsically "contain" such an operational view of attack

* grant from DGA, Bruz

trees. It is therefore essential for further tools development to investigate computational aspects in terms of relevant decision problems such tools will need to solve.

One of the basic decision problems we can think of is addressed in the setting of the path semantics, and is called the *non-emptiness of an attack tree*, where the issue is to decide whether the tree describes a non-empty set of attacks on a given system or not. If the answer is no, then the expert is done because her attack tree describes ways of attacking that cannot be implemented by an attacker, meaning that the system is safe. Otherwise, the expert is informed that the system is vulnerable, and should carry on with her analysis. To our knowledge, there is currently no result regarding this question.

In this paper, we formalize this non-emptiness decision problem, establish tight computational complexity bounds, and discuss the impact of these results for tools development. More precisely, we show that:

1. For arbitrary attack trees, namely with no restrictions on the operators used in their description[5], this problem is NP-complete (Theorem 1.);
2. Additionally to this general result, we consider the subclass of so-called *AND-free* attack trees, by disallowing the `AND` operator. For this subclass, we exhibit a polynomial-time algorithm to solve the non-emptiness problem (Theorem 4), and show that this restricted decision problem is NLOGSPACE-complete (Theorem 5);
3. Finally, we consider a variant of the non-emptiness decision problem where the input system has a symbolic presentation, as it is the case in most practical applications (see for example the tool ATSyRA [18]). We argue that the price to pay for this succinct way of specifying the system yields a PSPACE-complete complexity (Theorem 6).

The paper is organized as follows. We start by recalling the definition of transition systems in Section 2, and the central notions of concatenation and parallel decomposition of formal finite words needed to define the path semantics of attack trees in Section 3. Attack trees are introduced in Section 4, as well as the formal definition of the non-emptiness decision problem. Section 5 is dedicated to the non-emptiness problem for arbitrary attack trees, while Section 6 focuses on the subclass of `AND`-free attack trees. In Section 7, we discuss the case of symbolic transition systems, and conclude the contribution by pointing out some future work in Section 8.

2 Transition Systems

Let $Prop = \{\iota, \iota_1, \ldots, \gamma, \gamma_1 \ldots\}$ be a countable set of atomic propositions.

Definition 1. *A* labeled transition system *over* $Prop$ *is a structure* $\mathcal{S} = (S, \to, \lambda)$*, where* S *is a finite set of* states *(whose typical elements are* $s, s', s_0, s_1, \ldots$*);* $\to \subseteq S \times S$ *is the* transition relation*, and we write* $s \to s'$ *instead of* $(s, s') \in \to$*; and* $\lambda : Prop \to 2^S$ *is the* valuation *function that assigns a set of propositions to states.*

The size *of* $\mathcal{S}$ *is* $|\mathcal{S}| := |S| + |\to|$.

[5] operators one can find in the dedicated literature, see Definition 5

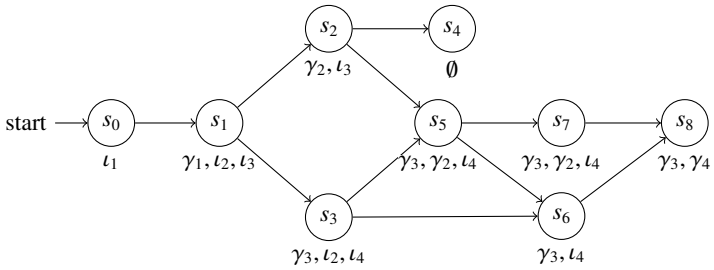


Fig. 1: A labeled transition system.

An example of labeled transition system with nine states $\{s_0, \ldots, s_8\}$ is depicted in Figure 1. In such structures, paths are central objects as they represent the dynamic of the system.

Definition 2. *A* path *of $\mathcal{S}$ is a sequence $\pi = s_0 \ldots s_n$ of states of $\mathcal{S}$, such that $n \geq 0$ and $s_i \rightarrow s_{i+1}$, for every $i < n$. The set of paths of $\mathcal{S}$ is denoted $\Pi(\mathcal{S})$.*

In the following, we write $s \rightarrow^* s'$ whenever there exists a path $\pi = s_0 \ldots s_n$ with $s = s_0$ and $s' = s_n$.

We consider two notions on paths, namely *concatenation* and *parallel decomposition*, that will serve us to define the path semantics of attack trees. Because paths can be seen as finite words, i.e. finite sequence of states, we define these notions in the abstract setting of words.

3 Concatenation and Parallel Decomposition

We write $w(i)$ for the $(i+1)$-th letter of the word w, so that letter positions in words start at 0. Also, let $|w|$ be the *size* of w, so that $|w| - 1$ is its last letter position. We also write $w.first$ and $w.last$ for $w(0)$ and $w(|w| - 1)$ respectively, and for $[k, l] \subseteq [0, |w| - 1]$, we write $w[k, l] := w(k) \ldots w(l)$. A *factor* of a word w is a word w', such that $w[k, l] = w'$ for some $[k, l] \subseteq [0, |w| - 1]$, and we call interval $[k, l]$ an *anchoring* of w' in w; note w' may have several anchorings in w.

We now introduce the *concatenation of words* (Definition 3) and the *parallel decompositions of a word* (Definition 4). The concatenation w of two words w_1 and w_2 is similar to the usual notion of concatenation except that the last letter of w_1 and the first letter w_2 which should be the same are merged. Figure 2 shows the concatenation of words $s_0 s_2 s_7 s_1$ and $s_1 s_4 s_6$.

Definition 3 (Concatenation). *Let w_1, w_2 be two words of respective sizes n_1 and n_2 and such that $w_1.last = w_2.first$. The* concatenation *of w_1 and w_2 , denoted by $w_1 \cdot w_2$, is the word of size $n_1 + n_2 - 1$, where $w[0, n_1 - 1] = w_1$ and $w[n_1, n_1 + n_2 - 1] = w_2$. We naturally extend the definition of concatenation to sets of words: for two sets of words W_1 and W_2, we let $W_1 \cdot W_2 := \{w_1 \cdot w_2 \mid w_1 \in W_1 \text{ and } w_2 \in W_2\}$.*

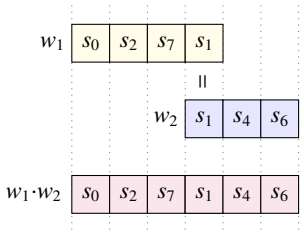


Fig. 2: Concatenation of words $s_0s_2s_7s_1$ and $s_1s_4s_6$.

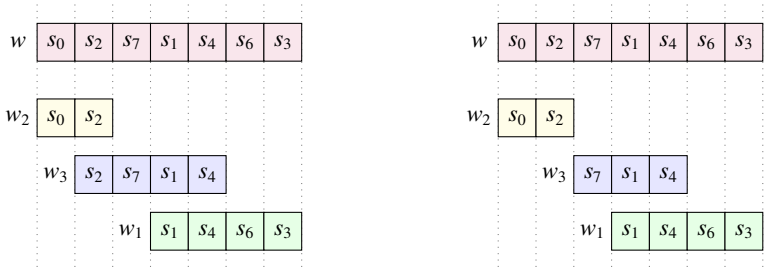
Intuitively, a parallel decomposition of a word w is the choice of a finite set of factors that entirely covers w. Figure 3a shows a possible parallel decomposition of the word $s_0s_2s_7s_1s_4s_6s_3$.

Definition 4 (Parallel decompositions of a word). *A set of words* $\{w_1, \ldots, w_n\}$ *is a* parallel decomposition *of a word* w *whenever the following holds.*

1. *For every* $i \in [1, n]$*, the word* w_i *is a factor of* w *at some anchoring* $[k_i, l_i]$*;*
2. *For every* $j \in [0, |w| - 2]$*,* $[j, j + 1] \subseteq [k_i, l_i]$*, for some* $i \in [1, n]$*.*

The intervals $[k_i, l_i]$ *form a* covering *of* $[0, |w| - 1]$*.*

Notice that our notion of covering is stronger than the classic notion of interval covering which requires that the union of intervals $[k_i, l_i]$ matches $[0, |w| - 1]$. Indeed, Point 2 of Definition 4 requires that each 2-size factor of w is also a factor of some of the words w_i. In particular, the three words w_1, w_2, w_3 as chosen in Figure 3b do not form a parallel decomposition of word $s_0s_2s_7s_1s_4s_6s_3$, since the 2-size word s_2s_7 is not a factor of any of these three words.



(a) Examples of words that form a parallel decomposition.

(b) Example of words that do not form a decomposition.

Fig. 3: Decomposition of word $s_0s_2s_7s_1s_4s_6s_3$.

We recall that $Prop = \{\iota, \iota_1, \ldots, \gamma, \gamma_1 \ldots\}$ is a countable set of atomic propositions, and we now define attack trees.

4 Attack Trees

In our formal setting, attack trees are finite labeled trees whose leaves are labeled by a pair $\langle \iota, \gamma \rangle$, where $\iota, \gamma \in Prop$ and whose internal nodes (non-leaves) are labeled by either symbol `OR`, symbol `SAND` (sequential and) or symbol `AND`. In our setting, and in most existing approaches in the literature, such labels correspond respectively to the union, the concatenation and the parallel decomposition of sets of paths (see Definition 6). W.l.o.g., we suppose that `OR`-nodes and `SAND`-nodes are binary, i.e. their nodes have exactly two children, since the corresponding semantics is associative (Definition 6). Figure 4 shows an attack tree with 4 leaves and 3 internal nodes.

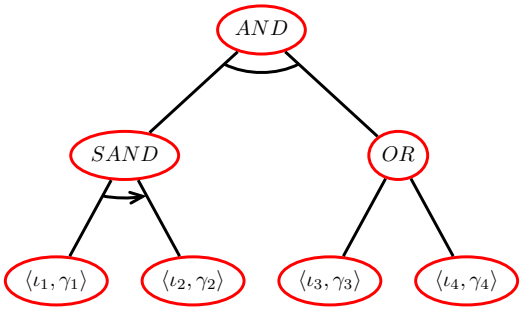


Fig. 4: Example of an attack tree.

Definition 5 (Attack tree). *An* attack tree *is defined by induction as follows.*

1. *A leaf labeled by a pair of propositions* $\langle \iota, \gamma \rangle \in Prop \times Prop$ *is an attack tree;*
2. *Given two attack trees* τ_1 *and* τ_2*, one can form the attack trees* $\mathtt{OR}(\tau_1, \tau_2)$ *and* $\mathtt{SAND}(\tau_1, \tau_2)$*;*
3. *Given a finite sequence* $\tau_1, \tau_2 \ldots, \tau_n$ *of attack trees, one can form the attack tree* $\mathtt{AND}(\tau_1, \ldots, \tau_n)$.

An attack tree τ is *`AND`-free* if it is built only by means of Rules 1. and 2. of Definition 5. We will refer to a pair $\langle \iota, \gamma \rangle$ of propositions labeling the leaves of attack trees as a *reachability goal*, and to propositions ι and γ as the *precondition* and the *postcondition* of this reachability goal $\langle \iota, \gamma \rangle$ respectively.

An attack tree τ is interpreted in a transition system $\mathcal{S}$ as a set $[\![\tau]\!]^{\mathcal{S}}$ of paths in $\mathcal{S}$.

Definition 6 (Path semantics). *The* path semantics *of* τ *in a transition system* $\mathcal{S}$ *is the set* $[\![\tau]\!]^{\mathcal{S}} \subseteq \Pi(\mathcal{S})$ *defined by induction as follows.*

- $[\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}} = \{\pi \in \Pi(\mathcal{S}) \mid \pi.first \in \lambda(\iota) \text{ and } \pi.last \in \lambda(\gamma)\}$
- $[\![\mathtt{OR}(\tau_1, \tau_2)]\!]^{\mathcal{S}} = [\![\tau_1]\!]^{\mathcal{S}} \cup [\![\tau_2]\!]^{\mathcal{S}}$
- $[\![\mathtt{SAND}(\tau_1, \tau_2)]\!]^{\mathcal{S}} = [\![\tau_1]\!]^{\mathcal{S}} . [\![\tau_2]\!]^{\mathcal{S}}$

– $[\![\texttt{AND}(\tau_1, \tau_2, \ldots, \tau_n)]\!]^{\mathcal{S}}$ *is the set of paths* π *of* $\mathcal{S}$ *that admit a parallel decomposition* $\{\pi_1, \pi_2, \ldots, \pi_n\}$ *with* $\pi_1 \in [\![\tau_1]\!]^{\mathcal{S}}, \ldots, \pi_n \in [\![\tau_n]\!]^{\mathcal{S}}$

Remark that the semantics for OR and SAND are associative because the corresponding operators on sets are. On the contrary, the semantics of AND is not associative, as shown in Example 1, and we therefore cannot restrict to a binary operator.

Example 1. Consider the system of Figure 1. The set $[\![\texttt{AND}(\langle \iota_1, \gamma_1 \rangle, \langle \iota_4, \gamma_4 \rangle, \langle \iota_2, \gamma_2 \rangle)]\!]^{\mathcal{S}}$ contains the four paths $s_0 s_1 s_2 s_5 s_7 s_8$, $s_0 s_1 s_3 s_5 s_7 s_8$, $s_0 s_1 s_2 s_5 s_6 s_8$, and $s_0 s_1 s_3 s_5 s_6 s_8$, but $[\![\texttt{AND}(\langle \iota_1, \gamma_1 \rangle, \langle \iota_4, \gamma_4 \rangle)]\!]^{\mathcal{S}} = \emptyset$ because there is no state that is both on a path in $[\![\langle \iota_1, \gamma_1 \rangle]\!]^{\mathcal{S}}$ and on a path in $[\![\langle \iota_4, \gamma_4 \rangle]\!]^{\mathcal{S}}$, so that the set $[\![\texttt{AND}(\texttt{AND}(\langle \iota_1, \gamma_1 \rangle, \langle \iota_4, \gamma_4 \rangle), \langle \iota_2, \gamma_2 \rangle)]\!]^{\mathcal{S}}$ is also empty.

Now that attack trees are defined, we turn to the central problem of this contribution.

5 The Non-emptiness Problem for Attack Trees

The *non-emptiness decision problem for attack trees*, that we shortly call Non-emptiness, is the following decision problem.

Non-emptiness: Given a system $\mathcal{S}$ and an attack tree τ, do we have $[\![\tau]\!]^{\mathcal{S}} \neq \emptyset$?

Theorem 1. Non-emptiness *is* NP*-complete.*

The rest of this section is dedicated to the proof of Theorem 1: we establish the NP upper bound in Subsection 5.1 (Theorem 2), and in Subsection 5.2, we resort to the result by [2] to obtain the NP lower bound (Theorem 3).

5.1 The Problem Non-emptiness is NP-easy

We provide a non-deterministic polynomial-time algorithm (Algorithm 2) that answers the problem Non-emptiness.

This algorithm, called $\texttt{nonemptiness}(\tau, \mathcal{S})$, relies on the *abstract semantics* (Definition 7) of attack trees, that consists only in sequences of *key states* that occur along paths of the path semantics. Notice that such sequences may not realize any path in $\mathcal{S}$, and will therefore be seen as words $w, w', \ldots \in S^*$.

Preliminarily to giving the definition of the abstract semantics of attack trees, we introduce the notion of *linearization* of a finite set of words: a linearization of words $w_1, \ldots, w_n$ is any word in the set $Lin(w_1, \ldots, w_n)$ defined as follows.

– If $n > 2$, then $Lin(w_1, \ldots, w_n) := Lin_2(Lin(w_1, \ldots, w_{n-1}), w_n)$;
– Otherwise $Lin(w_1, w_2) := Lin_2(w_1, w_2)$.

where $Lin_2(w_1, w_2)$ is defined inductively by: $Lin_2(\epsilon, \epsilon) := \epsilon$, and $Lin_2(sw, s'w') := s.Lin_2(w, s'w') \cup s'.Lin_2(s.w, w')$, to which we add $s.Lin_2(w, w')$ in case $s = s'$. For example, $Lin_2(s_2 s_7, s_2 s_4)$ contains $s_2 s_7 s_2 s_4$, $s_2 s_2 s_4 s_7$, and $s_2 s_7 s_4$.

Definition 7 (Abstract semantics). *The abstract semantics* $[\![\tau]\!]^{\mathcal{S}}_{abs} \subseteq S^*$ *is defined by induction over* τ*:*

- $[\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}}_{abs} = \{s_1 s_2 \mid s_1 \models \iota, s_2 \models \gamma\}$;
- $[\![\mathit{OR}(\tau_1, \tau_2)]\!]^{\mathcal{S}}_{abs} = [\![\tau_1]\!]^{\mathcal{S}}_{abs} \cup [\![\tau_2]\!]^{\mathcal{S}}_{abs}$;
- $[\![\mathit{SAND}(\tau_1, \tau_2)]\!]^{\mathcal{S}}_{abs} = [\![\tau_1]\!]^{\mathcal{S}}_{abs} \cdot [\![\tau_2]\!]^{\mathcal{S}}_{abs}$;
- $[\![\mathit{AND}(\tau_1, \ldots, \tau_n)]\!]^{\mathcal{S}}_{abs}$ *contains all linearizations* w *of some words* $w_1 \in [\![\tau_1]\!]^{\mathcal{S}}_{abs}, \ldots w_n \in [\![\tau_n]\!]^{\mathcal{S}}_{abs}$*, such that every letter occurrence of* w*, but* $w.first$ *and* $w.last$*, either is strictly between* $w_j.first$ *and* $w_j.last$ *for some* j*, or equals both* $w_j.first$ *and* $w_k.last$ *for some* $j \neq k$*.*

Intuitively, $[\![\tau]\!]^{\mathcal{S}}_{\text{abs}}$ contains *key states* in the sense that those are states satisfying the relevant pre/post-conditions appearing in the tree τ.

Example 2. Recall the labeled transition system of Figure 1. The word $s_2 s_7$ is in the set $[\![\langle \iota_3, \gamma_3 \rangle]\!]^{\mathcal{S}}_{\text{abs}}$ since s_2 and s_7 are states satisfying the precondition ι_3 and the postcondition γ_3 respectively, but $s_2 s_7$ is not a path in $\mathcal{S}$. Because $s_2 s_7$ is in $[\![\langle \iota_3, \gamma_3 \rangle]\!]^{\mathcal{S}}_{\text{abs}}$ and $s_7 s_8$ is in $[\![\langle \iota_4, \gamma_4 \rangle]\!]^{\mathcal{S}}_{\text{abs}}$, the word $s_2 s_7 s_8$ belongs to $[\![\texttt{SAND}(\langle \iota_2, \gamma_2 \rangle, \langle \iota_4, \gamma_4 \rangle)]\!]^{\mathcal{S}}_{\text{abs}}$.

Algorithm 2 $\texttt{nonemptiness}(\tau, \mathcal{S})$ consists in two steps:

(a) A call to the sub-routine $\texttt{guessAbstractPath}(\tau, \mathcal{S})$ (Algorithm 1) in order to guess a word w that plays the role of a *certificate* with key states;
(b) A check that w is "realizable" in $\mathcal{S}$, i.e. that there exists a path between any two consecutive key states occurring in w.

Step (a) amounts to executing Algorithm 1, which non-deterministically guesses a word in $[\![\tau]\!]^{\mathcal{S}}_{\text{abs}}$. In case of leaf tree $\langle \iota, \gamma \rangle$, the algorithm non-deterministically guesses two states s_1, s_2 and verifies the property that ι holds in s_1 and γ holds in s_2. If this property holds, Algorithm 1 returns the two-letter word $s_1 s_2$, otherwise it rejects the input. For a tree of the form $\texttt{OR}(\tau_1, \tau_2)$, the algorithm non-deterministically guesses one of the two sub-trees, i.e., some $i \in \{1, 2\}$, and then recursively executes $\texttt{guessAbstractPath}(\tau_i, \mathcal{S})$. For a tree of the form $\texttt{SAND}(\tau_1, \tau_2)$, the algorithm guesses two words w_1 and w_2 in $[\![\tau_1]\!]^{\mathcal{S}}_{\text{abs}}$ and $[\![\tau_2]\!]^{\mathcal{S}}_{\text{abs}}$ respectively, and returns the word $w_1 \cdot w_2$ whenever $w_1.last = w_2.first$, otherwise it rejects the input. For the case of a tree of the form $\texttt{AND}(\tau_1, \ldots, \tau_n)$, the algorithm guesses words w_i in $[\![\tau_i]\!]^{\mathcal{S}}_{\text{abs}}$, then it guesses a linearization of those, and finally verifies that this latter guess is indeed a linearization (see the **forall** loop in the last **case** of Algorithm 1).

The following proposition formally states the specification of Algorithm 1:

Proposition 1.
- *Any non-rejecting execution of Algorithm 1 returns a word in* $[\![\tau]\!]^{\mathcal{S}}_{abs}$.
- *Reciprocally, for every word in* $[\![\tau]\!]^{\mathcal{S}}_{abs}$*, there exists a non-rejecting execution of Algorithm 1 that returns this word.*

Proof. The proof can be conducted by induction on τ and is left to the reader.

Input: An attack tree τ and a transition system $\mathcal{S}$
Output: A word $w \in [\![\tau]\!]^{\mathcal{S}}_{\text{abs}}$
switch τ **do**
  case $\langle \iota, \gamma \rangle$ **do**
    guess $s_1, s_2 \in S$;
    check $s_1 \in \lambda(\iota)$ and $s_2 \in \lambda(\gamma)$;
    return $s_1 s_2$;
  end
  case $\mathit{OR}(\tau_1, \tau_2)$ **do**
    guess $i \in \{1, 2\}$;
    return `guessAbstractPath`$(\tau_i, \mathcal{S})$;
  end
  case $\mathit{SAND}(\tau_1, \tau_2)$ **do**
    w_1 := `guessAbstractPath`$(\tau_1, \mathcal{S})$;
    w_2 := `guessAbstractPath`$(\tau_2, \mathcal{S})$;
    check $w_1.last = w_2.first$;
    return $w_1 \cdot w_2$
  end
  case $\mathit{AND}(\tau_1, \ldots, \tau_n)$ **do**
    w_i := `guessAbstractPath`$(\tau_i, \mathcal{S})$ for each $1 \leq i \leq n$;
    guess w, a linearization of $w_1, \ldots, w_n$;
    forall *letters s of w except* $w.first$ *and* $w.last$ **do**
      check there exist $j, k \in [1, n]$ such that either s is strictly between $w_j.first$ and $w_j.last$ in w, or s equals both $w_j.first$ and $w_k.last$
    end
    return w;
  end
end

Algorithm 1: `guessAbstractPath`$(\tau, \mathcal{S})$.

Regarding Step (b) of Algorithm 2, the procedure consists in verifying that the word w resulting from Step (a) can be *realized by a path* in the system $\mathcal{S}$, in the sense that there exist sub-paths between every successive key states occurring in w (see Definition 8).

Definition 8. *Given a system $(S, \rightarrow, \lambda)$, a word $w = s_0 \dots s_n \in S^*$ is* realized by a path π in $\mathcal{S}$ *if $\pi = \pi_0 \cdot \ldots \cdot \pi_{n-1}$ for some π_i's that are paths from s_i to s_{i+1} in $\mathcal{S}$ respectively. Notice that $w.first = \pi.first$ and $w.last = \pi.last$. Note also that any factor of w is also realizable.*

Verifying that the word is realizable by a path uses the Boolean function $\texttt{reach}_{\mathcal{S}}$ whose specification is: given two states $s_1, s_2 \in S$, $\texttt{reach}_{\mathcal{S}}(s_1, s_2)$ is true iff there is a path from s_1 to s_2 in $\mathcal{S}$. It is well known that such a function can be implemented in polynomial time.

Input: An attack tree τ and a transition system $\mathcal{S}$
Output: Accept whenever $[\![\tau]\!]^{\mathcal{S}} \neq \emptyset$.
//Step (a)
$w :=$ `guessAbstractPath`$(\tau, \mathcal{S})$;
//Step (b)
foreach s_1, s_2 *successive in* w **do**
  check $\texttt{reach}_{\mathcal{S}}(s_1, s_2)$
end
accept

Algorithm 2: `nonemptiness`$(\tau, \mathcal{S})$.

The correctness of Algorithm 2 follows from Proposition 2:

Proposition 2. *The two following statements are equivalent:*

(i) There exists a word $w \in [\![\tau]\!]^{\mathcal{S}}_{abs}$ that can be realized by a path of $\mathcal{S}$;
(ii) $[\![\tau]\!]^{\mathcal{S}} \neq \emptyset$.

Proof. We show that (i) implies (ii) by establishing an inductive proof over τ that if $w \in [\![\tau]\!]^{\mathcal{S}}_{\text{abs}}$ can be realized by a path π of $\mathcal{S}$, then $\pi \in [\![\tau]\!]^{\mathcal{S}}$.

If $w \in [\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}}_{\text{abs}}$ then $w.first \models \iota$ and $w.last \models \gamma$, if w can be realized by some path π, then $w.first = \pi.first$ and $w.last = \pi.last$. One easily concludes that $\pi \in [\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}}$. If $w \in [\![\texttt{OR}(\tau_1, \tau_2)]\!]^{\mathcal{S}}_{\text{abs}}$, which by Definition 7, equals $[\![\tau_1]\!]^{\mathcal{S}}_{\text{abs}} \cup [\![\tau_2]\!]^{\mathcal{S}}_{\text{abs}}$, pick some i such that $w \in [\![\tau_i]\!]^{\mathcal{S}}_{\text{abs}}$. By induction hypothesis, we then get $\pi_i \in [\![\tau_i]\!]^{\mathcal{S}}$ that realizes w and because $[\![\tau_i]\!]^{\mathcal{S}} \subseteq [\![\texttt{OR}(\tau_1, \tau_2)]\!]^{\mathcal{S}}$, word w is realized by $\pi_i \in [\![\texttt{OR}(\tau_1, \tau_2)]\!]^{\mathcal{S}}$, which allows us to conclude. If $w \in [\![\texttt{SAND}(\tau_1, \tau_2)]\!]^{\mathcal{S}}_{\text{abs}}$, which by Definition 7, equals $[\![\tau_1]\!]^{\mathcal{S}}_{\text{abs}} \cdot [\![\tau_2]\!]^{\mathcal{S}}_{\text{abs}}$, then $w = w_1 \cdot w_2$, with $w_1 \in [\![\tau_1]\!]^{\mathcal{S}}_{\text{abs}}$ and $w_2 \in [\![\tau_2]\!]^{\mathcal{S}}_{\text{abs}}$. Since moreover w can be realized, so are its two factors w_1 and w_2, say by some paths π_1 and π_2. By induction hypothesis, $\pi_1 \in [\![\tau_1]\!]^{\mathcal{S}}$ and $\pi_2 \in [\![\tau_2]\!]^{\mathcal{S}}$. Now, $\pi_1.last = w_1.last = w_2.first = \pi_2.first$, word w is clearly realized by $\pi_1 \cdot \pi_2$ with $\pi \in [\![\texttt{SAND}(\tau_1, \tau_2)]\!]^{\mathcal{S}}$. The last case where $w \in [\![\texttt{AND}(\tau_1, \ldots, \tau_n)]\!]^{\mathcal{S}}_{\text{abs}}$ is tedious, and omitted here.

To show that (ii) implies (i), we establish by induction over τ that if $\pi \in [\![\tau]\!]^{\mathcal{S}}$, then there is a word $w \in [\![\tau]\!]^{\mathcal{S}}_{\text{abs}}$ that is realized by π.

Suppose $\pi \in [\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}}$, then clearly the word $(\pi.first)(\pi.last)$ is in $[\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}}_{\text{abs}}$, and is by construction realizable by π.

Suppose $\pi \in [\![\texttt{OR}(\tau_1, \tau_2)]\!]^{\mathcal{S}} = [\![\tau_1]\!]^{\mathcal{S}} \cup [\![\tau_2]\!]^{\mathcal{S}}$. Pick i such that $\pi \in [\![\tau_i]\!]^{\mathcal{S}}$. By induction hypothesis, there exists w that is realized by π and in $[\![\tau_i]\!]^{\mathcal{S}}_{\text{abs}} \subseteq [\![\texttt{OR}(\tau_1, \tau_2)]\!]^{\mathcal{S}}_{\text{abs}}$, which concludes the argument.

Suppose $\pi \in [\![\texttt{SAND}(\tau_1, \tau_2)]\!]^{\mathcal{S}}$. Pick $\pi_1 \in [\![\tau_1]\!]^{\mathcal{S}}$ and $\pi_2 \in [\![\tau_2]\!]^{\mathcal{S}}$ with $\pi = \pi_1 \cdot \pi_2$. By induction hypothesis, there is a word $w_1 \in [\![\tau_1]\!]^{\mathcal{S}}_{\text{abs}}$ that can be realized by π_1, and similarly, there is a word $w_2 \in [\![\tau_2]\!]^{\mathcal{S}}_{\text{abs}}$ that can be realized by π_2. Since $w_1.last = \pi_1.last = \pi_2.first = w_2.first$, the word $w = w_1 \cdot w_2$ is well defined, clearly belongs to $[\![\texttt{SAND}(\tau_1, \tau_2)]\!]^{\mathcal{S}}_{\text{abs}}$, and is realized by π.

The case where $\pi \in \texttt{AND}(\tau_1, \ldots, \tau_n)$ is tedious and left to the reader.

Proposition 3. *Algorithm 2 is non-deterministic and runs in polynomial time.*

Proof. Clearly Step (a) makes at most one call to Algorithm 1 (which is non-deterministic and runs in polynomial time, see just below) per each node of the input tree, so Step (a) runs in time linear in the size of the input. Step (b) executes a call to the polynomial-time algorithm *Reach* at most a number of times bounded by the size of the word output in Step (a) – hence a polynomial number.

Regarding the complexity of Algorithm 1, the guesses made are either some $i \in \{1, 2\}$, or a pair of states, or some linearization of a set of words. All those have a polynomial size because the first is constant sized, the second in logarithmic in the size of the input system $\mathcal{S}$, and any linearization has a size at most twice the number of leaves in the input tree.

This concludes the proof of Proposition 3.

By Proposition 2 and Proposition 3, we obtain:

Theorem 2. Non-emptiness $\in$ NP.

The next section completes the proof of Theorem 1.

5.2 The Problem Non-emptiness is NP-hard

We inherit from the result [2, Proposition 2] that can be rephrased as follows in our context.

Theorem 3. Non-emptiness *is* NP*-hard, even if we restrict to trees of the form* $\texttt{AND}(\langle \iota_1, \gamma_1 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)$.

The proof of Theorem 3 is based on a polynomial reduction from the propositional satisfiability problem to Non-emptiness for trees of the form $\texttt{AND}(\langle \iota_1, \gamma_1 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)$. Because the former is NP-hard [7], so is the latter, Non-emptiness.

We recall some basic vocabulary. Let $\{p_1, \ldots, p_r\}$ be a set of propositions. A *literal* ℓ is either a proposition p or its negation $\neg p$. A *clause* C is a disjunction of literals. The *propositional satisfiability problem* SAT is as follows.

Input: A set $\mathscr{C} = \{C_1, \ldots, C_m\}$ of clauses.
Output: Does there exist a valuation over $Prop$ that satisfies the set of clauses $\mathscr{C}$?

Consider $\mathscr{C} = \{C_1, \ldots, C_m\}$, an instance of SAT, and let $\{p_1, \ldots, p_r\}$ be an ordering of the set of propositions occurring in the clauses of $\mathscr{C}$. It is standard to write $|\mathscr{C}|$ for the cumulative sum of the clauses' size, where the size of a clause is the number of its literals. In the following, we denote by ℓ_i an occurrence of proposition p_i or $\neg p_i$.

We define the labeled transition system $\mathcal{S}_{\mathscr{C}} := (S_{\mathscr{C}}, \rightarrow_{\mathscr{C}}, \lambda_{\mathscr{C}})$ over the set of propositions $Prop = \{start, C_1, \ldots, C_m\}$, where $start$ is a fresh proposition, as follows:

- The set of states is $S_{\mathscr{C}} = \bigcup_{i=1}^{r} \{p_i, \neg p_i\} \cup \{init\}$, where $init$ is a fresh state;
- The transition relation is $\rightarrow_{\mathscr{C}} = \{(init, \ell_1)\} \cup \{(\ell_i, \ell_{i+1}) \mid 1 \leq i \leq r-1\}$;
- The labeling of states $\lambda_{\mathscr{C}} : \{start, C_1, \ldots, C_m\} \rightarrow 2^S$ is such that $\lambda_{\mathscr{C}}(start) = \{init\}$ and $\lambda_{\mathscr{C}}(C_i) = \{\ell \mid \ell \in C_i\}$ for every $1 \leq i \leq m$.

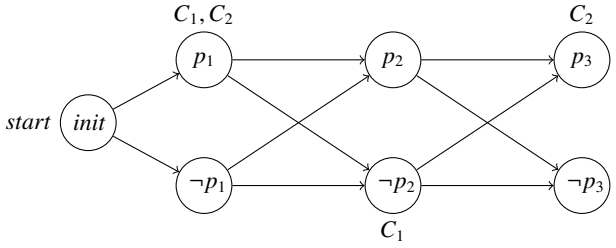


Fig. 5: The system $\mathcal{S}_{\{C_1,C_2\}}$ where $C_1 = p_1 \vee \neg p_2$ and $C_2 = p_1 \vee p_3$.

For example, the transition system corresponding to the set formed by the set of clauses $C_1 = p_1 \vee \neg p_2$ and $C_2 = p_1 \vee p_3$ is depicted in Figure 5.

We now let the attack tree $\tau_{\mathscr{C}} := \mathtt{AND}(\langle start, C_1\rangle, \langle start, C_2\rangle, \ldots, \langle start, C_m\rangle)$.

The reduction that we have described maps any instance $\mathscr{C} = \{C_1, \ldots, C_m\}$ of SAT to the instance $(\mathcal{S}_{\mathscr{C}}, \tau_{\mathscr{C}})$ of Non-emptiness. It is is trivially computable in polynomial time.

We now prove that $[\![\tau_{\mathscr{C}}]\!]^{\mathcal{S}_{\mathscr{C}}} \neq \emptyset$ if, and only if $\mathscr{C}$ is satisfiable.

($\Leftarrow$) Suppose that $\mathscr{C}$ is satisfiable. There exists a valuation over $Prop$ that satisfies a set of clauses $\mathscr{C}$. First, we consider the path that starts from $init$ and that follows the literals that are made true by this valuation. Second, we take the longest prefix π of that path that ends in a state labeled by C_j. As all C_i are satisfied by the valuation, all C_i appear on π, which shows $\pi \in [\![\tau_{\mathscr{C}}]\!]^{\mathcal{S}_{\mathscr{C}}}$.

($\Rightarrow$) Let $\pi \in [\![\tau_{\mathscr{C}}]\!]^{\mathcal{S}_{\mathscr{C}}}$. By definition of $\mathcal{S}_{\mathscr{C}}$, π cannot visit both a proposition and its negation. Therefore, π trivially denotes a partial valuation – that is completed by assigning false to all other propositions. Since $\pi \in [\![\tau_{\mathscr{C}}]\!]^{\mathcal{S}_{\mathscr{C}}}$, π visits a state labeled by C_i for every i, which shows that the valuation satisfies all the clauses. Hence, $\mathscr{C}$ is satisfiable.

6 The Non-emptiness Problem for AND-free Attack Trees

We here show that the complexity of deciding the non-emptiness of an attack tree boils down to NLOGSPACE if the input trees are AND-free. We write Non-emptiness$_{Af}$ for this restricted version of the problem.

For a start, we establish that Non-emptiness$_{Af}$ is in P (Theorem 4), and later in the section, we improve this bound by showing that Non-emptiness$_{Af}$ is NLOGSPACE-complete (Theorem 5).

Theorem 4. Non-emptiness$_{Af}$ *is in* P.

We prove Theorem 4 by developing the polynomial-time Algorithm 3 that answers Non-emptiness$_{Af}$. This algorithm amounts to verifying the non-emptiness of the set returned by the the divide-and-conquer Algorithm 5, namely the set $\texttt{pairs}(\tau, \mathcal{S})$ of pairs of states in $\mathcal{S}$ that are ends (first and last states) of some path in $[\![\tau]\!]^{\mathcal{S}}$.

Input: An AND-free attack tree τ and a transition system $\mathcal{S} = (S, \rightarrow, \lambda)$
Output: $[\![\tau]\!]^{\mathcal{S}} \neq \emptyset$?
return $\texttt{pairs}(\tau, \mathcal{S}) \neq \emptyset$

Algorithm 3: $\texttt{nonemptinessAf}(\tau, \mathcal{S})$.

Before detailing Algorithm 5, we recall the simple Algorithm 4 used for the base case of leaf trees. This latter algorithm computes in polynomial time the set $\texttt{ends}(\iota, \gamma, \mathcal{S})$ of pairs of states that end a given path in $[\![\tau]\!]^{\mathcal{S}}$. Algorithm 4 calls $ReachableFrom_{\mathcal{S}}(s)$ which computes the set of states reachable from s in $\mathcal{S}$, that is the set of states s' such that $s \rightarrow^* s'$; clearly, the set $ReachableFrom_{\mathcal{S}}(s)$ can be computed in polynomial time in the size of $\mathcal{S}$.

Input: Two propositions $\iota, \gamma \in Prop$ and a transition system $\mathcal{S} = (S, \rightarrow, \lambda)$ over $Prop$
Output: The set $\{(s, s') \in S \times S \mid s \in \lambda(\iota), s' \in \lambda(s') \text{ and } s \rightarrow^* s'\}$
$P := \emptyset$;
foreach $s \in \lambda(\iota)$ **do**
  | $P := P \cup \{s\} \times (ReachableFrom_{\mathcal{S}}(s) \cap \lambda(\gamma))$
end
return P

Algorithm 4: $\texttt{ends}(\iota, \gamma, \mathcal{S})$.

Since $\mathcal{S}$ is finite and since the size of the sets $\lambda(\iota)$ and $\lambda(\gamma)$ are less than or equal to the size of $\mathcal{S}$, and because $ReachableFrom_{\mathcal{S}}(s)$ is computable in polynomial time, we can claim the following.

Lemma 1. *Algorithm 4 terminates and its execution time is polynomial.*

It is also not hard to establish the correctness of Algorithm 4.

Input: An AND-free attack tree τ and a transition system $\mathcal{S} = (S, \rightarrow, \lambda)$
Output: The set $\{(\pi.first, \pi.last) \mid \pi \in [\![\tau]\!]^{\mathcal{S}}\}$
switch τ **do**
  case $\langle \iota, \gamma \rangle$ **do**
    return $\mathtt{ends}(\iota, \gamma, \mathcal{S})$;
  end
  case $\mathit{OR}(\tau_1, \tau_2)$ **do**
    return $\mathtt{pairs}(\tau_1, \mathcal{S}) \cup \mathtt{pairs}(\tau_2, \mathcal{S})$
  end
  case $\mathit{SAND}(\tau_1, \tau_2)$ **do**
    return $\{ (s_1, s_2) \mid$ there exists s_3 such that $(s_1, s_3) \in \mathtt{pairs}(\tau_1, \mathcal{S})$ and $(s_3, s_2) \in \mathtt{pairs}(\tau_2, \mathcal{S})) \}$
  end
end

Algorithm 5: $\mathtt{pairs}(\tau, \mathcal{S})$.

Lemma 2. *Algorithm 4 returns* $\{(s, s') \in S \times S \mid s \in \lambda(\iota), s' \in \lambda(s')$ *and* $s \rightarrow^* s'\}$.

We now describe the central Algorithm 5, which is defined by induction on τ.

Lemma 3. *Algorithm 5 terminates and computes in polynomial time the set*

$$\{(s, s') \in S \times S \mid \textit{there exists } \pi \in [\![\tau]\!]^{\mathcal{S}} \textit{ s.t. } s = \pi.first \textit{ and } s' = \pi.last\}$$

Proof. The algorithm terminates since recursive calls are executed on smaller trees and the base case is a call to Algorithm 4 which terminates by Lemma 1. The correctness of Algorithm 5 can be established by conducting an inductive reasoning on τ while taking into account the semantics of the OR and SAND operators according to Definition 6. It is left to the reader. Regarding the time complexity of Algorithm 5, one can easily see that each node of the tree is visited once and that for each node the computation is in polynomial time, so that the overall time complexity remains polynomial.

We now can conclude the proof of Theorem 4 by observing that deciding the non-emptiness of an AND-free attack tree is equivalent to deciding $\mathtt{pairs}_{\mathcal{S}}(\tau) \neq \emptyset$?, which can be achieved in polynomial time by Lemma 3 and the fact that verifying the non-emptiness of some set can be done in $O(1)$.

Actually, the optimal complexity of Non-emptiness$_{Af}$ is the following.

Theorem 5. Non-emptiness$_{Af}$ *is* NLOGSPACE*-complete.*

Proof. The NLOGSPACE-hardness of Non-emptiness$_{Af}$ follows from a trivial logspace reduction from the s–t-connectivity in an explicit graph – which is NLOGSPACE-complete according to [21] – to the non-emptiness of the path semantics of a leaf attack tree (of the form $\langle \iota, \gamma \rangle$).

For the NLOGSPACE-easiness, we describe Algorithm 6 which is a non-deterministic logspace algorithm that decides Non-emptiness$_{Af}$. Algorithm 6 may look technical but its idea is simple: non-deterministically guess a path in $\mathcal{S}$ and simultaneously perform an exploration of the tree akin to a depth-first traversal. For SAND-nodes, perform the depth-first traversal as usual. For OR-nodes, guess one of the two children

Input: An AND-free attack tree τ and a transition system $\mathcal{S}$
Output: Accept whenever $[\![\tau]\!]^S \neq \emptyset$.
guess $s \in S$;
node := root of τ;
lastOp := *down*;
repeat
  if *node* = $\langle \iota, \gamma \rangle$ **then**
    check $s \models \iota$;
    loop
      guess whether we break the loop or not; if yes, **break** the loop;
      guess $s' \in S$ with $s \to s'$;
      $s := s'$
    endLoop
    check $s \models \gamma$;
  end
  if *(lastOp* = down*) or (lastOp* = over*)* **then**
    Try to perform and update *node* with operation *down*, *over*, *up* in priority order;
    Store in *lastOp* the last performed operation
  else
    Try to perform and update *node* with operation *over*, *up* in priority order;
    Store in *lastOp* the last performed operation
  end
until *(node = root of* τ*) and (lastOp* = up*)*;
accept

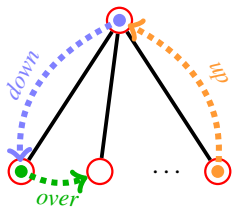
Algorithm 6: $\texttt{nonemptinessNL}_{ANDfree}(\tau,\mathcal{S})$.

to explore while the other child is dismissed. When a leaf node $\langle \iota, \gamma \rangle$ is visited, nondeterministically extend the path with a suffix and check that the first state of this suffix is labeled by ι and that its last state is labeled by γ (see the first **if**-block in the **repeat**-loop).

The constructed path is not entirely stored: only its current last state s is memorized which requires a logarithmic number of bits in the size of $\mathcal{S}$. This exploration is implemented in logarithmic space via a technical trick similar to the one proposed in [13] for tree canonization.

Before explaining the variant of the depth-first traversal we use, we describe the technical trick for a standard depth-first traversal [13].

The traversal relies on three operations: *down*, *over*, *up*. The standard operations work as follows: operation *down* moves to the first child of the current node and fails if the current node has no children; operation *over* moves to the next sibling (left to right) of the current node and fails if the current node has no next sibling; operation *up* moves to the parent of the current node and fails if the current node is the root.



In order to visit only one child of an OR node, we modify the behavior of operations *down* and *over*; the behavior of operation *up* remains unchanged: if the current node is an OR-node, operation *down* guesses a child and moves to it; if the parent of the current

node is an OR node, operation *over* always fails (instead of moving to the next sibling). The obtained modification of the depth-first traversal is such that exactly one child of an OR node is non-deterministically chosen and visited.

Algorithm 6 starts its exploration at the root of the attack tree and guesses a starting state s in $\mathcal{S}$. During the execution of the algorithm, variable s stores the last state in the current guessed path, variable *node* stores the current visited node in the tree and $lastOp$ stores the last operation that was performed. At the beginning, we consistently suppose (by convention) that operation *down* has been performed. The **repeat**-loop performs the modified traversal of the attack tree. As already mentioned above, the first **if**-block treats a leaf $\langle \iota, \gamma \rangle$: it non-deterministically moves forward in the path and checks that the built path complies with the pre/post-conditions ι and γ. The second **if**-block controls the depth-first traversal. The **repeat**-loop ends when the traversal is finished, namely when the current node is the root of τ and the last operation is *up*.

7 The Case of Symbolic Transition Systems

So far in this paper, we have assumed that the system $\mathcal{S}$ is described in extension. However, in realistic applications, this explicit description may be huge owing to the classic state explosion problem. A way to circumvent this explosion is to represent systems in an implicit manner, known as *symbolic transition systems*. Typical symbolic representations are data structures such as BDDs [5] or languages such as STRIPS [6].

We introduce the decision problem Non-emptiness$_{symb}$ akin to Non-emptiness but where the input system is given symbolically. The price to pay for dealing with a succinct presentation of the system $\mathcal{S}$ yields the following increase of complexity.

Theorem 6. Non-emptiness$_{symb}$ *is* PSPACE*-complete.*

Regarding the PSPACE-hardness of Non-emptiness$_{symb}$, it is known that the *symbolic reachability problem*, i.e. knowing if in a symbolic transition system there exists some path from a given set of source states to a set of target states, is PSPACE-complete [6]. As an immediate consequence, deciding the non-emptiness of attack trees is already PSPACE-hard for leaf trees, i.e. whether $[\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}} \neq \emptyset$.

Concerning the PSPACE-easiness of Non-emptiness$_{symb}$, we can adapt Algorithms 1 and Algorithm 2 for Non-emptiness as follows. First, guessing a state s of $\mathcal{S}$ is performed by guessing the polynomial number of bits that encode s in the symbolic representation of $\mathcal{S}$; this information is logarithmic in the exponential number of states denoted by the symbolic transition system, hence this information has a size that is polynomial in the size of the symbolic system. Second, checking $\texttt{reach}_{\mathcal{S}}(s_1, s_2)$ is an instance of the symbolic reachability problem, known to be computable by an algorithm running in polynomial space [6].

All in all, those adaptations of Algorithms 1 and 2 yield an algorithm that is non-deterministic with a logspace complexity. This shows that the problem Non-emptiness$_{symb}$ is in NPSPACE. Invoking Savitch's Theorem [19] that states the equality of the two complexity classes NPSPACE and PSPACE is enough to conclude.

8 Conclusion and Future Work

We have addressed the very natural decision problem of the non-emptiness of an attack tree, which involves an input tree and an input transition system, and we have studied its computational complexity. Mainly, the problem is (1) NP-complete for arbitrary trees, (2) NLOGSPACE-complete if we restrict to `AND`-free trees, and (3) PSPACE-complete for arbitrary trees and symbolic transition systems.

Regarding the most general problem Non-emptiness with no restriction on attack trees, the established NP upper bound (Theorem 4) means that when the system is represented explicitly and is of "reasonable" size, it is relevant to consider implementations based on one (or a combination) of the following intelligent search algorithmic techniques: backtracking, backjumping, integer linear programming, reduction to `SAT`, use of SMT solvers. The use of a SAT solver could be used to encode the `AND`-constraints (parallel decomposition). Actually, it has already been successfully applied for a related problem in [3]: deciding the membership of a path in the semantics of an attack tree τ with respect to a system $\mathcal{S}$, formally "$\pi \in [\![\tau]\!]^{\mathcal{S}}$?".

Regarding our complexity results for the problem Non-emptiness$_{Af}$, for the case of `AND`-free attack trees, we first showed that it is in P (Theorem 4), which means that we have an efficient algorithm. Even better, we showed that it is in NLOGSPACE (Theorem 5). Because the class NLOGSPACE falls within NC (Nick's class) [16, Theorem 16.1], the problem Non-emptiness$_{Af}$ can be efficiently solved on parallel architectures (see [16, p. 376]).

In the future, we plan to resort to solvers to design and implement a reasoning tool on the non-emptiness of attack trees. Actually, such a reasoning tool also requires to solve the reachability problem (see the procedure $\texttt{reach}_{\mathcal{S}}$ used in Algorithm 2). For these reasons, we will not only use a mere SAT solver but intend to draw on the $DPLL(T)$[6] architecture [4] of Satisfiability Modulo Theory (SMT) solvers. In our case, the theory T would be the system $\mathcal{S}$ itself, over which we solve the reachability problem. While an SMT solver architecture decomposes into a SAT solver and a decision procedure for T, our case would rather require an architecture decomposed into a SAT solver and a model checker. On the one hand, the constraints reflected by the abstract semantics $[\![\tau]\!]^{\mathcal{S}}_{\text{abs}}$ may be solved by the SAT solver that returns a possible valuation reflecting a word $w \in [\![\tau]\!]^{\mathcal{S}}_{\text{abs}}$. On the other hand, the model checker would verify that word w can be realized by a path in the system $\mathcal{S}$. Similarly to what is done in SMT solvers, the SAT solver and the model checker will exchange information: the SAT solver provides elements $w \in [\![\tau]\!]^{\mathcal{S}}_{\text{abs}}$ to the model checker and the model checker informs the SAT solver when a w is inconsistent within $\mathcal{S}$. Interestingly, such an approach would synthesize a "witness" path of any non-empty attack tree.

References

1. Zaruhi Aslanyan and Flemming Nielson. Model checking exact cost for attack scenarios. In *International Conference on Principles of Security and Trust*, pages 210–231, Berlin, Heidelberg, 2017. Springer.

[6] where DPLL stands for Davis-Putnam-Logemann-Loveland and T is a first-order theory

2. Maxime Audinot, Sophie Pinchinat, and Barbara Kordy. Is my attack tree correct? In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security – ESORICS 2017*, pages 83–102, Cham, 2017. Springer International Publishing.
3. Maxime Audinot, Sophie Pinchinat, and Barbara Kordy. Guided design of attack trees: a system-based approach – to be published. In *Computer Security Foundations Symposium, 2018. CSF'18. 31th IEEE*. IEEE, 2018.
4. Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, pages 825–885. IOS press, 2009.
5. Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.
6. Tom Bylander. The computational complexity of propositional STRIPS planning. *Artif. Intell.*, 69(1-2):165–204, 1994.
7. Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.
8. Ross Horne, Sjouke Mauw, and Alwen Tiu. Semantics for Specialising Attack Trees based on Linear Logic. *Fundam. Inform.*, 153(1-2):57–86, 2017.
9. Marieta Georgieva Ivanova, Christian W. Probst, René Rydhof Hansen, and Florian Kammüller. Attack tree generation by policy invalidation. In *WISTP*, volume 9311 of *LNCS*, pages 249–259. Springer, 2015.
10. Marieta Georgieva Ivanova, Christian W Probst, René Rydhof Hansen, and Florian Kammüller. Transforming graphical system models to graphical attack models. In *International Workshop on Graphical Models for Security*, pages 82–96. Springer, 2015.
11. Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Rolando Trujillo-Rasua. Attack Trees with Sequential Conjunction. In *SEC*, volume 455 of *IFIP Advances in Information and Communication Technology*, pages 339–353. Springer, 2015.
12. Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. Quantitative attack tree analysis via priced timed automata. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 156–171. Springer, 2015.
13. Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 400–404, 1992.
14. Sjouke Mauw and Martijn Oostdijk. Foundations of Attack Trees. In *ICISC*, volume 3935 of *Lecture Notes in Computer Science*, pages 186–198. Springer, 2005.
15. Hanne Riis Nielson, Flemming Nielson, and Roberto Vigo. Discovering, quantifying, and displaying attacks. *Logical Methods in Computer Science*, 12, 2016.
16. Christos H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007.
17. Sophie Pinchinat, Mathieu Acher, and Didier Vojtisek. Towards synthesis of attack trees for supporting computer-aided risk analysis. In *SEFM Workshops*, volume 8938 of *LNCS*, pages 363–375. Springer, 2014.
18. Sophie Pinchinat, Mathieu Acher, and Didier Vojtisek. ATSyRa: An Integrated Environment for Synthesizing Attack Trees – (Tool Paper). In *GraMSec@CSF*, volume 9390 of *LNCS*, pages 97–101. Springer, 2015.
19. Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.
20. Bruce Schneier. Attack Trees: Modeling Security Threats. *Dr. Dobb's Journal of Software Tools*, 24(12):21–29, 1999.
21. Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.