Adversarial Formal Semantics of Attack Trees and Related Problems

Thomas Brihaye
University of Mons
Mons, Belgium
thomas.brihaye@umons.ac.be

Sophie Pinchinat
Université de Rennes, IRISA
Rennes, France
sophie.pinchinat@irisa.fr

Alexandre Terefenko
Université de Rennes, IRISA
Rennes, France
University of Mons
Mons, Belgium
alexandre.terefenko@irisa.fr

Security is a subject of increasing attention in our actual society in order to protect critical resources from information disclosure, theft or damage. The informal model of attack trees introduced by Schneier, and widespread in the industry, is advocated in the 2008 NATO report to govern the evaluation of the threat in risk analysis. Attack-defense trees have since been the subject of many theoretical works addressing different formal approaches.

In 2017, M. Audinot et al. introduced a path semantics over a transition system for attack trees. Inspired by the latter, we propose a two-player interpretation of the attack-tree formalism. To do so, we replace transition systems by concurrent game arenas and our associated semantics consist of strategies. We then show that the emptiness problem, known to be NP-complete for the path semantics, is now PSPACE-complete. Additionally, we show that the membership problem is coNP-complete for our two-player interpretation while it collapses to P in the path semantics.

1 Introduction

Security is a subject of increasing attention in our actual society in order to protect critical resources from information disclosure, theft or damage. The informal model of attack trees was first introduced by Schneier [15] to schematically model possible threats one could execute against an information system. Attack trees have then been widespread in the industry and are advocated in the 2008 NATO report to govern the evaluation of the threat in risk analysis. The attack tree model is also a subject of increasing attention in the community of formal methods with a lot of different formal approaches [10, 8, 7, 6, 5, 12, 1] (see the survey [16]).

The first formal model of attack trees introduced in [15] aimed at describing a possible attack over a system by refining the main attack goal into sub-goals using either an operator *OR* or an operator *AND* to coordinate those refinements. The analysis conducted over those trees is "static" in the sense that the attacked system does not evolve during the attack. As such, there is no concept of a goal happening before or after another. In [6], the authors introduce a first formal semantics that can be qualified as "dynamic" by allowing a new operator, operator *SAND* (for sequential *AND*), to specify that sub-goals must be attained in a given order. Considering that the *SAND* operator is now commonly accepted, the authors of [1] propose a path semantics for attack trees over a transition system.

In this paper, our goal is to present a new semantics for attack trees in order to be able to model more realistic scenarios: we want our attacker to be able to adapt her actions according to the behavior of the environment – typically, a defender who tries to protect the system. This setting naturally yields a two-player semantics. Our approach is inspired by [1] where we generalize the path semantics to a game-theoretic framework, yielding a strategy semantics for attack trees, without changing their syntax.

D. Della Monica and P. Ganty (Eds.): 13th International Symposium on Games, Automata, Logics and Formal Verification (GandALF 22)
EPTCS 370, 2022, pp. 162–177, doi:10.4204/EPTCS.370.11

While the path semantics from [1] is compositional in a natural way, it turns out that the strategy semantics does not have such a nice property. Indeed, although composition of strategies is possible (see for example [11]), there is no immediate solution to compose two strategies in order to get a strategy that achieves the disjunction of what the formers achieve. This situation makes it difficult to design a compositional strategy semantics for attack trees. We therefore develop a non-trivial strategy semantics that is not compositionally obtained *per se*, but that makes use of the former compositional path semantics.

To our knowledge, our proposal is the first game-theoretic semantics for attack trees, and should not be mixed-up with the multi-player setting induced by the so-called classic model of *attack-defense trees* in the literature (see [9]): attack-defense trees are attack trees equipped with a new operator to express that some defender could use a countermeasure to prevent attacker from achieving her goal. Although well-understood in a "static" framework, we are not aware of any formal semantics of attack-defense trees for the “dynamic” one, i.e., over transition systems. This missing piece of work makes it difficult to conduct a comparison with our contribution, but, from the fact that the defender in our setting is fully formalized, as an opponent in the game arena, while the defender in attack-defense trees takes the form of an abstract entity, it seems that those two formalisms are not expressing the same kind of problems.

Our contribution in this paper is twofold.

We first develop a clean mathematical setting to obtain a formal strategy semantics for attack trees. For pedagogical reasons, we choose to consider a simplified version of the attack trees of [1] where atomic goals (at the leaves of the trees) are reachability goals with no preconditions. However, at the price of tedious definitions, the strategy semantics we propose can be adapted to atomic goals with preconditions. Regarding the design of this semantics, we heavily rely on the older one of [1] based on paths, and we justify our approach by providing evidence that a compositional strategy semantics is hopeless.

Second, we exploit the attack tree semantics to address and study the complexity of two decision problems: the *non-emptiness problem* and the *membership problem*. The former consists in determining if the semantics of an input tree is non-empty, while the latter consists in determining if an input element belongs to the semantics of an input tree. Our results are summarized in Table 1, where we distinguish between the path semantics and the strategy semantics.

Table 1: Complexity results

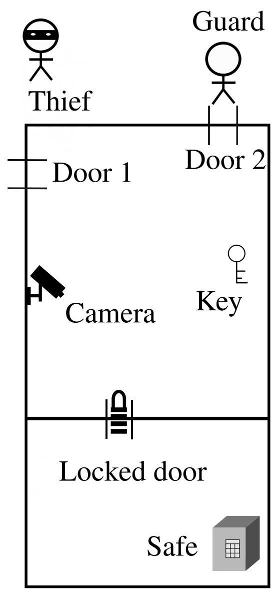
	Paths semantics	Strategy semantics
Non-Emptiness Problem	NP-complete	PSPACE-complete
Membership Problem	P	coNP-complete

Importantly, both decision problems have a practical counterpart. The non-emptiness of the path semantics of a tree reflects a situation where there exists a favorable scenario for attacker to perform her attack, while the non-emptiness of the strategy semantics reflects an intrinsic vulnerability of an information system. Regarding the membership problem for the path semantics, we are interested in knowing whether a log file of some information system execution makes evidence of an attack, while for the strategy semantics, we wonder if an attack policy is successful.

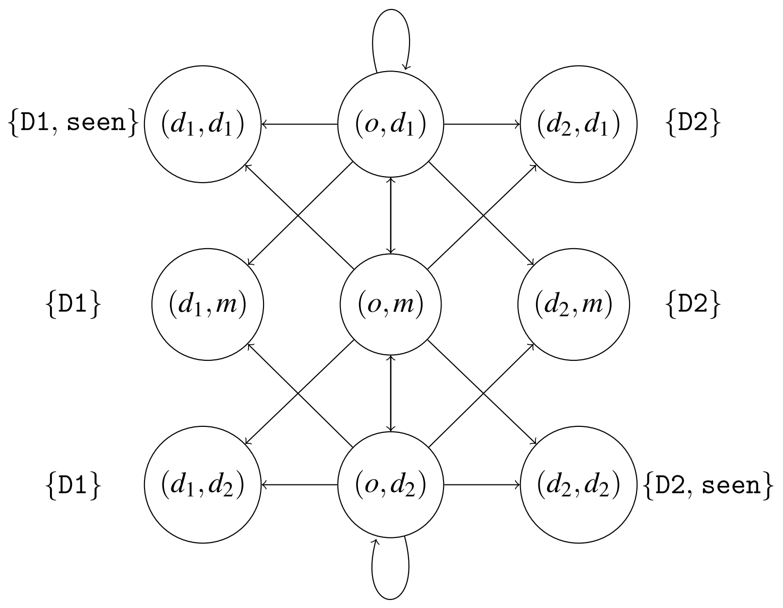
The paper is organized as follows. We start in Section 2 with an introductory example explaining informally the difference between path semantics and strategy semantics. After some background work in Section 3, we introduce in Section 4 the formal model of attack trees and define the path semantics inspired by [1] as well as our new strategy semantics. We then study the complexity of the Non-Emptiness and the Membership problems in Section 5.

2 Introductory example

Consider a thief (the attacker) who wants to steal some document inside a safe of a building without being seen. The building is composed of two rooms. The first room has two entrance doors (called door 1 and door 2) from the street. There is a guard keeping the entrance doors, but he can only control the bypassing of one of the two entrance doors at a time. The first room has also another door that leads to the second room. This door is locked but the key to unlock it is in the first room. There is also a camera in the first room monitoring the door to the second room. The second room contains a safe with the document that the thief wants to steal. Therefore, in order for the thief to attain his goal, he needs to enter the first room (by either door that is not currently controlled by the guard), then deactivate the camera and collect the key (in whichever order he wants) and finally unlock and go through the door leading to the second room. The thief can be seen by the guard if they appen to be in front of the same door or by the camera if activated when he is in front of the door to room 2. Figure 1a gives a picture of the situation where the thief is still outside of the building and the guard controls the second door.



(a) Building plan



(b) Graph representation of the possibles positions of the thief and the guard at the entrance of the building.

Figure 1: Introductory example building

In security, it is common to use an attack tree to model the goal of the attacker. An attack tree is a tree where each node describes a goal and the children of a node describe a refinement into sub-goals of the parent goal. To model those refinements, it is common to use three operators:

- *OR* operator means that at least one sub-goal needs be achieved to have the goal accomplished,
- *SAND* operator, read "sequential and", means that the sub-goals need be achieved in the left-to-right order to have the goal accomplished,
- *AND* operator means that the sub-goals need be achieved (in whatever order) to have the goal accomplished.

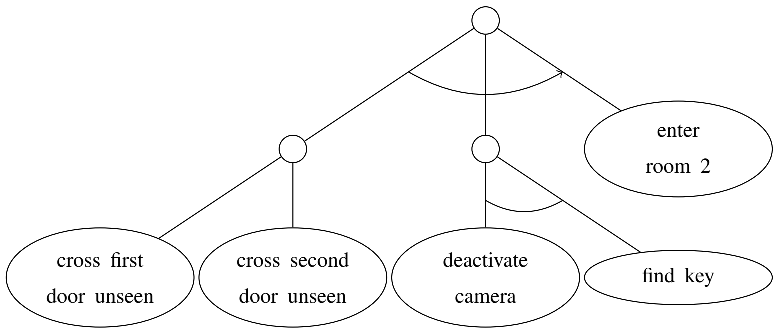


Figure 2: An attack tree to model the goal of our thief.

In Figure 2, we describe the goal of the thief by means of an attack tree. To distinguish the different types of nodes, we draw a curved line below an *AND* operator, a curved arrow below a *SAND* operator and nothing below an *OR* operator.

We focus our attention on the very first part of our problem, that is, the sub-goal of the thief to enter the building. We start by fixing a set of proposition $Prop = \{\texttt{D1},\texttt{D2},\texttt{seen}\}$ where `D1` holds when the thief is in front of the first door, `D2` holds when the thief is in front of the second door and `seen` holds when the guard sees the thief because they meet in front of the same door. The goal "crossing the first door unseen" of the thief can be modelled by the formula $\texttt{D1} \wedge \neg\texttt{seen}$ and the goal "crossing the second door unseen" is modelled by the formula $\texttt{D2} \wedge \neg\texttt{seen}$. If we assume that the guard can switch door whenever he wants but leaves the two doors unguarded for a brief amount of time during his motion, we can model the situation using the graph of Figure 1b where each state consists of a pair in the set $\{o,d_1,d_2\} \times \{m,d_1,d_2\}$. For a pair (a,b), element a determines the position of the thief (o means that he is still outside the building, while d_i indicates he is at door i) and element b determines the position of the guard (m means that he is currently in motion between the two doors, leaving them both unguarded). We write next to each state which propositions hold in it.

In a path semantics, a successful attack for goal $\texttt{D1} \wedge \neg\texttt{seen}$ consists of a sequence of states (i.e., a word) such that the valuation of the last state of this sequence satisfies formula $\texttt{D1} \wedge \neg\texttt{seen}$. In particular, the sequence $(o,m),(d_1,d_2)$ is a successful attack. Now, if we want to consider a successful attack for the attack tree consisting of the "OR" operator of objective $\texttt{D1} \wedge \neg\texttt{seen}$ and $\texttt{D2} \wedge \neg\texttt{seen}$, it is enough to consider the union of the set of all attacks for objective $\texttt{D1} \wedge \neg\texttt{seen}$ with the set of all attacks for objective $\texttt{D2} \wedge \neg\texttt{seen}$.

However, in the strategy semantics we introduce in this paper, we consider that an attack is successful if the attacker has a strategy that grants him to reach the states he needs to attain, independently of the environment. In our example, we can see that the thief has no strategy starting at position (o,m) to achieve goal $\texttt{D1} \wedge \neg\texttt{seen}$. Indeed, if the first move of the guard consists on going to door 1 and he then does not move any more, there is no way for the thief to cross the first door while unseen. Similarly, there is no strategy starting at position (o,m) to achieve goal $\texttt{D2} \wedge \neg\texttt{seen}$. However, if we consider the "OR" operator of the two goals $\texttt{D1} \wedge \neg\texttt{seen}$ and $\texttt{D2} \wedge \neg\texttt{seen}$, the strategy of the thief consisting in waiting for the guard to go to one of the two doors and then going to door 1 if the guard is at door 2 and vice versa is a successful strategy. Later, we will use similar a similar example to show that a compositional definition for a strategy semantics cannot be achieved.

3 Preliminary notions

Formal languages Given an alphabet (i.e., a finite set of symbols) Σ, notation Σ^* represents the set of finite words (i.e., sequences of symbols) over the alphabet Σ, with typical element $w = \ell_1 \dots \ell_n \in \Sigma^*$; the empty word is written ε. On the other hand, the set of infinite words is denoted by Σ^ω. A subset $L \subseteq \Sigma^*$ is a language. Given a word $w = \ell_1 \dots \ell_n \in \Sigma^*$, its set of prefixes is the language $Prefixes(w) = \{\ell_1 ... \ell_i | i \leq n\} \cup \{\varepsilon\}$, and we write $w' \lhd w$ whenever $w' \in Prefixes(w)$. A language L is prefix-closed if $w \in L$ implies $w' \in L$ for every $w' \lhd w$. The concatenation of a word $w = \ell_1 \dots \ell_n$ with another word $w' = \ell'_1 \dots \ell'_m$ is the word $ww' = \ell_1 \dots \ell_n \ell'_1 \dots \ell'_m$. The concatenation of a word with a language is defined in the usual way: for $w \in \Sigma^*$ and $L \subseteq \Sigma^*$, we let $wL := \{ww' | w' \in L\}$.

Game theory and game arena To formalize a strategy semantics for attack trees, we need standard two-player, zero-sum, perfect information games that we recall here.

A *game arena* is a finite graph on which two players play a game of unbounded duration. We choose to consider concurrent games, meaning that, at each round, each player makes an action. To define it properly, we consider two players called Player 1 and Player 2 and a finite set of propositions *Prop*.

Definition 3.1. A *two-player game arena* is a tuple $\mathscr{G} = (Pos, Act, \delta, val)$ where:

- $Pos = \{v, ...\}$ is a finite set of positions.
- $Act = Act_1 \times Act_2$ is a finite set of actions, as a product of the sets of actions of each player.
- $\delta : Pos \times Act \to Pos$ is a transition function,
- $val : Pos \to \mathscr{P}(Prop)$ is a valuation function.

In our introductory example of Section 2, if we consider the set of actions for the thief: {wait, go-to-Door-1, go-to-Door-2} and the following set of actions for the defender: {stay-at-current-door, leave-current-door, go-to-Door-1, go-to-Door-2}, then it is easy to see that the graph drawn in Figure 1b forms a game arena.

For the rest of this paper, we fix a game arena $\mathscr{G} = (Pos, Act, \delta, val)$.

For a game position $v \in Pos$, we define $Post(v) = \{v' \in Pos |\ \delta(v, a) = v'$ for some $a \in Act\}$ the set of all positions reachable from v in one step. For convenience, we assume that each player j can play every action a in Act_j at each position of the game arena, so that for each position $v \in Pos$, we have $Post(v) \neq \emptyset$. A play ρ is an infinite sequence of positions of the form $v_0 v_1 v_2 .... \in Pos^\omega$ such that for each $i \in \mathbb{N}$, there is $a \in Act$ such that $\delta(v_i, a) = v_{i+1}$. For $i \in \mathbb{N}$, we let $\rho_i = v_i$ be the i^{th} position of play ρ. The set of all plays is denoted by $Plays(\mathscr{G})$. Each non-empty prefix h of a play is called a *history* and the set of all histories is denoted by $Hist(\mathscr{G})$. For a history $h \in Hist(\mathscr{G})$, we define $last(h)$ as the last position of h. For $v \in Pos$, we also use the notations $Plays(\mathscr{G}, v)$ and $Hist(\mathscr{G}, v)$ to denote the set of all plays starting from v (i.e., $\rho_0 = v$) and the set of all histories starting from v, respectively.

Winning plays for Player 1 are obtained from a distinguished subset $\Gamma_1 \subseteq Plays(\mathscr{G})$. As we consider zero-sum games, all plays in $\Gamma_1 \backslash Plays(\mathscr{G})$ are winning for Player 2.

Classically, we introduce the notion of *strategy*, as a map prescribing how a player plays depending on the current history: a *strategy* μ^j for the Player j is a map $\mu^j : Hist(\mathscr{G}) \to Act_i$. The set of all strategies for the Player j is denoted as $Strat^j$.

A history $h = v_0 v_1 ... v_m$ is *consistent* with a strategy μ^j if for each $1 \leq i \leq m$, there exists $a = (a_1, a_2) \in Act$ such that we have $\delta(v_i, a) = v_{i+1}$ and $\mu^j(v_0 v_1 ... v_i) = a_j$. We say that a play ρ is consistent with a strategy μ^j if all prefixes of ρ are histories consistent with μ^j. The set of all plays consistent with μ^j is denoted by $Outcomes(\mu^j)$. From a game position v we say that a strategy is *winning* if all

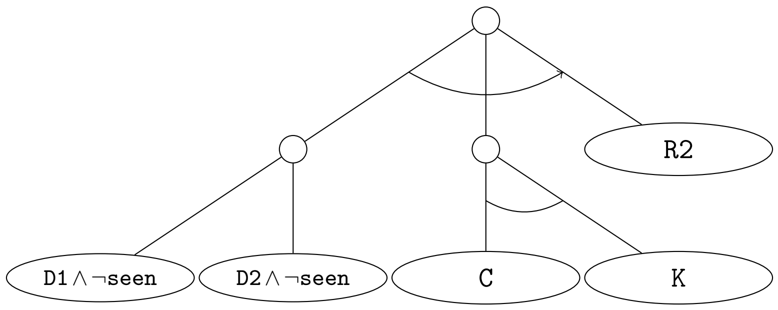


Figure 3: Formal version of the attack tree in Figure 2.

outcomes starting at v are winning. In other words, for a strategy of say Player 1, a strategy μ is winning if $Outcomes(\mu) \cap Plays(\mathscr{G}, v) \subseteq \Gamma_1$.

A classic kind of concurrent games are the *reachability games*. In such games, a player wants to reach some positions while the other player tries to prevent it from happening. These games are clearly zero-sum. More formally, we say that a game is a reachability game for Player 1 if there exists $W_1 \subseteq Pos$ such that $\Gamma_1 = \{\rho \in Plays(\mathscr{G}) | \rho_i \in W_1 \text{ for some } i \in \mathbb{N}\}$.

In a game arena, we says that a position v satisfies the formula ϕ if its valuation $val(\ ())$ satisfies the formula in classic propositional logic, denoted by $v \models \phi$. Note that a Boolean formula ϕ over *Prop* describes the reachability game $(\mathscr{G}, W_1)$ where $W_1 = \{v \in Pos | v \models \phi\}$.

4 Attack trees and their semantics

In this section, we start with the formal definition of attack tree used in this paper. Then we develop two semantics for attack trees: the path semantics and the strategy semantics.

4.1 Syntax of attack trees

To formalize attack trees, we start by fixing a set of propositions *Prop*.

Definition 4.1. An *attack tree* τ over *Prop* is:

- either a leaf composed of a unique Boolean formula ϕ over *Prop*,
- or an expression $OP(\tau_1, ..., \tau_n)$ where *OP* ranges over *OR*, *AND* and *SAND* and $\tau_1, ..., \tau_n$ are attack trees.

We define the size of an attack tree τ, noted $|\tau|$, by the number of its nodes.

Example 4.2. We will formalise the example introduced in Section 2. To represent the situation, we use the following set of propositions: $Prop = \{\texttt{D1}, \texttt{D2}, \texttt{seen}, \texttt{C}, \texttt{K}, \texttt{R2}\}$. We have that `D1` holds when the thief has crossed the first door, `D2` holds when the thief has crossed the second door, `seen` holds when the guard sees the thief, `C` holds when the camera is on, `K` holds when the thief has the key and finally `R2` holds when the thief is in the second room.

We can now propose a formal definition for the attack tree in Figure 2: $\tau = SAND(OR(\texttt{D1} \wedge \neg\texttt{seen}, \texttt{D2} \wedge \neg\texttt{seen}), AND(\texttt{C}, \texttt{K}), \texttt{R2})$ to model the objective of the attacker. The graph representation of τ is given by Figure 3.

Let us notice that the attack trees used in [1] are in fact slightly different: the leaves of the attack trees of that paper are of the form $< \phi_1, \phi_2 >$ with ϕ_1 and ϕ_2 two Boolean formulas. Formula ϕ_1 describes a precondition for the objective to begin with and formula ϕ_2 describes the postcondition for the objective to be granted. However, in this paper, we never consider preconditions. So a leaf ϕ of our attack trees can be seen as a leaf $< true, \phi >$ of attack trees introduced in [1]. Although the semantics defined in this paper could also be defined for attack trees with preconditions, not considering them makes the setting more pedagogical.

The first semantics for attack trees we introduce is a path semantics inspired by [1]. Informally, in the path semantics, we consider all sequences of events that lead to a successful attack. The idea is to determine which scenarios are favourable for the attacker. One could also say that an attack can occur if the attacker is lucky.

The second semantics is our main contribution, and is named the *strategy semantics* for attack trees. In this approach, the attacker should not rely on an opportunity offered by the environment but should be able to find the right sequence of actions whatever the environment does. Otherwise said, an attack is not a favourable scenario anymore but a winning strategy for attacker in some two-player game arena.

4.2 Path semantics for attack trees

To give a path semantics over our trees, we first need to fix a transition system to model which actions/sequences of actions can be executed by the attacker in the system. A transition system is composed of a finite set of states together with a transition relation between pairs of states. We decided not to label every transition with an action as we only consider perfect information here. We also provide a valuation function to our transition system, informing which propositions of *Prop* holds in a state of the transition system.

Definition 4.3. A *transition system* over *Prop* is a triplet $\mathscr{S} = (S, \delta, val)$ where:

- S is a finite set of states,
- $\delta \subseteq S \times S$ is a relation of transitions,
- $val : S \to \mathscr{P}(Prop)$ is a valuation function.

The size of $\mathscr{S}$, noted $|\mathscr{S}|$, is defined by its number of states.

We can see from Definition 4.3 that a transition system is a notion close to a game arena. Indeed, it is easy to associate a transition system with a game arena $\mathscr{G} = (Pos, Act, \delta, val)$, by merging the two players into a single one in the following way: $\mathscr{S}_{\mathscr{G}} = (Pos, \{(v, v') \in Pos \times Pos|$ there exists $a \in Act$ such that $\delta(v, a) = v'\}, val)$. Later, we denote $\mathscr{S}_{\mathscr{G}}$ by $\mathscr{G}$ when it is clear from the context.

For the rest of this section, we fix a transition system $\mathscr{S} = (S, \delta, val)$ over *Prop*. A *path* in a transition system is a finite non-empty sequence of states $\pi = s_0 s_1 ... s_n$ such that, for each $0 \leq i < n$, $(s_i, s_{i+1}) \in \delta$. The size of a path is its number of states. We denote the set of all paths in $\mathscr{S}$ by $\Pi_{\mathscr{S}}$.

In order to define the path semantics we need to introduce operators over paths.

Definition 4.4. Let $\pi = s_0 s_1 ... s_n$ and $\pi' = s'_0 s'_1 ... s'_m$ be two paths in $\mathscr{S}$ with $n, m \geq 0$. The *synchronised concatenation* of π_1 and π_2 is defined only if $s_n = s'_0$ and is given by t $\pi \cdot \pi' = s_0 s_1 ... s_n s'_1 ... s'_m$.

We lift this operations to sets of paths the following way: if Π_1 and Π_2 are two sets of paths, then $\Pi_1 \cdot \Pi_2 = \{\pi_1 \cdot \pi_2 | \pi_1 \in \Pi_1$ and $\pi_2 \in \Pi_2\}$.

The authors of [1] introduce the operator of *parallel composition of paths*. However, our definition of attack trees grants us the possibility to use a simpler operator.

Definition 4.5. Let Π_1, Π_2 be two sets of paths of $\mathcal{S}$. The *merge* of Π_1 and Π_2 is the set of paths $\Pi_1 \triangle \Pi_2 = \{\pi_1 \in \Pi_1 |$ there exists $\pi_2 \in \Pi_2$ such that $\pi_2 \lhd \pi_1\} \cup \{\pi_2 \in \Pi_2|$ there exists $\pi_1 \in \Pi_1$ such that $\pi_1 \lhd \pi_2\}$

Unlike the parallel composition of [1], thanks to the transitivity of the prefix relation, the merge operator is associative.

We can now define our path semantics.

Definition 4.6. Let τ be a attack tree over *Prop*. The path semantics of τ over $\mathcal{S}$ is the set of paths $Paths_{\mathcal{S}}(\tau)$ inductively defined as follow:

- $Paths_{\mathcal{S}}(\phi) = \{s_0 s_1 ... s_n \in \Pi_{\mathcal{S}} | s_n \models \phi\}$
- $Paths_{\mathcal{S}}(OR(\tau_1, ..., \tau_n)) = Paths_{\mathcal{S}}(\tau_1) \cup ... \cup Paths_{\mathcal{S}}(\tau_n)$
- $Paths_{\mathcal{S}}(SAND(\tau_1, ..., \tau_n)) = Paths_{\mathcal{S}}(\tau_1) \cdot ... \cdot Paths_{\mathcal{S}}(\tau_n)$
- $Paths_{\mathcal{S}}(AND(\tau_1, ..., \tau_n)) = Paths_{\mathcal{S}}(\tau_1) \triangle ... \triangle Paths_{\mathcal{S}}(\tau_n)$

It is easy to verify that the semantics of Definition 4.6 is equivalent to the one introduced in [1] if we restrict to the attack trees whose leaves are of the form $< true, \phi >$.

Remark that, in our framework, for ϕ_1 and ϕ_2 two formulas over *Prop*, the interpretation of $SAND(\phi_1, \phi_2)$ is that ϕ_1 must hold at some point and ϕ_2 must hold at some point afterwards. This requirement does not prevent ϕ_2 from holding before ϕ_1.

We also want to point out that our semantics consider that the simultaneity of objectives is always successful: for ϕ_1 and ϕ_2 two formulas over *Prop*, if ϕ_1, $\phi_2 \in val(s)$, then $s \in Paths_{\mathcal{S}}(SAND(\phi_1, \phi_2))$ and $s \in Paths_{\mathcal{S}}(AND(\phi_1, \phi_2))$.

Example 4.7. If we consider the game arena $\mathcal{G}$ given in Figure 1b, we have that $(d_1, d_2) \models \texttt{D1} \wedge \neg\texttt{seen}$, thus the path $(o,m)(d_1,d_2) \in Paths_{\mathcal{G}}(\texttt{D1} \wedge \neg\texttt{seen})$. This gives us also $(o,m)(d_1,d_2) \in Paths_{\mathcal{G}}(OR(\texttt{D1} \wedge \neg\texttt{seen}, \texttt{D2} \wedge \neg\texttt{seen}))$.

4.3 Strategy semantics for attack trees

We start this section by formally defining *strategic trees* as well as some handful operators over them. We use a definition of a tree really close to the one made from prefix-closed languages (for example in [3, p. 15]) except that we fix a letter to represent the root.

Definition 4.8. A *strategic tree* (written s-tree for short) over an alphabet Σ is a language T of the form ℓL with $\ell \in \Sigma$ and L is a prefix-closed language over Σ.

For an s-tree $T = \ell L$, ℓ is called the *root*. For a word $w \in T$, if there exists no $w' \in T$ such that $w \lhd w'$ then we call w a *leaf*. The set of all leaves of T is denoted by $Leaves(T)$. For two words $w, w' \in T$ such that $w \lhd w'$, if there exist no $w'' \in T$ such that $w \lhd w'' \lhd w'$, then we says that w is the *parent* of w' and w' is a *child* of w. The set of all children of a word w in a s-tree T is denoted by $Children_T(w)$. The depth of an s-tree is the size of the longest word in it.

Example 4.9. Figure 4 shows an s-tree over the alphabet *Pos*, the set of positions of the game arena of Figure 1b.

As in Example 4.9, for the particular case where alphabet Σ is the set of positions on some game arena, we develop several notions on s-trees and show that strategies can be presented as s-trees.

For the rest of this section we fix a game arena $\mathcal{G} = (Pos, Act, \delta, val)$.

The next lemma asserts that all histories consistent with a strategy and starting from a given position form an s-tree.

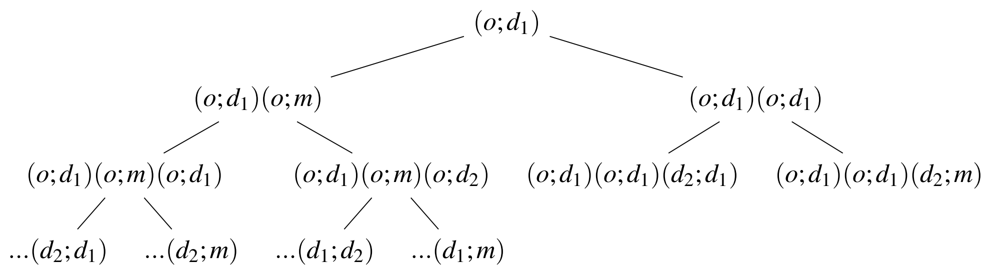


Figure 4: strategic tree $T^{\mu}_{(o,d_1)}$

Lemma 4.10. *Let μ be a strategy for some player and $v \in Pos$ be a game position. The language $T^{\mu}_{v} = Prefixes(Outcomes(\mu)) \cap Hist(\mathscr{G}, v)$ is an s-tree over alphabet Pos, and is called the* s-tree associated with μ from position v.

The proof is straightforward from the definition of T^{μ}_{v}.

By Lemma 4.10, each branch of T^{μ}_{v} is the succession of all prefixes (in terms of words) of a play consistent with μ. Reciprocally, each play consistent with μ and starting from position v is represented by a branch of T^{μ}_{v}. Therefore T^{μ}_{v} fully describes the strategy μ starting from position v.

Example 4.11. Consider the game arena $\mathscr{G}$ given in Figure 1b and the strategy μ for the thief consisting in waiting one unit of time, then, if the guard is at some door, going to the other door and if the guard is currently in motion, waiting another unit of time before going to the door where the guard will not be. If we call this strategy μ, the strategic tree $T^{\mu}_{(o,d_1)}$ is given in Figure 4.

As we put the focus on attack trees, we take the convention that, in the game arena, Player 1 is called *Attacker* and Player 2 is called *Defender*. In this setting, Attacker tries to achieve an attack that is described by some attack tree τ, while Defender tries to prevent it from happening. In other words, the winning plays for the Attacker are given as $\Gamma_A = Paths_{\mathscr{G}}(\tau)$. Our strategy semantics consists of the set of winning strategies for this game.

We start by motivating a construction only for a leaf attack tree. The strategy semantics for an attack tree ϕ is the set of all strategies that are winning for the reachability game defined by ϕ. Remark that for the case of reachability games, once a winning position is reached, the continuation of the play does not matter. Therefore, for reachability games, the s-tree corresponding to a winning strategy can be cut as a finite tree: this cut consists in removing all children of a node describing a history ending in a position where ϕ holds. This way of cutting motivates the definition of prefix of s-trees as follows:

Definition 4.12. Let T be an s-tree over Σ^*. An s-tree T' is a *prefix* of T if $root(T') = root(T)$, and $T' \subseteq T$, and for every $w \in T' \setminus Leaves(T')$, we have $Children_{T'}(w) = Children_{T}(w)$.

Example 4.13. For the s-tree $T^{\mu}_{(o,d_1)}$ of Figure 4 and the two trees given in Figure 5, we have T_a is a prefix of $T^{\mu}_{(o,d_1)}$, but T_b is not because $Children_{T^{\mu}_{(o,d_1)}}(o,d_1) = \{(o,d_1)(o,m),(o,d_1)(o,d_1)\} \neq Children_{T_b}(o,d_1) = \{(o,d_1)(o,d_1)\}$.

With this notion of prefix, it is immediate to characterise attack trees that witness a strategy.

Definition 4.14. Consider a leaf attack tree ϕ, and write $\phi \subseteq Pos$ for the set of positions where ϕ holds. Consider μ a strategy for Attacker in the reachability game $(\mathscr{G}, \phi)$ and T^{μ}_{v} the associated s-tree from position v. A finite s-tree T is a *witness* of μ from position v if T is a finite prefix of T^{μ}_{v}, and $Leaves(T) \subseteq Pos^{*}\phi$.

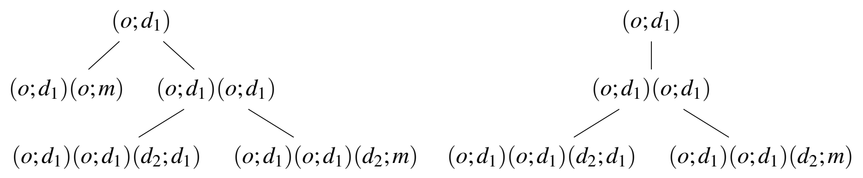


Figure 5: two s-trees: T_a (left) and T_b (right)

Definition 4.14 can be generalised to an arbitrary reachability condition $W_1 \subseteq Pos$ as follows: T is a witness of μ from position v if T is a finite prefix of T_v^μ and $h \in Leaves(T)$ implies $last(h) \in W_1$.

Example 4.15. In the game arena of Figure 1b, if we consider the reachability condition $W_1 = \{(o,m), (d_2,m), (d_2,d_1)\}$, then the attack tree T_a of Figure 5 is a witness for the s-tree $T^{\mu}_{(o,d_1)}$ drawn in Figure 4.

Definition 4.14 leads us to the following intuitive lemma.

Lemma 4.16. *Let μ be a strategy for Attacker and W_1 be a winning condition. Then μ is a winning strategy for $(\mathscr{G}, W_1)$ from position $v \in Pos$ if, and only if, there exists a witness T of μ from v.*

The proof relies on the König's Lemma.

Thus, for a leaf ϕ, the strategy semantics is all witnesses that can be constructed from a winning strategy over the reachability game defined by ϕ. Moreover, an s-tree is in the semantics of a leaf attack tree if it is a prefix of some strategy and if all its leaves are in the path semantics of the attack tree. The former condition guarantees that our s-tree has the shape of a strategy, while the latter guarantees that the strategy is winning. As we will see below, those are the two conditions we use to define the strategy semantics of arbitrary attack trees.

For the first condition, we say that an s-tree T is *well-formed* if there exists a strategy μ and a position v such that T is a prefix of T_v^μ. For the second condition, we use the following definition:

Definition 4.17. Let τ be an attack tree. A *τ-s-tree* is a finite s-tree T over Pos such that $Leaves(T) \subseteq Paths_{\mathscr{G}}(\tau)$.

Since for a leaf attack tree ϕ, we have $Paths_{\mathscr{G}}(\phi) = Pos^*\phi$, a witness T (Definition 4.14) is a ϕ-s-tree. We now have all the material to define the strategy semantics of an attack tree.

Definition 4.18. Let τ be an attack tree. The *strategy semantics* associated with τ, written $Strat_{\mathscr{G}}(\tau)$ is the set of all well-formed τ-s-trees.

In particular, $Strat_{\mathscr{G}}(\phi)$ is the set of all witnesses in the reachability game $(\mathscr{G}, \phi)$.

We can see that the idea is far from the one of attack-defence trees in [8]. In attack-defence trees, the countermeasure is a structure similar to an attack tree whose semantics describes paths that prevent an attack from succeeding, and by no means a strategy of the attacker's opponent in the arena.

Now that we defined our semantics, we might want to know if it can be obtained in a compositional manner ? Namely, if the semantics of a compound tree can be defined in terms of the semantics of its subtrees: More formally.. can we define $Strat_{\mathscr{G}}(OP(\tau_1, ..., \tau_n))$ on the basis of $Strat_{\mathscr{G}}(\tau_1), ..., Strat_{\mathscr{G}}(\tau_n)$? Sadly, the answer is no:

Example 4.19. Consider the game arena defined in Figure 1b. Obviously, our attacker here will be the thief while the guard will do the defender role. We also consider a new proposition: *Start* which only holds at position $\{o,m\}$. We have that the semantics of $SAND(start, \texttt{D1} \wedge \neg\texttt{seen})$ is empty. Indeed, the guard can choose to only keep door 1 and thus, the thief will not be able to attain D1 while remaining unseen. Similarly, $Strat_{\mathscr{G}}(SAND(start, \texttt{D2} \wedge \neg\texttt{seen}))$ is empty. However, the strategy consisting on waiting one unit of time then going through the door not controlled by the guard is a winning strategy, it is easy to construct a witness for that strategy that attains the objective of $OR(SAND(start, \texttt{D1} \wedge \neg\texttt{seen}), SAND(start, \texttt{D2} \wedge \neg\texttt{seen}))$ and thus is in its strategy semantics.

The previous example showcases an empty semantics for τ_1 and τ_2 but a non-empty one for $OR(\tau_1, \tau_2)$. This is because, for ϕ_1 and ϕ_2 two propositional formulas over *Prop*, there are more strategies to achieve $\phi_1 \vee \phi_2$ than strategies only achieving ϕ_1 or only achieving ϕ_2. We can for example consider a strategy that, depending on the move of the opponent, chooses whether it prefers to attain ϕ_1 or to attain ϕ_2.

Remark that, using the "merge" operator of [11] provides us a compositional semantics for attack trees with *SAND*-only operators. However, we have already argues that the *OR* operator have some problems just as the *AND* operator for more elaborate examples. Still, it is possible to tune the semantics so that it becomes compositional for the AND operator, at the price of loosing clarity, but more regrettably without solving the hopeless case of the OR operator.

5 Decision Problems over attack trees

In this section, we discuss two common decision problems over semantics of attack trees and determine their complexities with respect to the path semantics and the strategy semantics. The first problem we consider is the Non-Emptiness problem. This problem consists of, given an attack tree and a game arena, deciding whether its semantics is not empty:

Definition 5.1. The *Non-Emptiness* problem is the following decision problem for a fixed semantics $[\![\cdot]\!]_{\mathscr{G}}$ of attack trees:
Input: $\mathscr{G}$, a game arena, τ, an attack tree.
Output: *Yes* if $[\![\tau]\!]_{\mathscr{G}} \neq \emptyset$, *No* otherwise.

The Non-Emptiness problem for the path semantics is denoted by PNE while the Non-Emptiness problem for the strategy semantics is denoted SNE. A positive instance of PNE tells us that Attacker has a favourable scenario to attack. A positive instance of SNE tells us that Attacker has a strategy (it is possible for him to attack successfully the system independently of the defender/environment comportment).

We now turn to the Membership problem.

Definition 5.2. The *Membership problem* is the following decision problem for a fixed attack tree semantics $[\![\cdot]\!]_{\mathscr{G}}$ of of type X:
Input: $\mathscr{G}$, a game arena, τ, an attack tree and $x \in X$.
Output: Yes if $x \in [\![\tau]\!]_{\mathscr{G}}$, No otherwise.

The Membership problem for the path semantics is denoted by PM while the Membership problem for the strategy semantics is denoted SM. PM consists of determining whether a path is an attack or not. It can be really useful if we have an attack tree describing an attack goal over an information system and a log file of that system. Determining if the system has been attacked is equivalent to determining whether the path described by the log file is in the path semantics of the attack tree or not. The idea behind SM is different: it is useful to determine whether a strategy is winning or not for a given attack objective. We start to analyse the complexity of PM and take advantage of it for the proofs of the other results. We then consider SNE. After that, PNE is easily determined as a particular case of SNE and we finish by SM whose proof uses similar and simpler constructions than the one for SNE.

If we use attack trees with preconditions, the problem PM is NP-hard; this comes from the fact that the *packed interval covering problem*, which can be easily captured by the parallel composition (see [13]), is NP-complete (see [14]). However, PM becomes simpler if we discard preconditions:

Theorem 5.3. *PM is in* P.

For a polynomial algorithm, we use the fact that a word is in the semantics of an attack tree, then adding an arbitrary prefix to it keeps it in the semantics. As a consequence, we do not need to recompute which sub-goals of the attack tree are satisfied whenever we add a position in front of a path. Thus, the shape of the problem is well-suited for a backward induction over the input path. Moreover, determining if a given input path satisfies an attack tree knowing whether it satisfies the sub-trees can be done in linear time over the size of the attack tree.

We now turn to the complexity of SNE.

Theorem 5.4. *SNE is* PSPACE-*complete.*

For the membership, we construct an alternating algorithm (see [2]) solving the problem that can be executed in polynomial time. This algorithm consists of synthesizing a history over the game arena and then verifying that this history is an attack (by Theorem 5.3, this verification is doable in polynomial time). To construct this history, we finitely iterate first to make a non-deterministic existential guess for the action of Attacker and then a non-deterministic universal guess for the action of Defender. We then show that the resulting history is in the path semantics of the input attack tree τ if, and only if, the strategy semantics of τ is not empty. We guarantee a polynomial time execution, namely that the resulting history need not be too long with the following lemma.

Lemma 5.5. *Let $\mathscr{G} = (Pos, Act, \delta, val)$ be a game arena and τ be an attack tree with n leaves. If $Strat_{\mathscr{G}}(\tau) \neq \emptyset$, then there exists $T \in Strat_{\mathscr{G}}(\tau)$ of depth $d \leq |Pos| \times n$.*

The basic idea behind to prove Lemma 5.5 is that, memoryless strategies suffice in reachability games (see [4]).

We design Algorithm 1 to solve SNE whose idea is explained above and show that it belongs to PSPACE.

Algorithm 1 $SNE(\mathscr{G}, \tau)$

```
Input: $\mathscr{G}$ a game arena and $\tau$ an attack tree with $n$ leaves
Output: $True$ if $Strat_{\mathscr{G}}(\tau) \neq \emptyset$, $False$ otherwise.
1: $h \leftarrow$ empty list
2: $v \leftarrow$ [$\exists$]guess position in $Pos$
3: $h.append(v)$
4: while $size(h) < |Pos| \times n$ do
5:     [$\exists$]guess break or not
6:     $a_1 \leftarrow$ [$\exists$]guess action in $Act_A$
7:     $a_2 \leftarrow$ [$\forall$]guess action in $Act_D$
8:     $h.append(\delta(last(h), (a_1, a_2)))$
9: end while
10: return $h \in Paths_{\mathscr{G}}(\tau)$
```

Lemma 5.6. *Algorithm* 1 *is an alternating polynomial-time algorithm and solves SNE.*

Proof. We start by showing the complexity of the algorithm, then we show its correctness.

From the loop at Line 4, it is executed polynomially many times in the size of the input attack tree and of the game arena. We also know (Theorem 5.3) that the condition $h \in Paths_{\mathscr{G}}(\tau)$ at Line 10 can be evaluated in polynomial, therefore, Algorithm 1 is polynomial-time alternating.

Assume Algorithm 1 returns *True*, then, for each choice made by universal guess, there exists a choice made by existential guess guaranteeing that the obtained history is in $Paths_{\mathscr{G}}(\tau)$. As a consequence, the choices made by the existential guesses reflect a strategy in the game arena that satisfies τ so, $Strat_{\mathscr{G}}(\tau) \neq \emptyset$. Conversely, if $Strat_{\mathscr{G}}(\tau) \neq \emptyset$, then there exists (by Lemma 5.5) an s-tree T of depth $\leq |Pos| \times n$. Thus the existential guesses can simply follow the strategy given by T and then choose to go out from the main loop by the "break" command at Line 5 of Algorithm 1 whenever the sequence of choices (existential and universal) in the execution is reflected by a full branch of the s-tree T.

□

For the PSPACE-hardness of SNE, our construction is inspired by the one in [1]: the authors reduce (in polynomial time) the SAT problem to the PNE problem with attack trees (using preconditions). In fact, even if in that paper, authors use attack trees with preconditions, we can adapt it without preconditions. We can even cast the approach to QBF that we first recall:

Definition 5.7. The *quantified Boolean formula* (QBF) is the following decision problem:
Input: a formula of the form $Q_1x_1,...,Q_nx_n\psi(x_1,...,x_n)$ with $Q_i \in \{\exists,\forall\}$ and ψ a Boolean formula in conjunctive normal form over propositions $x_1,...,x_n$.
Output: *Yes* if the input formula is true, *No* otherwise.

Lemma 5.8. *The QBF problem can be reduced to SNE in polynomial time.*

It is easy to understand the reduction principle on an example.

Example 5.9. Consider the formula $\psi = \exists x_1 \forall x_2 \exists x_3, x_1 \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee x_3)$. Let $C_1 = x_1$, $C_2 = (x_2 \vee x_3)$ and $C_3 = (\neg x_2 \vee x_3)$ be the three clauses in ψ. The game arena $\mathscr{G}$ associated with this formula is drawn in Figure 6: for each position v_i (resp. $\neg v_i$), the proposition p_i holds if $v_i \in C_i$ (resp. $\neg v_i \in C_i$). Remark that this game arena is a special case of game arena called turn-based game arena: only one player makes an action in each position, we say that a position belongs to the player who can play on it. We decide classically which position belongs to each player based on quantifiers of ψ (see the proof of Lemma 5.8 for further explanations). We represent Attacker positions with a circle and Defender positions with a square (position v_3 and position $\neg pos_3$ have only one successor position, therefore, it does not matter which player makes the move; by convention, we say they belong to the attacker). Then, ψ holds if, and only if, $Strat_{\mathscr{G}}(SAND(start, AND(p_1, p_2, p_3))) \neq \emptyset$.

We now start the proof of lemma 5.8:

Proof. Let $Q_1x_1,...,Q_nx_n\psi(x_1,...,x_n)$ with $Q_i \in \{\exists,\forall\}$ and with ψ a Boolean formula over variables $x_1,...,x_n$ be an instance of the QBF problem. Since ψ is in conjunctive normal form, we can write it as $\psi = \psi_1 \wedge ... \wedge \psi_k$ with ψ_i denoting disjunctive clauses containing literals of the form x_j or $\neg x_j$ with $x_i \in \{x_1,...,x_n\}$.

We consider the set of propositions $Prop = \{Start, p_1,...,p_k\}$ with the following game arena: $\mathscr{G} = (Pos, Act, \delta, val)$, where $Pos = \{Start\} \cup \{v_i | 1 \leq i \leq n\} \cup \{\neg v_i | 1 \leq i \leq n\}$, $Act_A = Act_D = \{True, False\}$. If $Q_0 = \exists$, then position *Start* is an Attacker position, otherwise, it's a defender position. Moreover, playing action *True* at position *start* leads to position v_1 while playing *False* leads to position $\neg pos_1$. Similarly, for each $2 \leq i \leq n$, if $Q_i = \exists$ then v_{i-1} and $\neg v_{i-1}$ are Attacker positions, otherwise, they are Defender positions. Furthermore, playing *True* at position v_{i-1} or $\neg v_{i-1}$ leads to position v_i

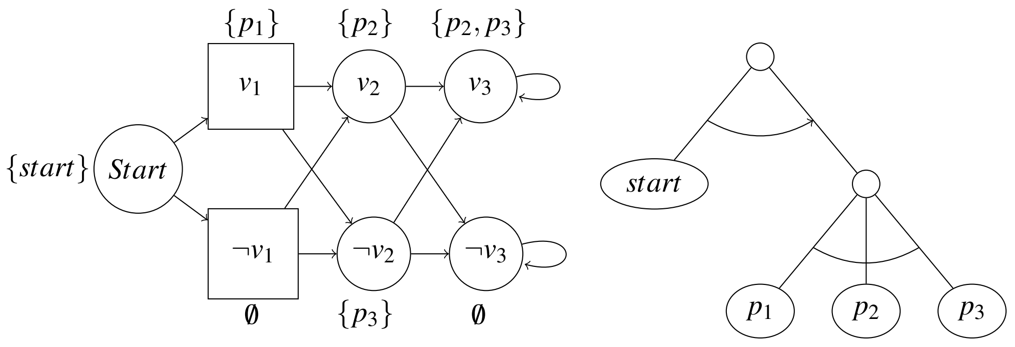


Figure 6: Game arena and attack tree associated to the formula given in Example 5.9

while playing *False* leads to $\neg v_i$. Positions v_n and $\neg v_n$ are Attacker positions, moreover, the transitions over those two positions are self loops.

We define $val(Start) = \{Start\}$ and for each $i \leq i \leq n$, $val(v_i) = \{p_j | x_i \in \psi_j\}$ and $val(\neg v_i) = \{p_j | \neg x_i \in \psi_j\}$. From this definition, if we consider that the attacker tries to satisfy the input QBF formula and the defender tries to prevent it, we have a classic game. We then only need to show that the objective of the attacker can be well described using an attack tree, which is the case by considering $\tau = SAND(Start, AND(p_1, ..., p_n))$. Indeed, if there exists a strategy to satisfy the input QBF formula, then this strategy satisfies $\psi_1, ..., \psi_k$ and thus, can be executed in the constructed game arena to achieve $AND(p_1, ..., p_n)$ while starting at position *Start*, therefore, that strategy is in τ. Conversely, if $Strat_{\mathcal{G}}(\tau) \neq \emptyset$, then one of such strategies assures that we satisfy the input QBF instance. □

By Lemma 5.8 , SNE is PSPACE-hard, which achieves the proof of Theorem 5.4.

We now turn to PNE.

Theorem 5.10. *PNE is* NP*-complete.*

For the NP-membership, since our problem is a particular case of the problem discussed in [1], it is at least as easy. For the NP-hardness we reduce SAT: if we apply the same construction as in the proof of Lemma 5.8, since we cannot leave any choice for the defender in a transition system and the path semantics is defined over a transition system and not a game arena, we can reduce formulas of QBF only using $\exists$ operators. In other words, we can reduce SAT. In fact, by doing so, we are doing the exact construction of the proof in [1]. Moreover the attack tree with preconditions $AND(< start, \phi_1 > , ..., < start, \phi_n >)$ used in that paper is completely equivalent to $SAND(start, AND(\phi_1, ..., \phi_n))$ in our formalism. Thus the proof in [1] can be well adapted for our problem.

Lastly, we study SM.

Theorem 5.11. *SM is* CONP*-complete.*

For the membership, we can use the same idea as for the membership of the SNE except that, now, we already know the strategy of the attacker, we thus do not need to use any existential guess for the action of Attacker. In other words, it is equivalent to simply considering Defender choosing a branch of the attack tree and then verifying if it forms an attack or not. Therefore, we use a variant of Algorithm 1 without existential choices, this gives us a CONP algorithm.

For the hardness, we still use the idea of the construction behind the SNE, but now, we consider that only the actions of Defender matter in the progress of the game arena. This way, we can reduce the UNSAT problem, known to be CONP-complete, to SM. The UNSAT problem is nothing less than the sub-problem of the QBF problem where an instance of the problem only uses "$\forall$" quantifiers.

This concludes the discussion over decision problems; our results are summarised in Table 1.

6 Future work

In this paper, we proposed a strategy semantics for attack trees, useful to tackle some practical questions (SNE and SM) not expressible with standard semantics provided by the literature. The price to pay is to renounce a compositional semantics of attack trees. One way to regain it might be to consider a strategy semantics based on a tree automata: we associate with each attack tree a tree automaton recognising its strategy semantics. This is currently work. Moreover, being able to consider automata recognising the strategy semantics allows us to model attack scenarios with constraints, for example, considering that the attacker cannot perform a given action more than a certain amount of time.

Moreover, we are currently exploring the possibility to expand the path and the strategy semantics to attack-defense trees. The main idea is to consider a counter operator in attack trees. This generalisation could lead to a better understanding of the differences between the strategy semantics and the attack-defence tree formalism.

References

[1] Maxime Audinot, Sophie Pinchinat & Barbara Kordy (2017): *Is my attack tree correct?* In: *European Symposium on Research in Computer Security*, Springer, pp. 83–102, doi:10.1007/978-3-319-66402-6_7.

[2] Ashok K Chandra & Larry J Stockmeyer (1976): *Alternation.* In: *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, IEEE, pp. 98–108, doi:10.1109/SFCS.1976.4.

[3] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison & Marc Tommasi (2008): *Tree automata techniques and applications.*

[4] Luca De Alfaro, Thomas A Henzinger & Orna Kupferman (2007): *Concurrent reachability games. Theoretical computer science* 386(3), pp. 188–217, doi:10.1016/j.tcs.2007.07.008.

[5] Ross Horne, Sjouke Mauw & Alwen Tiu (2017): *Semantics for specialising attack trees based on linear logic. Fundamenta Informaticae* 153(1-2), pp. 57–86, doi:10.3233/FI-2017-1531.

[6] Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Saša Radomirović & Rolando Trujillo-Rasua (2015): *Attack trees with sequential conjunction.* In: *IFIP International Information Security and Privacy Conference*, Springer, pp. 339–353, doi:10.1007/978-3-319-18467-8_23.

[7] Aivo Jürgenson & Jan Willemson (2008): *Computing exact outcomes of multi-parameter attack trees.* In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, pp. 1036–1051, doi:10.1007/978-3-540-88873-4_8.

[8] Barbara Kordy, Sjouke Mauw, Saša Radomirović & Patrick Schweitzer (2010): *Foundations of attack–defense trees.* In: *International Workshop on Formal Aspects in Security and Trust*, Springer, pp. 80–95, doi:10.1007/978-3-642-19751-2_6.

[9] Barbara Kordy, Sjouke Mauw, Saša Radomirović & Patrick Schweitzer (2014): *Attack–defense trees. Journal of Logic and Computation* 24(1), pp. 55–87, doi:10.1093/logcom/exs029.

[10] Sjouke Mauw & Martijn Oostdijk (2005): *Foundations of attack trees.* In: *International Conference on Information Security and Cryptology*, Springer, pp. 186–198, doi:10.1007/11734727_17.

[11] Soumya Paul, Ramaswamy Ramanujam & Sunil Simon (2015): *Automata and compositional strategies in extensive form games.* In: *Models of Strategic Reasoning*, Springer, pp. 174–201, doi:10.1007/978-3-662-48540-8_6.

[12] Sophie Pinchinat, Barbara Fila, Florence Wacheux & Yann Thierry-Mieg (2019): *Attack trees: a notion of missing attacks*. In: *International Workshop on Graphical Models for Security*, Springer, pp. 23–49, doi:10.1007/978-3-030-36537-0_3.

[13] Sophie Pinchinat, François Schwarzentruber & Sébastien Lê Cong (2020): *Library-Based Attack Tree Synthesis*. In: *International Workshop on Graphical Models for Security*, Springer, pp. 24–44, doi:10.1007/978-3-030-62230-5_2.

[14] Abdallah Saffidine, Sébastien Lê Cong, Sophie Pinchinat & François Schwarzentruber (2019): *The Packed Interval Covering Problem is NP-complete*. *arXiv preprint arXiv:1906.03676*.

[15] Bruce Schneier (1999): *Attack trees*. *Dr. Dobb's journal* 24(12), pp. 21–29, doi:10.1002/9781119183631.ch21.

[16] Wojciech Wideł, Maxime Audinot, Barbara Fila & Sophie Pinchinat (2019): *Beyond 2014: Formal Methods for Attack Tree–based Security Modeling*. *ACM Computing Surveys (CSUR)* 52(4), pp. 1–36, doi:10.1145/3331524.