Is my attack tree correct?

Maxime Audinot, Sophie Pinchinat, Barbara Kordy

Is my attack tree correct?

Maxime Audinot[1,2], Sophie Pinchinat[1,2], and Barbara Kordy[1,3]

[1] IRISA, Rennes, France
[2] University Rennes 1, Rennes, France
[3] INSA Rennes, France

Abstract. Attack trees are a popular way to represent and evaluate potential security threats on systems or infrastructures. The goal of this work is to provide a framework allowing to express and check whether an attack tree is consistent with the analyzed system. We model real systems using transition systems and introduce attack trees with formally specified node labels. We formulate the correctness properties of an attack tree with respect to a system and study the complexity of the corresponding decision problems. The proposed framework can be used in practice to assist security experts in manual creation of attack trees and enhance development of tools for automated generation of attack trees.

1 Introduction

An attack tree is a graphical model allowing a security expert to illustrate and analyze potential security threats. Thanks to their intuitiveness, attack trees gained a lot of popularity in the industrial sector [15], and organizations such as NATO [24] and OWASP [20] recommend their use in threat assessment processes. The root of an attack tree represents an attack objective, *i.e.,* an attacker's goal, and the rest of the tree decomposes this goal into sub-goals that the attacker may need to reach in order to perform his attack [26]. In this paper, we develop a formal framework to evaluate *how well an attack tree describes the attacker's goal with respect to the system that is being analyzed*. This work has been motivated by the two following practical problems.

First, in the industrial context, attack trees are usually created manually by security experts who may not have an exhaustive knowledge about all the facets (technical, social, physical) of the analyzed system. This process is often supported by the use of libraries containing generic models for standard security threats. Although using libraries provides a good starting point, the resulting attack tree may not always be fully consistent with the system that is being analyzed. This problem might be reinforced by the fact that the node names in attack trees are often very short, and may thus lack precision or be inaccurate and misleading. If the tree is incomplete or imprecise, the results of its analysis (*e.g.,* estimation of the attack's cost or its probability) might be inaccurate. If the tree contains branches that are irrelevant for the considered system, the time of its analysis might be longer than necessary. This implies that a manually created tree needs to be validated against a system to be analyzed before it can be used as a formal model on which the security of the system will be evaluated.

Second, to limit the burden of their manual creation, several academic proposals for automated generation of attack trees have recently been made [30,23,11]. In particular,

we are currently developing the ATSyRA tool for assisted generation of attack trees from system models [23]. Our experience shows that, due to the complexity and scalability issues, a fully automated generation is impossible. Some generation steps must thus be supported by humans. Such a semi-automated approach gives the expert a possibility of manually decomposing a goal, in such a way that an automated generation of the subtrees can be performed. This work provides formal foundations for the next version of our tool which will assist the expert in producing trees that, by design, are correct with respect to the underlying system.

Contribution. To address the problems identified above, we introduce a mathematical framework allowing us to formalize the notion of attack trees and to define as well as verify their practically-relevant correctness properties with respect to a given system. We model real-life systems using finite transition systems. The attack tree nodes are *labeled with formally specified goals formulated in terms of preconditions and postconditions* over the possible states of the transition system. Formalizing the labels of the attack tree nodes allows us to overcome the problem of imprecise or misleading text-based node names and makes formal treatment of attack trees possible. We define the notion of *Admissibility* of an attack tree with respect to a given system and introduce the correctness properties for attack trees, called *Meet*, *Under-Match*, *Over-Match*, and *Match*. These properties express the precision with which a given goal is refined into sub-goals with respect to a given system. We then *establish the complexity of verifying the correctness properties* to apprehend the nature of potential algorithmic solutions to be implemented.

Related work. In order to use any modeling framework in practice, formal foundations are necessary. Previous research on formalization of attack trees focused mainly on mathematical semantics for attack tree-based models [19,13,14,12,10], and various algorithms for their quantitative analysis [25,16,1]. However, all these formalizations rely on an action-based approach, where the attacker's goals represented by the labels of the attack tree nodes are expressed using actions that the attacker needs to perform to achieve his/her objective. In this work, we pioneer a state-based approach to attack trees, where the attacker's goals relate to the states of the modeled system. The advantage of such a state-based approach is that it may benefit from verification and model checking techniques, in a natural way, as this has already been done in the case of attack graphs [28,21]. In our framework, the label of each node of an attack tree is formulated in terms of preconditions and postconditions over the states of the modeled system: intuitively speaking, the goal of the attacker is to start from any state in the system that satisfies the preconditions and reach a state where the postconditions are met. The idea of formalizing the labels of attack tree nodes in terms of preconditions and postconditions has already been explored in [22]. However there, the postcondition (*i.e.,* consequence) of an action is represented by a parent node and its children model the preconditions and the action itself.

Model checking of attack trees, especially using tools such as PRISM or UPPAAL, has already been successfully employed, in particular to support their quantitative analysis, as in [8,17,2]. Such techniques provide an effective way of handling a multi-parameter evaluation of attack scenarios, *e.g.,* identifying the resources needed for a

successful attack or checking whether there exists an attack whose cost is lower than a given value and whose probability of success is greater than a certain threshold. However, these approaches either do not consider any particular system beforehand, or they rely on a model of the system that features explicit quantitative aspects.

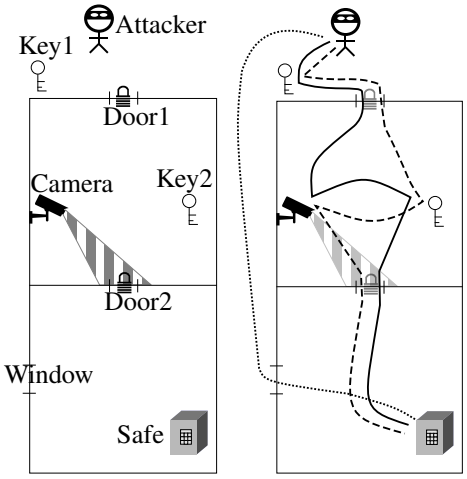
The link between the analyzed system and the corresponding attack tree is made explicit in works dealing with automated generation of attack trees from system models [11,23]. The systems considered in [11] capture locations, assets, processes, policies, and actors. The goal of the attacker is to reach a given location or obtain an asset, and the attack tree generation algorithm relies on invalidation of policies that forbid him to do so. In the case of [23], the ATSyRA tool is used to effectively generate a transition system for a real-life system: starting from a domain-specific language describing the original system, ATSyRA compiles this description into a symbolic transition system specified in the guarded action language GAL [29]. ATSyRA can already handle the physical layer of a system (locations and connections/accesses between them) and we are currently working on extending it with the digital layer. Since our experience shows that generating a transition system from a description in a domain-specific language is possible and efficient, in this paper we suppose that the transition system for a real system has been previously created and is available.

Finally, to the best of our knowledge, the problem of defining and verifying the correctness of an attack tree with respect to the analyzed system has only been considered in [3] which has been the starting point for the work presented in this paper.

2 Motivating example

Before presenting our framework, we first introduce a motivating example on which we will illustrate the notions and concepts employed in this paper.

The system modeled in our running example is a building containing a safe holding a confidential document. The goal of the attacker is to reach the safe without being detected. We purposely keep this example small and intuitive to ease the understanding of the proposed framework. The floor plan of the building is depicted in Fig. 1a. It contains two rooms, denoted by Room1 and Room2, two doors – Door1 allowing to move from outside of the building to Room1 and Door2 connecting Room1 and Room2 – as well as one window in Room2. Both doors are initially locked and it is left unspecified whether the window is open or not.



(a) Floor plan (b) Attack scenarios

Fig. 1: Running example building

Such unspecified information expresses that the analyst cannot predict whether the window will be open or closed in the case of a potential attack or that he has a limited knowledge about the system. In both cases, this lack of information needs to be taken into account during the analysis process. The two doors can be unlocked by means of Key1 and Key2, respectively. We assume that a camera that monitors Door2 is located in Room1. The camera is initially on but it can be switched off manually. The safe is in Room2.

The attacker is located outside of the building and his goal is to *reach the safe without being detected by the camera*. In Fig. 1b, we have depicted three scenarios (that we will call paths) allowing the attacker to reach his goal. In the first scenario (depicted using dotted line), the attacker goes straight through the window, if it is open. In the remaining two scenarios, the attacker gathers the necessary keys and goes through the two doors, switching off the camera on his way. These two scenarios differ only in the order in which the concurrent actions are sequentially performed. Since collecting Key2 and switching off the camera are independent actions, the attacker can first collect Key2 and then switch the camera off (dashed line), or switch the camera off before collecting Key2 (solid line).

The *system* in our example consists of the building and the attacker. It is modeled using state variables whose values determine possible configurations of the system.

- `Position` – variable describing the attacker's position, ranging over {`Outside`, `Room1`, `Room2`};
- `WOpen` – Boolean variable describing whether the window is open (`tt`) or not (`ff`);
- `Locked1` and `Locked2` – Boolean variables to describe whether the respective doors are locked or not;
- `Key1` and `Key2` – Boolean variables to describe whether the attacker possesses the respective key;
- `CamOn` – Boolean variable describing if the camera is on;
- `Detected` – Boolean variable to describe if the camera detected the attacker, *i.e.,* whether the attacker has crossed the area monitored by the camera while it was on.

Given a set of state variables, we express possible configurations of a system using propositions. *Propositions* are either equalities of the form `state_variable=value` or Boolean combinations of such equalities. Intuitively, a proposition expresses a constraint on the possible configurations. A configuration in which all the variables are left unspecified is called the *empty configuration*. We denote it by $\top$.

In order to analyze the security of a system, security experts often use the model of attack trees. An *attack tree* is a tree in which each node represents an attacker objective, and the children of a node represent a decomposition of this objective into sub-objectives. In this work, we consider attack trees with three types of nodes:

- `OR` nodes representing alternative choices – to achieve the goal of the node, the attacker needs to achieve the goal of at least one child;
- `AND` nodes representing conjunctive decomposition – to achieve the goal of the node, the attacker needs to achieve all of the goals represented by its children (the children of an `AND` node are connected with an arc);

– SAND nodes representing sequential decomposition – to achieve the goal of the node, the attacker needs to achieve all of the goals represented by its children in the given order (the children of a SAND node are connected with an arrow).

The attack tree given in Fig. 2 illustrates that in order to enter Room2 undetected (root node of type OR), the attacker can either enter through the window or through the doors. In order to use the second alternative (node of type AND), he needs to make sure that the camera is deactivated and that he reaches Room2. To achieve the last objective (node of type SAND), he first needs to unlock Room1, then unlock Room2, and finally enter to Room2.

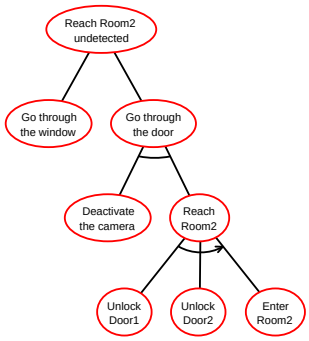


Fig. 2: Attack tree with informal, text-based node names

One of the most problematic aspects of attack trees are the informal, text-based names of their nodes. These names are often very short and thus do not express all the information that the tree author had in mind while creating the tree. In particular, the textual names relate to the objective that the attacker should reach, however, they usually do not capture the information about the initial situation from which he starts.

To overcome the weakness of text-based node names, we propose to formalize the attacker's goal using two configurations: the *initial configuration*, usually denoted by ι, is the configuration before the attack starts, i.e., represents preconditions; and the *final configuration*, usually denoted by γ, represents postconditions, *i.e.,* the state to be reached to succeed in the attack. The goal with initial configuration ι and final configuration γ is written $\langle \iota, \gamma \rangle$.

In our running example, the initial configuration is $\iota := (\texttt{Position} = \texttt{Outside}) \wedge (\texttt{Key1} = \texttt{ff}) \wedge (\texttt{Key2} = \texttt{ff}) \wedge (\texttt{Locked1} = \texttt{tt}) \wedge (\texttt{Locked2} = \texttt{tt}) \wedge (\texttt{CamOn} = \texttt{tt})$. It describes that the attacker is originally outside of the building, he does not have any of the keys, the two doors are locked, and the camera is on. The final configuration is $\gamma := (\texttt{Position} = \texttt{Room2}) \wedge (\texttt{Detected} = \texttt{ff})$, *i.e.,* the attacker reached Room2 without being detected.

Fig. 3 illustrates how such formally specified goals are used to label the nodes of attack trees. The goal $\langle \iota, \gamma \rangle$ introduced above is the label of the root node of the tree. It is then refined into sub-goals $\langle \iota_i, \gamma_i \rangle$, where i reflects the position of the node in the tree.
Sub-goal $\langle \iota_1, \gamma_1 \rangle$: The attacker, who wants to reach the safe in Room2 without being detected, is located outside of the building and the window is initially open. We let $\iota_1 := (\texttt{Position} = \texttt{Outside}) \wedge (\texttt{Key1} = \texttt{ff}) \wedge (\texttt{Key2} = \texttt{ff}) \wedge (\texttt{Locked1} = \texttt{tt}) \wedge (\texttt{Locked2} = \texttt{tt}) \wedge (\texttt{CamOn} = \texttt{tt}) \wedge (\texttt{WOpen} = \texttt{tt})$ and $\gamma_1 := \gamma$.

Sub-goal $\langle \iota_2, \gamma_2 \rangle$: This sub-goal is similar to the previous one, but the window is originally closed. We let $\iota_2 := (\texttt{Position} = \texttt{Outside}) \wedge (\texttt{Key1} = \texttt{ff}) \wedge (\texttt{Key2} = \texttt{ff}) \wedge (\texttt{Locked1} = \texttt{tt}) \wedge (\texttt{Locked2} = \texttt{tt}) \wedge (\texttt{CamOn} = \texttt{tt}) \wedge (\texttt{WOpen} = \texttt{ff})$ and $\gamma_2 := \gamma$.

Sub-goal $\langle \iota_{21}, \gamma_{21} \rangle$: The attacker, who might be in any initial configuration, wants to deactivate the camera. We then let $\iota_{21} := \top$ and $\gamma_{21} :=$ (`CamOn = ff`).

Sub-goal $\langle \iota_{22}, \gamma_{22} \rangle$: Similar to sub-goal $\langle \iota_2, \gamma_2 \rangle$, with the difference that we do not care whether the camera is initially on and we no longer require that the attacker remains undetected. We let $\iota_{22} :=$ (`Position = Outside`) $\wedge$ (`Key1 = ff`) $\wedge$ (`Key2 = ff`) $\wedge$ (`Locked1 = tt`) $\wedge$ (`Locked2 = tt`) $\wedge$ (`WOpen = ff`) and $\gamma_{22} :=$ (`Position = Room2`).

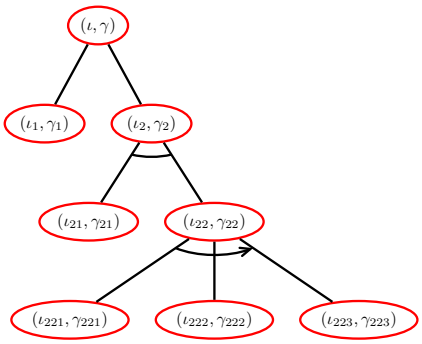


Fig. 3: Attack tree with formal labels

Sub-goal $\langle \iota_{221}, \gamma_{221} \rangle$: The initial situation is the same as in the sub-goal $\langle \iota_{22}, \gamma_{22} \rangle$, but we require that the attacker unlocks Door1 but not Door2: $\iota_{221} := \iota_{22}$ and $\gamma_{221} :=$ (`Locked1 = ff`) $\wedge$ (`Locked2 = tt`).

Sub-goal $\langle \iota_{222}, \gamma_{222} \rangle$: Now, the objective is to go from a state where Door1 is unlocked and Door2 is locked (like in the configuration γ_{221}) to a state where both doors are unlocked. We let $\iota_{222} := \gamma_{221}$ and $\gamma_{222} :=$ (`Locked1 = ff`) $\wedge$ (`Locked2 = ff`).

Sub-goal $\langle \iota_{223}, \gamma_{223} \rangle$: Finally, the last sub-goal is for the attacker, starting in a state where both doors are unlocked, to reach Room2. We let $\iota_{223} := \gamma_{222}$ and $\gamma_{223} := \gamma_{22}$.

3 Formal modeling

We now provide formal notations and definitions of transition systems and attack trees that we have informally described in Sect. 2.

3.1 Transition systems

We model real-life systems using finite transition systems. Transition system is a simple, yet powerful formal tool to represent a dynamic behavior of a system by listing all its possible states and transitions between them. The finiteness of the state transition system is a reasonable and realistic assumption. A formal model can either be finite because the real-life underlying system is intrinsically finite, or it can have a finite representation obtained by standard abstraction techniques, as used in verification, static analysis, and model-checking.

We fix the set Prop of propositions that we use to formalize possible configurations of the real system. In the rest of the paper, we suppose that Prop contains propositions of the form ι, γ, to denote preconditions (ι) and postconditions (γ) of the goals.

Definition 1 (Transition system).
A transition system *over* Prop *is a tuple* $\mathcal{S} = (S, \rightarrow, \lambda)$*, where* S *is a finite set of* states

(elements of S are denoted by s, s_i for $i \in \mathbb{N}$), $\rightarrow \subseteq S \times S$ is the transition relation *of the system (which is assumed left-total), and $\lambda : \text{Prop} \rightarrow 2^S$ is the* labeling *function. We say that a state s* is labeled by *p when $s \in \lambda(p)$. The* size *of $\mathcal{S}$ is $|\mathcal{S}| = |S| + |\rightarrow|$.*

For the rest of this paper, we assume that we are given a transition system $\mathcal{S}$ over Prop. A *path* in $\mathcal{S}$ is a non-empty sequence of states. We use typical elements $\pi, \pi', \pi_1, \ldots, \rho, \ldots$ to denote paths. The *size* of a path π, denoted by $|\pi|$, is its number of transitions, and $\pi(i)$ is the element at position i in π, for $0 \leq i \leq |\pi|$. An empty path[4] is a path of size 0. We write $\Pi(\mathcal{S})$ for the set of all paths in $\mathcal{S}$. For $\iota, \gamma \in$ Prop, we shortly say that a path π "goes from ι to γ" whenever $\pi(0) \in \lambda(\iota)$ and $\pi(|\pi|) \in \lambda(\gamma)$. The set of *direct successors* of a set of states $S' \subseteq S$ is $Post_{\mathcal{S}}(S') = \{s \in S \mid \exists s' \in S' \text{ such that } (s', s) \in \rightarrow\}$. The set of *successors* of a set of states $S' \subseteq S$ is $Post^*_{\mathcal{S}}(S') = \{s \in S \mid \exists \pi \text{ with } \pi(0) \in S' \text{ and } \pi(|\pi|) = s\}$, and the set of *predecessors* of $S' \subseteq S$ is $Pre^*_{\mathcal{S}}(S') = \{s \in S \mid \exists \pi \text{ with } \pi(0) = s \text{ and } \pi(|\pi|) \in S'\}$.

A factor of a path π is a subsequence composed of consecutive elements of π. Formally, a *factor* of a path π is a path π', such that there exists $0 \leq k \leq |\pi| - |\pi'|$, where $\pi(i + k) = \pi'(i)$, for $0 \leq i \leq |\pi'|$. An *anchoring* of π' in π is an interval $[k, l] \subseteq [0, |\pi|]$ where for all $i \in [k, l]$, $\pi'(i - k) = \pi(i)$ and $l - k = |\pi'|$. Notice that we may have $|\pi'| = 0$. We denote by $\pi[k, l]$ the factor of π of anchoring $[k, l]$. In other words, the anchorings of π' in π are the intervals $[k, l]$ of positions in π such that $\pi[k, l] = \pi'$.

We now introduce concatenation and parallel decomposition of paths – two notions that will serve us to define the semantics of sequential and conjunctive refinements in attack trees, respectively.

Definition 2 (Concatenation of paths). *Let $\pi_1, \pi_2, \ldots, \pi_n \in \Pi(\mathcal{S})$ be paths, such that $\pi_i(|\pi_i|) = \pi_{i+1}(0)$ for $0 \leq i < n - 1$. The* concatenation *of $\pi_1, \pi_2, \ldots, \pi_n$, denoted by $\pi_1.\pi_2.\ldots.\pi_n$, is the path π, where $\pi[\sum_{k=1}^{i-1} |\pi_k|, \sum_{k=1}^{i-1} |\pi_k| + |\pi_i|] = \pi_i$[5]. We generalize the concatenation to sets of paths by letting $\Pi.\Pi' = \{\pi \in \Pi(\mathcal{S}) \mid \exists i, 0 \leq i \leq |\pi| \text{ and } \pi[0, i] \in \Pi \text{ and } \pi[i, |\pi|] \in \Pi'\}$.*

Definition 3 (Parallel decomposition of paths). *A set $\{\pi_1, \ldots, \pi_n\} \subseteq \Pi(\mathcal{S})$ is a* parallel decomposition *of $\pi \in \Pi(\mathcal{S})$ if for every $1 \leq i \leq n$ the path π_i is a factor of π for some anchoring $[k_i, l_i]$, such that every interval $[j, j + 1] \subseteq [0, |\pi|]$ is contained in $[k_i, l_i]$ for some $i \in \{1, \ldots, n\}$ (which trivially holds if $|\pi| = 0$). We then say that the sequence $\pi_1, \ldots, \pi_n$ is a* parallel decomposition *of π for the anchorings $[k_1, l_1], \ldots, [k_n, l_n]$.*

Lemma 1. *Given a path $\pi \in \Pi(\mathcal{S})$, and a sequence $k_1, l_1, \ldots, k_n, l_n \in [0, |\pi|]$, deciding whether $\pi[k_1, l_1], \ldots, \pi[k_n, l_n]$ is a parallel decomposition of π for the anchorings $[k_1, l_1], \ldots, [k_n, l_n]$ can be done in time $O(n\,|\pi|)$.*

Proof. Verifying that $\pi[k_1, l_1], \ldots, \pi[k_n, l_n]$ is a parallel decomposition of π for the anchorings $[k_1, l_1], \ldots, [k_n, l_n]$ amounts to checking that for every interval $[j, j + 1] \subseteq [0, |\pi|]$, there is an $i \in [1, n]$ such that $[j, j + 1] \subseteq [k_i, l_i]$. This can clearly be done in time $O(n\,|\pi|)$ by a naive approach.

An example of a parallel decomposition is illustrated in Fig. 4, where $\pi_1 = \pi[0, 2]$, $\pi_2 = \pi[3, 5]$, and $\pi_3 = \pi[1, 4]$.

[4] Since a path is a non-empty sequence of states, the empty path contains exactly one state.

[5] We use the convention that $\sum_{k=1}^{0} |\pi_k| = 0$.

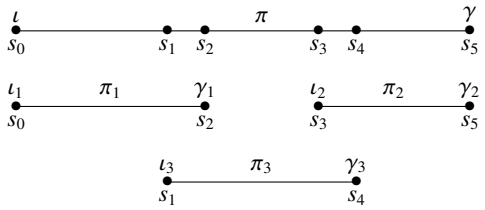


Fig. 4: Parallel decomposition of π into $\{\pi_1, \pi_2, \pi_3\}$.

A *cycle* in a path $\pi \in \Pi(\mathcal{S})$ is a factor π' of π such that $\pi'(0) = \pi'(|\pi'|)$. An *elementary path* is a path with no cycle. Remark that an elementary path π does not contain any state more than once, so $|\pi| \leq |\mathcal{S}|$. Removing a cycle π' of anchoring $[k, l]$ from a path π yields the path $\pi[0, k].\pi[l, |\pi|]$. Removing all the cycles from π consists in iteratively removing cycles until the resulting path is elementary. Note that the resulting path may depend on the order in which the cycles are removed.

We illustrate the notions defined in this section on our running example.

Example 1. We use the state variables introduced in Sect. 2 to describe the states of a part of our building system. By z_0 we denote the state where `Position = Outside` (the attacker is outside); `WOpen = ff` (the window is closed); `Locked1 = Locked2 = tt` (both doors are locked); `Key1 = Key2 = ff` (the attacker does not have any key); `CamOn = tt` (the camera is on); `Detected = ff` (the attacker has not been detected). Furthermore, we consider seven additional states z_i, such that, for every $1 \leq i \leq 7$, the specification of z_i is the same as the specification of z_{i-1}, except one variable: state z_1 is as z_0 but `Key1 = tt` (the attacker has Key1); state z_2 is as z_1 but `Locked1 = ff` (Door1 is unlocked); state z_3 is as z_2 but `Position = Room1` (the attacker is in Room1); z_4 is as z_3 but `CamOn = ff` (the camera is off); z_5 is as z_4 but `Key2 = tt` (the attacker has Key2); state z_6 is as z_5 but `Locked2 = ff` (Door2 is unlocked); state z_7 is as z_6 but `Position = Room2` (the attacker is in Room2).

To model the dynamic behavior of the system, we set $(z_{i-1}, z_i) \in \rightarrow$, for all $1 \leq i \leq 7$. Given p = (`Position = Outside`)$\wedge$(`Locked1 = tt`) and p' = (`Position = Room1`)$\vee$ (`Position = Room2`), we have $z_0, z_1 \in \lambda(p)$ and $z_i \in \lambda(p')$, for $3 \leq i \leq 7$.

The path $\rho = z_0z_1z_2z_3z_4z_5z_6z_7$, corresponds to the scenario depicted using solid line in Fig. 1b. The set $\{z_0z_1z_2z_3z_4,\ z_3z_4z_5z_6z_7\}$ is an example of parallel decomposition of ρ. To show that while being in Room1 the attacker can turn off but also turn on the camera, we could add the transition (z_4, z_3) to $\rightarrow$. In this case, the attacker could also take the path $\rho' = z_0z_1z_2z_3z_4z_3z_4z_5z_6z_7$ which is not elementary because it contains the cycle $z_3z_4z_3$.

3.2 Attack trees

To evaluate the security of systems, we use attack trees. An attack tree does not replace the state-transition system model – it complements it with additional information on how the corresponding real-life system could be attacked. There exist a plethora of methods and algorithms for quantitative and qualitative reasoning about security using

attack trees [15]. However, accurate results can only be obtained if the attack tree is in some sense consistent with the analyzed system. Our goal is thus to validate the relevance of an attack tree with respect to a given system. To make this validation possible, we need a model capturing more information than just text-based names of the nodes. In this section, we therefore introduce a formal definition of attack trees, where the difference with the classical definition is the presence of a goal of the form $\langle \iota, \gamma \rangle$ at each node.

Definition 4 (Attack tree). *An* attack tree T *over the set of propositions* Prop *is either a leaf* $\langle \iota, \gamma \rangle$*, where* $\iota, \gamma \in$ Prop*, or a composed tree of the form* $(\langle \iota, \gamma \rangle, \mathit{OP})(T_1, T_2, \ldots, T_n)$*, where* $\iota, \gamma \in$ Prop*,* $\mathit{OP} \in \mathit{O}$ *has arity* $n \geq 2$*, and* T_1*,* T_2*, . . . ,* T_n *are attack trees. The* main goal *of an attack tree* $T = (\langle \iota, \gamma \rangle, \mathit{OP})(T_1, T_2, \ldots, T_n)$ *is* $\langle \iota, \gamma \rangle$ *and its* operator *is* OP*.*

The size of an attack tree $|T|$ *is the number of the nodes in* T*. Formally,* $|\langle \iota, \gamma \rangle| = 1$ *and* $|(\langle \iota, \gamma \rangle, \mathit{OP})(T_1, T_2, \ldots, T_n)| = 1 + \Sigma_{i=1}^{n} |T_i|$.

As an example, the tree in Fig. 3 is $T = (\langle \iota, \gamma \rangle, \mathtt{OR})(T_1, T_2)$. The subtree $T_1 = \langle \iota_1, \gamma_1 \rangle$ is a leaf and $T_2 = (\langle \iota_2, \gamma_2 \rangle, \mathtt{AND})(\langle \iota_{21}, \gamma_{21} \rangle, T_{22})$ is a composed tree with $T_{22} = (\langle \iota_{22}, \gamma_{22} \rangle, \mathtt{SAND})(\langle \iota_{221}, \gamma_{221} \rangle, \langle \iota_{222}, \gamma_{222} \rangle, \langle \iota_{223}, \gamma_{223} \rangle)$.

Before introducing properties that address correctness of an attack tree, we need to define the *path semantics* of goal expressions that arise from tree descriptions. A *goal expression* is either a mere atomic goal of the form $\langle \iota, \gamma \rangle$ or a composed goal of the form $\mathtt{OP}(\langle \iota_1, \gamma_1 \rangle, \langle \iota_2, \gamma_2 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)$, where $\mathtt{OP} \in \{\mathtt{OR}, \mathtt{SAND}, \mathtt{AND}\}$. The path semantics of a goal expression is defined as follows.

- $[\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}} = \{\pi \in \Pi(\mathcal{S}) \mid \pi \text{ goes from } \iota \text{ to } \gamma\}$
- $[\![\mathtt{OR}(\langle \iota_1, \gamma_1 \rangle, \langle \iota_2, \gamma_2 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)]\!]^{\mathcal{S}} = [\![\langle \iota_1, \gamma_1 \rangle]\!]^{\mathcal{S}} \cup [\![\langle \iota_2, \gamma_2 \rangle]\!]^{\mathcal{S}} \cup \ldots \cup [\![\langle \iota_n, \gamma_n \rangle]\!]^{\mathcal{S}}$
- $[\![\mathtt{SAND}(\langle \iota_1, \gamma_1 \rangle, \langle \iota_2, \gamma_2 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)]\!]^{\mathcal{S}} = [\![\langle \iota_1, \gamma_1 \rangle]\!]^{\mathcal{S}}.[\![\langle \iota_2, \gamma_2 \rangle]\!]^{\mathcal{S}}.\ldots.[\![\langle \iota_n, \gamma_n \rangle]\!]^{\mathcal{S}}$
- $[\![\mathtt{AND}(\langle \iota_1, \gamma_1 \rangle, \langle \iota_2, \gamma_2 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)]\!]^{\mathcal{S}} = \{\pi \in \Pi(\mathcal{S}) \mid \forall i \in \{1, \ldots, n\}\ \exists \pi_i \in [\![\langle \iota_i, \gamma_i \rangle]\!]^{\mathcal{S}},$ s.t. $\{\pi_1, \pi_2, \ldots, \pi_n\}$ is a parallel decomposition of $\pi\}$.

Consider the goal $\langle \iota, \gamma \rangle$ of our running example, and let $\mathcal{Z}$ be the system introduced in Example 1. We have $[\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}} = \{z_0 z_1 z_2 (z_3 z_4)^k z_5 z_6 z_7 \mid k \geq 1\}$, where $(z_3 z_4)^k$ is the path composed of k executions of $z_3 z_4$.

4 Correctness properties of attack trees

We now define four correctness properties for attack trees, illustrate them on our running example, and discuss their relevance for real-life security analysis.

4.1 Definitions

Before formalizing the correctness properties for attack trees, we wish to discard attack trees with "useless" nodes. To achieve this, we define the *admissibility* of an attack tree T w.r.t. the system $\mathcal{S}$.

The property that an attack tree T is *admissible* w.r.t. a system $\mathcal{S}$ is inductively defined as follows. A leaf tree $\langle \iota, \gamma \rangle$ is *admissible* whenever $[\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}} \neq \emptyset$. A composed tree $(\langle \iota, \gamma \rangle, \mathtt{OP})(T_1, \ldots, T_n)$ is *admissible* whenever three conditions hold: (a) $[\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}} \neq \emptyset$, (b) $[\![\mathtt{OP}(\langle \iota_1, \gamma_1 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)]\!]^{\mathcal{S}} \neq \emptyset$, where $\langle \iota_i, \gamma_i \rangle$ is the main goal of T_i ($1 \leq i \leq n$), and (c) every subtree T_i is admissible.

We now propose four notions of correctness, that provide various formal meanings to the local refinement of a goal in an admissible tree.

Definition 5 (Correctness properties).
Let T be a composed admissible attack tree of the form $(\langle \iota, \gamma \rangle, OP)(T_1, T_2 \ldots, T_n)$, and assume $\langle \iota_i, \gamma_i \rangle$ is the main goal of T_i, for $i \in \{1, \ldots, n\}$. The tree T has the

1. Meet property *if* $[\![OP(\langle \iota_1, \gamma_1 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)]\!]^{\mathcal{S}} \cap [\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}} \neq \emptyset$.
2. Under-Match property *if* $[\![OP(\langle \iota_1, \gamma_1 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)]\!]^{\mathcal{S}} \subseteq [\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}}$.
3. Over-Match property *if* $[\![OP(\langle \iota_1, \gamma_1 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)]\!]^{\mathcal{S}} \supseteq [\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}}$.
4. Match property *if* $[\![OP(\langle \iota_1, \gamma_1 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)]\!]^{\mathcal{S}} = [\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}}$.

Clearly the Match property implies all other properties, whereas Under- and Over-Match properties are incomparable – as illustrated in Sect. 4.2 – and they both imply the Meet property. Note that a tree T has the Match property if, and only if, it has both the Under-Match property and the Over-Match property.

The correctness properties of Definition 5 are *local* (at the root of the subtree), but they can easily be made *global* by propagating their requirement to all of the subtrees. As there are $|T|$ many subtrees, the complexity of globally deciding these properties has the same order of magnitude as in the local case.

4.2 Illustration on the running example

In the system $\mathcal{Z}$ defined in Example 1 and composed of the states $z_0, \ldots, z_7$, we add two states. First, the state z_0' that is similar to z_0 except that we assume that the window is open, *i.e.,* `WOpen = tt`, and second, the state z_7' that is similar to z_0' except that we assume that the attacker is in Room2, *i.e.,* `Position = Room2`. As a consequence the transitions of the system $\mathcal{Z}$ become $z_0' \to z_0 \to z_1 \to z_2 \to z_3 \leftrightarrow z_4 \to z_5 \to z_6 \to z_7$ and $z_0' \to z_7'$, where the latter models that if the window is open, the attacker can reach Room2 undetected by entering through the window.

Let us consider the attack tree $T(\langle \iota, \gamma \rangle, \mathtt{OR})(\langle \iota_1, \gamma_1 \rangle, T_2)$ from Fig. 3, where the main goal of T_2 is $\langle \iota_2, \gamma_2 \rangle$. Since in system $\mathcal{Z}$, the set of paths $[\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}}$ is exactly the union of $[\![\langle \iota_1, \gamma_1 \rangle]\!]^{\mathcal{S}}$ and $[\![\langle \iota_2, \gamma_2 \rangle]\!]^{\mathcal{S}}$, the tree T has the Match property w.r.t. $\mathcal{Z}$. This means that in order to achieve goal $\langle \iota, \gamma \rangle$, it is necessary and sufficient to achieve goal $\langle \iota_1, \gamma_1 \rangle$ or goal $\langle \iota_2, \gamma_2 \rangle$.

We now consider the sub-tree T_2 of T rooted at the node labeled by $\langle \iota_2, \gamma_2 \rangle$ in Fig. 3. The tree T_2 is of the form $(\langle \iota_2, \gamma_2 \rangle, \mathtt{AND})(\langle \iota_{21}, \gamma_{21} \rangle, T_2')$ where the main goal of T_2' is $\langle \iota_{22}, \gamma_{22} \rangle$. Our objective is to analyze the relationship between the main goal $\langle \iota_2, \gamma_2 \rangle$ of T_2 and the composed goal $\mathtt{AND}(\langle \iota_{21}, \gamma_{21} \rangle, \langle \iota_{22}, \gamma_{22} \rangle)$. In other words, we ask how does the aim of reaching Room2 undetected via building relates with turning off the camera

($\langle \iota_{21}, \gamma_{21} \rangle$) and reaching Room2 ($\langle \iota_{22}, \gamma_{22} \rangle$). A quick analysis of system $\mathcal{Z}$ shows that indeed achieving both subgoals $\langle \iota_{21}, \gamma_{21} \rangle$ and $\langle \iota_{22}, \gamma_{22} \rangle$ is necessary to achieve goal $\langle \iota_2, \gamma_2 \rangle$, but actually it is not sufficient. Consider the path $\delta = z'_0 z_0 z_1 z_2 z_3 z_4 z_5 z_6 z_7$. This path achieves goal $\texttt{AND}(\langle \iota_{21}, \gamma_{21} \rangle, \langle \iota_{22}, \gamma_{22} \rangle)$, as it can be decomposed into $\delta_{21} = z'_0 z_0 z_1 z_2 z_3 z_4$ and $\delta_{22} = z_0 z_1 z_2 z_3 z_4 z_5 z_6 z_7$, achieving $\langle \iota_{21}, \gamma_{21} \rangle$ and $\langle \iota_{22}, \gamma_{22} \rangle$, respectively. However, $\delta \notin [\![\langle \iota_2, \gamma_2 \rangle]\!]^{\mathcal{S}}$, since $z'_0 \notin \lambda(\iota_2)$ (recall that ι_2 requires the window to be closed which is not the case in z'_0). This is what the Over-Match property reflects. As a consequence, the main tree T does not have the global Match property w.r.t. $\mathcal{Z}$.

Symmetrically to the Over-Match property, Under-Match reflects a sufficient but not necessary condition. Under-Match is illustrated in the extended version of this work [4]. Regarding the Meet property, we invite the reader to consider the following discussion on the relevance of the correctness properties we have proposed.

4.3 Relevance of the correctness properties

The main objective of introducing the four correctness properties is to be able to validate an attack tree with respect to a system $\mathcal{S}$, *i.e.,* verify how faithfully the tree represents potential threats on $\mathcal{S}$. This is of special importance for the trees that are created manually or which are borrowed from an attack tree library.

In the perfect world, we would expect to work with attack trees having the (global) Match property, *i.e.,* where the refinement of every (sub-)goal covers perfectly all possible ways of reaching the (sub-)goal in the system. However, a tree created by a human will rarely have this property. The experts usually do not have perfect knowledge about the system and might lack information about some relevant data. Trees that have been created for similar systems are often reused but they might actually be incomplete or inaccurate with respect to the current system. Finally, requiring the (global) Match property might also be unrealistic for goals expressed only with a couple $\langle \texttt{precondition}, \texttt{postcondition} \rangle$. Therefore, Match is often too strong to be the property expected by default.

In practice, experts base their trees on some example scenarios, which implies that they obtain trees having the (global) Meet property. The Meet property – which ensures that there is at least one path in the system satisfying both the parent goal and its refinement – is the minimum that we expect from an attack tree so that we can consider that it is (in some sense) correct and so that we can start reasoning about the security of the underlying system.

However, in order to be able to perform a thorough and accurate analysis of security, one needs stronger properties to hold. One of the purposes of attack trees is to provide a summary of possible individual attack scenarios in order to quantify the security-relevant parameters, such as their cost, their time or their probability. This helps the security experts to compare and rank the different scenarios, to be able to deduce the most probable ones and propose suitable countermeasures. The classical bottom-up algorithm for quantification of attack trees, described for instance in [19], assigns the parameter values to the leaf nodes and then propagates them up to the root, using functions that depend on the type of the refinement used (in our case `OR`, `AND`, `SAND`). This means that the value of the parent node depends solely on the values of its children. To make such a bottom-up quantification meaningful from the attacker's perspective, we

need to require at least the (global) Under-Match property. Indeed, this property stipulates that all the paths satisfying a refinement of a node's goal also satisfy the goal itself. Under-Match corresponds thus to an under-approximation of the set of scenarios and it is enough to consider it for the purpose of finding a vulnerability in the system.

To make the analysis meaningful from the point of view of the defender, we will rather require the Over-Match property. This property means that all the paths satisfying the parent goal also satisfy its decomposition into sub-goals. Since the Over-Match property corresponds to an over-approximation of the set of scenarios, it is enough to consider it for the purpose of designing countermeasures.

Our method to evaluate the correctness of an attack tree is to check Admissibility and the (global) Meet property. If it holds, then we say that the attack tree construction is correct w.r.t. to the analyzed system. We then look at the stronger properties. Depending on the situation, the expert might want to ensure either the (global) Under-Match or the (global) Over-Match property. If the tree fails to verify the desired property with respect to a given system $\mathcal{S}$, then it needs to be reshaped before it can be employed for the security analysis of the real system modeled by $\mathcal{S}$.

5 Complexity issues

In this section, we address the complexity of deciding our four correctness properties introduced in Definition 5. For full proofs, we refer the reader to the extended version of this work [4]. Table 1 gives an overview of the obtained results. In the case of the `OR` and the `SAND` operators, all the correctness properties are decided in polynomial time, which is promising in practice. However, for the `AND` operator, checking the Admissibility property and the Meet property is NP-complete, and checking the Under-Match property is co-NP-complete. These last two problems are therefore intractable [9], but recall that their complexity in practice might be lower thanks to much favorable kinds of instances (see for example [18]).

	Admissibility	Meet	Under-Match	Over-Match	Match
`OR`	P	P	P	P	P
`SAND`	P	P	P	P	P
`AND`	NP-c	NP-c	co-NP-c	co-NP	co-NP

Table 1: Complexities of the correctness properties.

We first state two lemmas that will be useful for our complexity analysis. Lemma 2 provides a bound to the size of paths we need to consider in the system for the verification of correctness properties. Lemma 3 provides the complexity of checking if a path reflects a particular combination of subgoals.

Lemma 2. *Let $\mathcal{S}$ be a transition system, $\mathtt{OP} \in \{\mathtt{OR}, \mathtt{AND}, \mathtt{SAND}\}$, and $\iota_1, \gamma_1, \ldots \iota_n, \gamma_n \in$* Prop. *For every path π in $[\![\mathtt{OP}(\langle \iota_1, \gamma_1 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)]\!]^{\mathcal{S}}$, there exists a path π' of linear size*

in $|\mathcal{S}|$ and n that is also in $[\![\mathtt{OP}(\langle\iota_1,\gamma_1\rangle,\ldots,\langle\iota_n,\gamma_n\rangle)]\!]^{\mathcal{S}}$ and which preserves the ends of π, i.e., $\pi'(0)=\pi(0)$ and $\pi'(|\pi'|)=\pi(|\pi|)$. More precisely, $|\pi'|\in O((2n-1)\,|S|)$.

Lemma 3. *Let $\mathcal{S}$ be a transition system, $\iota_1,\gamma_1,\ldots\iota_n,\gamma_n$ be propositions in* Prop*, and let $\pi\in\Pi(\mathcal{S})$. Determining whether $\pi\in[\![\mathtt{OP}(\langle\iota_1,\gamma_1\rangle,\langle\iota_2,\gamma_2\rangle,\ldots,\langle\iota_n,\gamma_n\rangle)]\!]^{\mathcal{S}}$ can be done in time $O(|\pi|+n)$, if $\mathtt{OP}=\mathtt{SAND}$, and in time $O(|\pi|\,n)$, if $\mathtt{OP}=\mathtt{AND}$.*

The proofs of the two lemmas are provided in [4].

5.1 Checking Admissibility (column 1 of Table 1)

We now investigate the complexity of deciding the admissibility of an attack tree.

Proposition 1. *Given a system $\mathcal{S}$ and $\iota_1,\gamma_1,\ldots\iota_n,\gamma_n\in$* Prop*, deciding $[\![\langle\iota,\gamma\rangle]\!]^{\mathcal{S}}\neq\emptyset$, deciding $[\![\mathtt{OR}(\langle\iota_1,\gamma_1\rangle,\ldots,\langle\iota_n,\gamma_n\rangle)]\!]^{\mathcal{S}}\neq\emptyset$, and deciding $[\![\mathtt{SAND}(\langle\iota_1,\gamma_1\rangle,\ldots,\langle\iota_n,\gamma_n\rangle)]\!]^{\mathcal{S}}\neq\emptyset$ are decision problems in* P.

Proof.

1. Determining if $[\![\langle\iota,\gamma\rangle]\!]^{\mathcal{S}}$ is not empty amounts to performing a standard reachability analysis in $\mathcal{S}$, which can be done in polynomial time.
2. By the path semantics of the OR operator, $[\![\mathtt{OR}(\langle\iota_1,\gamma_1\rangle,\ldots,\langle\iota_n,\gamma_n\rangle)]\!]^{\mathcal{S}}\neq\emptyset$ if and only if there is $i\in[1,n]$, such that $[\![\langle\iota_j,\gamma_j\rangle]\!]^{\mathcal{S}}\neq\emptyset$, which by the case 1 of this proof, yields a polynomial time algorithm.
3. Checking that $[\![\mathtt{SAND}(\langle\iota_1,\gamma_1\rangle,\ldots,\langle\iota_n,\gamma_n\rangle)]\!]^{\mathcal{S}}\neq\emptyset$ can be done by a forward analysis: for $1\leq i\leq n$, we define a sequence of state sets S_i by induction over i as follows: we let $S_1=\lambda(\iota_1)$. Next, for $2\leq i<n$, $S_{i+1}=\lambda(\iota_{i+1})\cap\lambda(\gamma_i)\cap Post^*_{\mathcal{S}}(S_i)$. Clearly, $[\![\mathtt{SAND}(\langle\iota_1,\gamma_1\rangle,\ldots,\langle\iota_n,\gamma_n\rangle)]\!]^{\mathcal{S}}\neq\emptyset$ if, and only if $S_n\neq\emptyset$. Moreover, computing S_n takes at most $n\,|S|$ steps, since each S_{i+1} is computed from S_i in at most $|S|$ steps.

In the case of the AND operator the reasoning is more complex.

Proposition 2. *Given a system $\mathcal{S}$ and $\iota_1,\gamma_1,\ldots\iota_n,\gamma_n\in$* Prop*, deciding the non-emptiness $[\![\mathtt{AND}(\langle\iota_1,\gamma_1\rangle,\ldots,\langle\iota_n,\gamma_n\rangle)]\!]^{\mathcal{S}}\neq\emptyset$ is* NP*-complete.*

Proof. NP**-easy**: We can use the algorithm of Lemma 3, with the algorithm guessing a path of polynomial size according to Lemma 2. NP**-hard**: We recall that a *set of clauses* $\mathscr{C}$ over a set of (propositional) variables $\{p_1,\ldots,p_r\}$ is composed of elements (the clauses) $C\in\mathscr{C}$ such that C is a set of literals, that is either a variable p_i or its negation $\neg p_i$. The set $\mathscr{C}$ is *satisfiable* if there exists a valuation of the variables $p_1,\ldots,p_r$ that renders all the clauses of $\mathscr{C}$ true. The SAT problem is: given a set of clauses $\mathscr{C}$, to decide if it is satisfiable. It is well-known that SAT is an NP-complete problem [6].

Now, let $\mathscr{C}=\{C_1,\ldots,C_m\}$ be a set of clauses over variables $\{p_1,\ldots,p_r\}$ (ordered by their index) that is an input of the SAT problem. Classically, we let $|\mathscr{C}|$ be the sum of the sizes of all the clauses in $\mathscr{C}$, where the size of a clause is the number of its literals.

In the following, we let the symbol ℓ_i denote either p_i or $\neg p_i$, for every $i\in\{1,\ldots,r\}$. We define the labeled transition system $\mathcal{S}_{\mathscr{C}}=(S_{\mathscr{C}},\rightarrow_{\mathscr{C}},\lambda_{\mathscr{C}})$ over the set of propositions $\{start,C_1,\ldots,C_m\}$, where *start* is a fresh proposition, as follows. The set of states

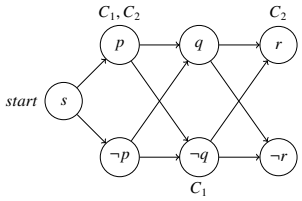


Fig. 5: The system $\mathcal{S}_{\{C_1,C_2\}}$ where $C_1 = p \vee \neg q$ and $C_2 = p \vee r$.

is $S_{\mathscr{C}} = \bigcup_{i=1}^{r} \{p_i, \neg p_i\} \cup \{s\}$, where s is a fresh state; the transition relation is $\rightarrow_{\mathscr{C}} = \{(\ell_i, \ell_{i+1}) \mid i \in [1, r-1]\} \cup \{(s, \ell_1)\}$; and the labeling of states $\lambda_{\mathscr{C}} : \{start, C_1, \ldots, C_m\} \rightarrow 2^S$ is such that $\lambda_{\mathscr{C}}(start) = \{s\}$ and $\lambda_{\mathscr{C}}(C_i) = \{\ell \in C_i\}$ for $1 \leq i \leq m$. Note that, by definition, $|\mathcal{S}_{\mathscr{C}}|$ is polynomial in $|\mathscr{C}|$. For example, the transition system corresponding to the set formed by clauses $C_1 = p \vee \neg q$ and $C_2 = p \vee r$ is depicted in Fig. 5.

It is then easy to establish that $[\![\texttt{AND}(\langle start, C_1\rangle, \langle start, C_2\rangle, \ldots, \langle start, C_m\rangle)]\!]^{\mathcal{S}_{\mathscr{C}}} \neq \emptyset$ if, and only if $\mathscr{C}$ is satisfiable.

According to the formal definition of the statement "T is admissible w.r.t. $\mathcal{S}$" as defined in Sect. 4, it is easy to combine the results of Propositions 1 and 2, to conclude that verifying that a tree is admissible is an NP-complete problem.

5.2 Checking the Meet property (column 2 of Table 1)

Preliminaries on temporal logic. We consider a syntactic fragment of the temporal logic CTL [5] where the only temporal operator is "eventually", here denoted by symbol $\Diamond$, and where Boolean operators are conjunction and disjunction. The syntax of the formulas is $\varphi ::= p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Diamond\varphi$. The semantics of formulas is given with regard to a labeled transition system $\mathcal{S} = (S, \rightarrow, \lambda)$: each formula φ denotes a subset of states, which we note $[\varphi]_{\mathcal{S}}$, and which is defined by induction: $[p]_{\mathcal{S}} = \lambda(p)$, $[\varphi \wedge \varphi']_{\mathcal{S}} = [\varphi]_{\mathcal{S}} \cap [\varphi']_{\mathcal{S}}$, $[\varphi \vee \varphi']_{\mathcal{S}} = [\varphi]_{\mathcal{S}} \cup [\varphi']_{\mathcal{S}}$, and $[\Diamond\varphi]_{\mathcal{S}} = Pre^*_{\mathcal{S}}([\varphi]_{\mathcal{S}})$, where $Pre^*_{\mathcal{S}}$ is defined in Sect. 3.1. Recall that $s \in [\Diamond\varphi]_{\mathcal{S}}$ if, and only if, there is a path in $\mathcal{S}$ starting from s and that reaches a state in $[\varphi]_{\mathcal{S}}$. It is well-established that computing $[\varphi]_{\mathcal{S}}$ can be done in polynomial time in $|\mathcal{S}|$ and $|\varphi|$ (see for example [27]).

We now turn to the complexity of verifying the Meet property.

Proposition 3. *Given a system $\mathcal{S}$ and $\iota, \gamma, \iota_1, \gamma_1, \ldots \iota_n, \gamma_n \in$ Prop, the problem of deciding $[\![\texttt{OR}(\langle \iota_1, \gamma_1\rangle, \ldots, \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}} \cap [\![\langle \iota, \gamma\rangle]\!]^{\mathcal{S}} \neq \emptyset$, and the problem of deciding $[\![\texttt{SAND}(\langle \iota_1, \gamma_1\rangle, \ldots, \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}} \cap [\![\langle \iota, \gamma\rangle]\!]^{\mathcal{S}} \neq \emptyset$ are in* P.

Proof.

1. Let $\varphi_{\texttt{OR}} := \bigvee_{i=1}^{n} \iota \wedge \iota_i \wedge \Diamond(\gamma \wedge \gamma_i)$. We claim that $[\![\texttt{OR}(\langle \iota_1, \gamma_1\rangle, \ldots \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}} \cap [\![\langle \iota, \gamma\rangle]\!]^{\mathcal{S}} \neq \emptyset$ iff $[\varphi_{\texttt{OR}}]_{\mathcal{S}} \neq \emptyset$. We easily conclude our proof from the claim and the fact that computing $[\varphi_{\texttt{OR}}]_{\mathcal{S}}$ can be done in polynomial time.

2. Let $\varphi_{\texttt{SAND}} := \iota \wedge \iota_1 \wedge \Diamond(\gamma_1 \wedge \iota_2 \wedge \Diamond(\gamma_2 \wedge \ldots \Diamond(\gamma_n \wedge \gamma)))$. We claim that $[\![\texttt{SAND}(\langle \iota_1, \gamma_1\rangle, \ldots \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}} \cap [\![\langle \iota, \gamma\rangle]\!]^{\mathcal{S}} \neq \emptyset$ iff $[\varphi_{\texttt{SAND}}]_{\mathcal{S}} \neq \emptyset$. We easily conclude our proof from the claim and the fact that computing $[\varphi_{\texttt{SAND}}]_{\mathcal{S}}$ can be done in polynomial time.

The proofs of the two claims can be found in the extended version [4].

Again, the AND operator turns out to be intrinsically more complex to deal with.

Proposition 4. *Given a system $\mathcal{S}$ and $\iota, \gamma, \iota_1, \gamma_1, \ldots \iota_n, \gamma_n \in$ Prop, deciding $[\![\mathit{AND}(\langle \iota_1, \gamma_1\rangle, \ldots, \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}} \cap [\![\langle \iota, \gamma\rangle]\!]^{\mathcal{S}} \neq \emptyset$ is an* NP*-complete problem.*

Proof. NP**-easy:** We can construct a non-deterministic polynomial time algorithm that guesses a path $\pi \in \Pi(\mathcal{S})$, of polynomial size in $|\mathcal{S}|$ and n (this is justified by Lemma 2), and checks that $\pi \in [\![\texttt{AND}(\langle \iota_1, \gamma_1\rangle, \ldots, \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}}$, which can be done in polynomial time in the size of π, which is also in polynomial time in $|\mathcal{S}|$ and n by the choice of π (see Lemma 3). NP**-hard:** we reduce the problem of deciding $[\![\texttt{AND}(\langle \iota_1, \gamma_1\rangle, \ldots, \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}} \neq \emptyset$ which is NP-hard by Proposition 2. The details are given in the extended version [4].

As a consequence of Propositions 3 and 4, it is NP-complete to verify that an attack tree has the Meet property, but if we restrict to attack trees that contain only OR or SAND operators, the problem becomes P.

5.3 Checking the Under-Match property (column 3 of Table 1)

The OR and SAND operators do not pose any problem. Due to the lack of space, we omit the proof which can be found in the extended version [4].

Proposition 5. *Given a system $\mathcal{S}$ and $\iota, \gamma, \iota_1, \gamma_1, \ldots \iota_n, \gamma_n \in$ Prop, deciding $[\![\mathit{OR}(\langle \iota_1, \gamma_1\rangle, \ldots, \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}} \subseteq [\![\langle \iota, \gamma\rangle]\!]^{\mathcal{S}}$, and deciding $[\![\mathit{SAND}(\langle \iota_1, \gamma_1\rangle, \ldots, \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}} \subseteq [\![\langle \iota, \gamma\rangle]\!]^{\mathcal{S}}$ are decision problems in* P.

As previously, the AND operator yields a more complex problem to solve.

Proposition 6. *Given a system $\mathcal{S}$ and $\iota, \gamma, \iota_1, \gamma_1, \ldots \iota_n, \gamma_n \in$ Prop, deciding $[\![\mathit{AND}(\langle \iota_1, \gamma_1\rangle, \ldots, \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}} \subseteq [\![\langle \iota, \gamma\rangle]\!]^{\mathcal{S}}$ is a* co-NP*-complete problem.*

This proof is given in the extended version [4].

5.4 Checking the Over-Match property (column 4 of Table 1)

Again, the cases for the OR and AND operators are smooth whereas the case of the AND operator is more difficult. Full proofs of these results are long and can be found in [4].

Proposition 7. *Given a system $\mathcal{S}$ and $\iota, \gamma, \iota_1, \gamma_1, \ldots \iota_n, \gamma_n \in$ Prop, deciding $[\![\mathit{OR}(\langle \iota_1, \gamma_1\rangle, \ldots, \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}} \supseteq [\![\langle \iota, \gamma\rangle]\!]^{\mathcal{S}}$ and deciding $[\![\mathit{SAND}(\langle \iota_1, \gamma_1\rangle, \ldots, \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}} \supseteq [\![\langle \iota, \gamma\rangle]\!]^{\mathcal{S}}$ are decision problems in* P*. On the contrary deciding $[\![\mathit{AND}(\langle \iota_1, \gamma_1\rangle, \ldots, \langle \iota_n, \gamma_n\rangle)]\!]^{\mathcal{S}} \supseteq [\![\langle \iota, \gamma\rangle]\!]^{\mathcal{S}}$ is a decision problem in* co-NP.

Finally, we can get an upper bound for column 5 of Table 1 (the Match property) by taking the maximum between upper bound complexities for Under-Match and Over-Match, which achieves the filling of Table 1.

6 Conclusion and future work

In this work, we have developed and studied a formal setting to assist experts in the design of attack trees when a particular system is considered. The system is described by a finite state-transition system that reflects its dynamics and whose finite paths (sequences of states) denote attack scenarios. The attack tree nodes are labeled with pairs $\langle \iota, \gamma \rangle$ expressing the attacker's goals in terms of pre and postconditions. The semantics of attack trees is based on sets of finite paths in the transition system. Such sets of paths can be characterized as a mere reachability condition of the form "all paths from condition ι to condition γ", or by a combination of those by means of `OR`, `AND`, and `SAND`.

We have exhibited the Admissibility property which allows us to check whether it makes sense to analyze a given attack tree in the context of a considered system. We then propose four natural correctness properties on top of Admissibility, namely

- Meet – the node's refinement makes sense in a given system;
- Under (resp. Over) Match – the node's refinement under-approximates (resp. over-approximates) the goal of the node in a given system; and
- Match – the node's refinement expresses exactly the node's goal in a given system.

While analyzing an attack tree with respect to a system, we propose to start by checking whether each of its subtrees satisfies the Meet property – this is the minimum that we require from a correct attack tree. If this is the case, we can then check how well the tree refines the main attacker's goal, using (Under- and Over-) Matching. Our study reveals that the highest complexity in such analysis is due to conjunctive refinements (*i.e.,* the `AND` operator), as opposed to disjunctive and sequential refinements, cf. Table 1. The reason is that the semantics that we use in our framework relies on paths in a transition system and thus modeling and verification for paths' concatenation (used to formalize the `SAND` refinements) is much simpler than those for parallel decomposition (used to formalize the `AND` refinements). Indeed, the latter requires to analyze the combinatorics of paths representing children of a conjunctively refined node.

The framework presented in this paper offers numerous possibilities for practical applications in industrial setting. First, it can be used to estimate the quality of a refinement of an attack goal, that an expert could borrow from an attack pattern library. The correctness properties introduced in this work allow us to evaluate the relevance of often generic refinements in the context of a given system. Second, classical attack trees use text-based nodes that represent a desired configuration to be reached (our postcondition γ) without specifying the initial configuration (our precondition ι) where the attack will start from. Given a transition system $\mathcal{S}$ describing a real system to be analyzed, the text-based goals can be straightforwardly translated into formal propositions expressing the final configurations (*i.e.,* γ) to be reached by the attacker. The expert may also specify the initial configurations (*i.e.,* ι), but if he does not do so, they can be automatically generated from the transition system, by simply taking all states belonging to the set $Pre^*_{\mathcal{S}}(\lambda(\gamma))$ of predecessors of $\lambda(\gamma)$ in $\mathcal{S}$.

For pedagogical reasons, we have focused on simple atomic goals (*i.e.,* node labels) that are definable in terms of a precondition and a postcondition. As one of the future directions, we would like to enrich the language of atomic goals, for instance by adding

variables with history or invariants. Variables with history can be used to express properties such as *"Once detected, the attacker will always stay detected"*. With invariants, we may add constraints to the goals, as in *"Reach Room2 undetected without ever crossing Room1"*. If invariants are added to atomic goals, for instance using LTL formulas, the complexity of some problems presented in this paper may increase. In that case, checking that a path satisfies the semantics of a node might no longer be done in constant time, but in polynomial time, or even in PSPACE-complete, if arbitrary LTL formulas are allowed [7]. It would then be relevant to study the interplay between the expressiveness of the atomic goals and the complexity of verifying these correctness properties.

It would also be interesting to extend our framework to capture more complex properties than those defined in Definition 5. Pragmatic examples of such properties would be validities and tests expressed in an adequate logic. *Validities* would be formulas that are true in any system. An example of a validity would look like $\texttt{AND}(\langle\iota,\gamma\rangle,\langle\iota',\gamma'\rangle) \sqsupseteq \texttt{SAND}(\langle\iota,\gamma\rangle,\langle\iota',\gamma'\rangle)$, with the meaning that a sequential composition is a particular case of parallel composition. *Tests* would be formulas which might be true in some systems, but not necessarily in all cases. For instance, a formula like $\texttt{AND}(\langle\iota,\gamma\rangle,\langle\iota',\gamma'\rangle) \sqsubseteq \texttt{SAND}(\langle\iota,\gamma\rangle,\langle\iota',\gamma'\rangle)$ would mean that, in a given system, it is impossible to realize both $\langle\iota,\gamma\rangle$ and $\langle\iota',\gamma'\rangle$ otherwise than sequentially in this particular order.

Finally, we are currently working on integrating the framework developed in this work to the ATSyRA tool. The ultimate goal is to design software for generation of attack trees satisfying the correctness properties that we have introduced. The short-term objective is to validate the practicality of the proposed framework and its usability with respect to the complexity results that we have proven in this work.

References

1. Aslanyan, Z., Nielson, F.: Pareto efficient solutions of attack-defence trees. In: POST. LNCS, vol. 9036, pp. 95–114. Springer (2015)
2. Aslanyan, Z., Nielson, F.: Model checking exact cost for attack scenarios. In: International Conference on Principles of Security and Trust. Springer (2017)
3. Audinot, M., Pinchinat, S.: On the Soundness of Attack Trees. In: Graphical Models for Security. LNCS, vol. 9987, pp. 25–38. Springer (2016)
4. Audinot, M., Pinchinat, S., Kordy, B.: Is my attack tree correct? (extended version). CoRR abs/1706.08507 (2017), `http://arxiv.org/abs/1706.08507`
5. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Workshop on Logic of Programs. pp. 52–71. Springer (1981)
6. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on Theory of computing. pp. 151–158. ACM (1971)
7. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence. pp. 854–860. Association for Computing Machinery (2013)
8. Gadyatskaya, O., Hansen, R.R., Larsen, K.G., Legay, A., Olesen, M.C., Poulsen, D.B.: Modelling attack–defense trees using timed automata. In: FORMATS. LNCS, vol. 9884, pp. 35–50. Springer (2016)
9. Garey, M.R., Johnson, D.S.: Computers and intractability, vol. 29. W. H. Freeman and Company (2002)

10. Horne, R., Mauw, S., Tiu, A.: Semantics for specialising attack trees based on linear logic. Fundam. Inform. 153(1-2), 57–86 (2017)
11. Ivanova, M.G., Probst, C.W., Hansen, R.R., Kammüller, F.: Transforming Graphical System Models to Graphical Attack Models. In: Graphical Models for Security. LNCS, vol. 9390, pp. 82–96. Springer (2015)
12. Jhawar, R., Kordy, B., Mauw, S., Radomirović, S., Trujillo-Rasua, R.: Attack Trees with Sequential Conjunction. In: SEC. IFIP AICT, vol. 455, pp. 339–353. Springer (2015)
13. Jürgenson, A., Willemson, J.: Serial Model for Attack Tree Computations. In: ICISC. LNCS, vol. 5984, pp. 118–128. Springer (2009)
14. Kordy, B., Mauw, S., Radomirovic, S., Schweitzer, P.: Attack–defense trees. J. Log. Comput. 24(1), 55–87 (2014)
15. Kordy, B., Pièvre-Cambacédès, L., Schweitzer, P.: Dag-based attack and defense modeling: Don't miss the forest for the attack trees. Computer Science Review 13-14, 1–38 (2014)
16. Kordy, B., Pouly, M., Schweitzer, P.: Probabilistic reasoning with graphical security models. Inf. Sci. 342, 111–131 (2016)
17. Kumar, R., Ruijters, E., Stoelinga, M.: Quantitative attack tree analysis via priced timed automata. In: FORMATS. LNCS, vol. 9268, pp. 156–171. Springer (2015)
18. Leyton-Brown, K., Hoos, H.H., Hutter, F., Xu, L.: Understanding the empirical hardness of NP-complete problems. Communications of the ACM 57(5), 98–107 (2014)
19. Mauw, S., Oostdijk, M.: Foundations of Attack Trees. In: ICISC. LNCS, vol. 3935, pp. 186–198. Springer (2005)
20. OWASP: CISO AppSec Guide: Criteria for managing application security risks (2013)
21. Phillips, C.A., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: Workshop on New Security Paradigms. pp. 71–79. ACM (1998)
22. Pieters, W., Padget, J., Dechesne, F., Dignum, V., Aldewereld, H.: Effectiveness of qualitative and quantitative security obligations. J. Inf. Sec. Appl. 22, 3–16 (2015)
23. Pinchinat, S., Acher, M., Vojtisek, D.: ATSyRa: An Integrated Environment for Synthesizing Attack Trees – (Tool Paper). In: Graphical Models for Security. LNCS, vol. 9390, pp. 97–101. Springer (2015)
24. Research, N., (RTO), T.O.: Improving Common Security Risk Analysis. Tech. Rep. AC/323(ISP-049)TP/193, North Atlantic Treaty Organisation, University of California, Berkeley (2008)
25. Roy, A., Kim, D.S., Trivedi, K.S.: Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. Security and Communication Networks 5(8), 929–943 (2012)
26. Schneier, B.: Attack Trees: Modeling Security Threats. Dr. Dobb's Journal of Software Tools 24(12), 21–29 (1999)
27. Schnoebelen, P.: The complexity of temporal logic model checking. Advances in modal logic 4(393-436), 35 (2002)
28. Sheyner, O., Haines, J.W., Jha, S., Lippmann, R., Wing, J.M.: Automated Generation and Analysis of Attack Graphs. In: IEEE S&P. pp. 273–284. IEEE Computer Society (2002)
29. Thierry-Mieg, Y.: Symbolic model-checking using its-tools. In: TACAS. LNCS, vol. 9035, pp. 231–237. Springer (2015)
30. Vigo, R., Nielson, F., Nielson, H.R.: Automated Generation of Attack Trees. In: CSF. pp. 337–350. IEEE Computer Society (2014)