# On quantitative analysis of attack–defense trees with repeated labels

Barbara Kordy and Wojciech Wideł

Univ Rennes, INSA Rennes, CNRS, IRISA, Rennes, France
{barbara.kordy, wojciech.widel}@irisa.fr

**Abstract.** Ensuring security of complex systems is a difficult task that requires utilization of numerous tools originating from various domains. Among those tools we find *attack–defense trees*, a simple yet practical model for analysis of scenarios involving two competing parties. Enhancing the well-established model of attack trees, attack–defense trees are trees with labeled nodes, offering an intuitive representation of possible ways in which an attacker can harm a system, and means of countering the attacks that are available to the defender. The growing palette of methods for quantitative analysis of attack–defense trees provides security experts with tools for determining the most threatening attacks and the best ways of securing the system against those attacks. Unfortunately, many of those methods might fail or provide the user with distorted results if the underlying attack–defense tree contains multiple nodes bearing the same label. We address this issue by studying conditions ensuring that the standard bottom-up evaluation method for quantifying attack–defense trees yields meaningful results in the presence of repeated labels. For the case when those conditions are not satisfied, we devise an alternative approach for quantification of attacks.

## 1 Introduction

Beginning with 19th century chemistry and a groundbreaking work of Cayley, who used them for the purposes of enumeration of isomers, trees – connected acyclic graphs – have a long history of application to various domains. Those include safety analysis of systems using the model of *fault trees* [9], developed in 1960s, and security analysis with the assistance of the *attack trees*, which the fault trees inspired. Attack trees were introduced by Schneier in [26], for the purpose of analyzing security of systems and organizations. Seemingly simple, attack trees offer a compromise between expressiveness and usability, which not only makes them applicable for industrial purposes [22], but also puts them at the core of many more complex models and languages [10,23]. An extensive overview and comparison of attack tree-based graphical models for security can be found in [19]. A survey focusing on scalability, complexity analysis and practical usability of such models has recently been provided in [11].

Attack–defense trees [17] are one of the most well-studied extensions of attack trees, with new methods of their analysis developed yearly [1,2,7,20]. Attack–defense trees enhance attack trees with nodes labeled with goals of a defender,

thus enabling modeling of interactions between the two competing actors. They have been used to evaluate the security of real-life systems, such as ATMs [6], RFID managed warehouses [3] and cyber-physical systems [15]. Both the theoretical developments and the practical studies have proven that attack–defense trees offer a promising methodology for security evaluation, but they also highlighted room for improvements. The objective of the current paper is to address the problem of quantitative analysis of attack–defense trees with repeated labels.

*Related work.* It is well-know that the analysis of an attack–defense tree becomes more difficult if the tree contains repeated labels. This difficulty is sometimes recognized, e.g., in [1,20], where authors explicitly assume lack of repeated labels in order for their methods to be valid. In some works the problem is avoided (or overlooked) by interpretation of repeated labels as distinct instances of the same goal, thus, *de facto* as distinct goals (e.g., [7,12,17,21]), or by distinguishing between the repetitions occurring in specific subtrees of a tree, as in [2]. Recently, Bossuat and Kordy have established a classification of repeated labels in attack–defense trees, depending on whether the corresponding nodes represent exactly the same instance or different instances of a goal [4]. They point out that, if the meaning of repeated labels is not properly specified, then the fast, bottom-up method for identifying attacks that optimize an attribute (e.g., *minimal cost*, *probability of success*, etc.), as used in [14,17,21], might yield tainted results.

Repeated labels are also problematic in other tree-based models, for instance fault trees. Whereas some methods for qualitative analysis of fault trees with repeated basic events (or generally, shared subtrees) have been developed [27,5], their quantification might rely on approximate methods. For example, the probability of a system failure can be evaluated using rare event approximation approach (see [9], Chapter XI), while a simple bottom-up procedure gives an exact result in fault trees with no shared subtrees [25]. This last observation is consistent with the results previously obtained for attack–defense trees (see Theorems 2, 3, 4 in [1]).

*Contribution.* The contribution of this work is threefold. First, we determine sufficient conditions ensuring that the standard quantitative bottom-up analysis of attack–defense trees with repeated labels is valid. Second, we prove that some of these conditions are in fact necessary for the analysis to be compatible with a selected semantics for attack–defense trees. Finally, for the case when these conditions are not satisfied, we propose a novel, alternative method of evaluation of attributes that takes the presence of repeated labels into account.

*Paper structure.* The model of attack–defense trees is introduced in detail in the next section. In Section 3, the attributes and exisitng methods for their evaluation are explained. In Section 4, we present our main results on quantification of attack–defense trees with repeated labels. We give proofs of these results in Section 5, and conclude in Section 6.

## 2 Attack–defense trees

Attack–defense trees are rooted trees with labeled nodes that allow for an intuitive graphical representation of scenarios involving two competing actors, usually called *attacker* and *defender*. Nodes of a tree are labeled with goals of the actors, with the label of the root of the tree being the main goal of the modeled scenario. The actor whose goal is represented by the root is called *proponent* and the other one is called *opponent*. The aim of the proponent is to achieve the root goal, whereas the opponent tries to make this impossible.

In order for an actor to *achieve* some particular goal `g`, they might need to achieve other goals. In such a case the node labeled with `g` is a *refined node*. The basic model of attack–defense trees (as introduced in [17]) admits two types of refinements: the goal of a *conjunctively refined node* (an `AND` node) is achieved if the goals of all its child nodes are achieved, and the goal of a *disjunctively refined node* (an `OR` node) is achieved if at least one of the goals of its children is achieved. If a node is not refined, then it represents a goal that is considered to be directly achievable, for instance by executing a simple action. Such a goal is called a *basic action*. Hence, in order to achieve goals of refined nodes, the actors execute (some of) their basic actions. What distinguishes attack–defense trees model from attack trees is the possibility of the goals of the actors to be *countered* by goals of their adversary, which themselves can be again countered, and so on. To represent the countering of a goal, the symbol `C` will be used. A goal `g` is *countered* by a goal $\mathtt{g}'$ (denoted $\mathtt{C}(\mathtt{g}, \mathtt{g}')$) if achieving $\mathtt{g}'$ by one of the actors makes achieving `g` impossible for the other actor.

It is not rare that in an attack–defense tree, whether generated by hand or in a semi–automatic way [28,24,13] some nodes bear the same label. In such a case, there are two ways of interpreting them:

1. either the nodes represent the same single instance of the goal – e.g., cutting the power off in a building can be done once and has multiple consequences, thus a number of refined nodes might have a node labeled `cutPowerOff` among their child nodes, but all these nodes will represent exactly the same action of cutting the power off;
2. or else each of the nodes is treated as a distinct instance of the goal. For instance, while performing an attack, the attacker might need to pass through a door twice – once to enter and second time to leave a building. Since these actions refer to the same door and the same attacker, the corresponding nodes will, in most cases, hold the same label `goThroughDoor`. However, it is clear that they represent two different instances of the same goal.

In this work we assume the first of these ways of interpretation. In particular, following [4], we call a basic action that serves as a label for at least two nodes a *clone* or a *cloned basic action*, and interpret them as the same instance of a goal. Nodes representing distinct instances of the same goal or distinct goals are assumed in this work to have different labels.

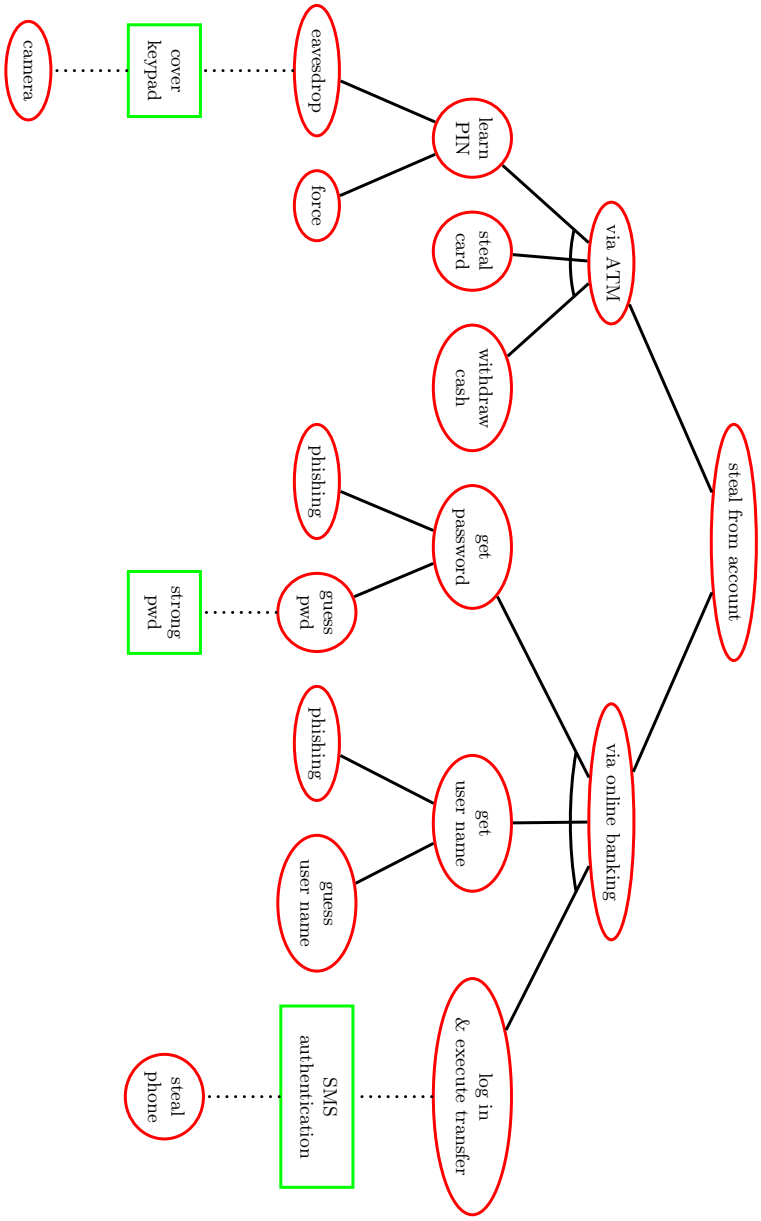


Fig. 1: Attack–defense tree for stealing money from a bank account

An example of attack–defense tree[1] is represented in Fig. 1. In this tree, the proponent is the attacker and the opponent is the defender. According to the attack–defense trees' convention, nodes representing goals of the attacker are depicted using red circles, and those of the defender using green rectangles. Children of an `AND` node are joined by an arc, and countermeasures are attached to nodes they are supposed to counter via dotted edges.

*Example 1.* In the attack–defense scenario represented by the attack–defense tree from Fig. 1, the proponent wants to steal money from the opponent's account. To achieve this goal, they can use physical means, i.e., force the opponent to reveal their PIN, steal the opponent's card and then withdraw money from an ATM. One way of learning the PIN would be to eavesdrop on the victim when they enter the PIN. This could be prevented by covering the keypad with hand. Covering the keypad fails if the proponent monitors the keypad with a hidden micro–camera installed at an appropriate spot. Another way of getting the PIN would be to force the opponent to reveal it.

Instead of attacking from a physical angle, the proponent can steal money by exploiting online banking services. In order to do so, they need to learn the opponent's user name and password. Both of these goals can be achieved by creating a fake bank website and using phishing techniques for tricking the opponent into entering their credentials. The proponent could also try to guess what the password and the user name are. Using very strong password would counter such guessing attack. Once the proponent obtains the credentials, they use them for logging into the online banking services and execute a transfer. Transfer dispositions might be additionally secured with two-factor authentication using mobile phone text messages. This security measure could be countered by the proponent by stealing the opponent's phone.

Note that even though there are two nodes labeled with `phishing` in the tree, they actually represent the same instance of the same action. The proponent does not need to perform two different phishing attacks to get the password and the user name — setting up one phishing website and sending one phishing e-mail will suffice for the proponent to get both credentials. Thus, the two nodes labeled `phishing` are clones.

Let us now introduce a formal notation for attack–defense trees, which we will use throughout this paper. Such notation is necessary to formally define the meaning of attack–defense trees in terms of formal semantics and to specify the algorithms for their quantitative analysis.

We use symbols p and o to distinguish between the proponent and the opponent. By $\mathbb{B}^{\mathrm{p}}$ and $\mathbb{B}^{\mathrm{o}}$ we denote the sets of labels representing basic actions of the proponent and of the opponent, respectively. We assume that $\mathbb{B}^{\mathrm{p}} \cap \mathbb{B}^{\mathrm{o}} = \emptyset$, and we set $\mathbb{B} = \mathbb{B}^{\mathrm{p}} \cup \mathbb{B}^{\mathrm{o}}$. For $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$, the symbol $\bar{\mathrm{s}}$ stands for the other actor, i.e., $\bar{\mathrm{p}} = \mathrm{o}$ and $\bar{\mathrm{o}} = \mathrm{p}$. We denote the elements of $\mathbb{B}^{\mathrm{s}}$ with $\mathtt{b}^{\mathrm{s}}$, for $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$. Attack–defense trees can be seen as terms generated by the following grammar,

[1] The example is based on one of exemplary trees provided by ADTool [8].

where $\mathtt{OR}^{\mathrm{s}}$ and $\mathtt{AND}^{\mathrm{s}}$ are unranked refinement operators, i.e., they may take an arbitrary number of arguments, and $\mathtt{C}^{\mathrm{s}}$ is a binary counter operator.

$$T^{\mathrm{s}} \colon \mathtt{b}^{\mathrm{s}} \;\mid\; \mathtt{OR}^{\mathrm{s}}(T^{\mathrm{s}}, \ldots, T^{\mathrm{s}}) \;\mid\; \mathtt{AND}^{\mathrm{s}}(T^{\mathrm{s}}, \ldots, T^{\mathrm{s}}) \;\mid \mathtt{C}^{\mathrm{s}}(T^{\mathrm{s}}, T^{\bar{\mathrm{s}}}) \tag{1}$$

*Example 2.* Consider the tree from Fig. 1. The term corresponding to the subtree rooted in the `via ATM` node is

$$\mathtt{AND}^{\mathrm{p}}\Big(\mathtt{OR}^{\mathrm{p}}\Big(\mathtt{C}^{\mathrm{p}}\big(\mathtt{eavesdrop}, \mathtt{C}^{\mathrm{o}}(\mathtt{coverKey}, \mathtt{camera})\big), \mathtt{force}\Big), \mathtt{stealCard}, \mathtt{withdrawCash}\Big),$$

where the labels of basic actions have been shortened for better readability.

We denote the set of trees generated by grammar (1) with $\mathbb{T}$.

In order to analyze possible attacks in an attack–defense tree, in particular, determine cheapest ones, or the ones that require the least amount of time to execute, one needs to decide what is considered to be an attack. This can be achieved with the help of semantics that provide formal interpretations for attack–defense trees. Several semantics for attack–defense trees have been proposed in [17]. Below, we recall two ways of interpreting attack–defense trees and the notions of attack they entail.

**Definition 1.** *The propositional semantics for attack–defense trees is a function $\mathcal{P}$ that assigns to each attack–defense tree a propositional formula, in a recursive way, as follows*

$$\mathcal{P}(\mathtt{b}) = x_{\mathtt{b}}, \qquad \mathcal{P}(\mathtt{OR}^{\mathrm{s}}(T_1^{\mathrm{s}}, \ldots, T_k^{\mathrm{s}})) = \mathcal{P}(T_1^{\mathrm{s}}) \vee \cdots \vee \mathcal{P}(T_k^{\mathrm{s}}),$$
$$\mathcal{P}(\mathtt{C}^{\mathrm{s}}(T_1^{\mathrm{s}}, T_2^{\bar{\mathrm{s}}})) = \mathcal{P}(T_1^{\mathrm{s}}) \wedge \neg\, \mathcal{P}(T_2^{\bar{\mathrm{s}}}), \quad \mathcal{P}(\mathtt{AND}^{\mathrm{s}}(T_1^{\mathrm{s}}, \ldots, T_k^{\mathrm{s}})) = \mathcal{P}(T_1^{\mathrm{s}}) \wedge \cdots \wedge \mathcal{P}(T_k^{\mathrm{s}}),$$

*where $\mathtt{b} \in \mathbb{B}$, and $x_{\mathtt{b}}$ is the corresponding propositional variable. Two attack–defense trees are equivalent wrt $\mathcal{P}$ if their interpretations are equivalent propositional formulæ.*

Definition 1 formalizes one of the most intuitive and widely used ways of interpreting attack–defense trees, where every basic action is assigned a propositional variable indicating whether or not the action is satisfiable. In the light of the propositional semantics, an attack in an attack–defense tree $T$ is any assignment of values to the propositional variables, such that the formula $\mathcal{P}(T)$ evaluates to true. We note that this natural approach is often used without invoking the propositional semantics explicitly (e.g., in [1] or [7]). Observe also that due to the idempotency of the logical operators $\vee$ and $\wedge$, and the fact that every basic action is assigned a single variable, when the propositional semantics is used, cloned actions are indeed treated as the same instance of the same action. In particular, this implies that the trees $\mathtt{AND}^{\mathrm{p}}(\mathtt{b}, \mathtt{OR}^{\mathrm{p}}(\mathtt{b}, \mathtt{b}'))$ and $\mathtt{b}$ are equivalent under the propositional interpretation. Such approach might not always be desirable, especially when we do not only want to know *whether* attacks

are possible, but actually *how* they can be achieved. To accommodate this point of view, the set semantics has recently been introduced in [4]. We briefly recall its construction below.

In the sequel, we set

$$S \odot Z = \{(P_S \cup P_Z, O_S \cup O_Z) | (P_S, O_S) \in S,\ (P_Z, O_Z) \in Z\}, \tag{2}$$

for $S, Z \subseteq\ \mathbb{B}^{\mathrm{p}} \times \mathbb{B}^{\mathrm{o}}$. Furthermore, for a set $X$ we denote its power set with $\wp(X)$.

**Definition 2.** *The set semantics for attack–defense trees is a function* $\mathcal{S} \colon \mathbb{T} \to \wp\big(\wp(\mathbb{B}^{\mathrm{p}}) \times \wp(\mathbb{B}^{\mathrm{o}})\big)$ *that assigns to each attack–defense tree a set of pairs of sets of labels, as follows*

$$\begin{aligned}
&\mathcal{S}\left(\mathtt{b}^{\mathrm{p}}\right) = \left\{\left(\{\mathtt{b}^{\mathrm{p}}\}, \emptyset\right)\right\}, && \mathcal{S}\left(\mathtt{b}^{\mathrm{o}}\right) = \left\{\left(\emptyset, \{\mathtt{b}^{\mathrm{o}}\}\right)\right\},\\
&\mathcal{S}\left(\mathtt{OR}^{\mathrm{p}}(T_1^{\mathrm{p}}, \ldots, T_k^{\mathrm{p}})\right) = \bigcup_{i=1}^{k} \mathcal{S}(T_i^{\mathrm{p}}), && \mathcal{S}\left(\mathtt{OR}^{\mathrm{o}}(T_1^{\mathrm{o}}, \ldots, T_k^{\mathrm{o}})\right) = \bigodot_{i=1}^{k} \mathcal{S}(T_i^{\mathrm{o}}),\\
&\mathcal{S}\left(\mathtt{AND}^{\mathrm{p}}(T_1^{\mathrm{p}}, \ldots, T_k^{\mathrm{p}})\right) = \bigodot_{i=1}^{k} \mathcal{S}(T_i^{\mathrm{p}}), && \mathcal{S}\left(\mathtt{AND}^{\mathrm{o}}(T_1^{\mathrm{o}}, \ldots, T_k^{\mathrm{o}})\right) = \bigcup_{i=1}^{k} \mathcal{S}(T_i^{\mathrm{o}}),\\
&\mathcal{S}\left(\mathtt{C}^{\mathrm{p}}(T_1^{\mathrm{p}}, T_2^{\mathrm{o}})\right) = \mathcal{S}(T_1^{\mathrm{p}}) \odot \mathcal{S}(T_2^{\mathrm{o}}), && \mathcal{S}\left(\mathtt{C}^{\mathrm{o}}(T_1^{\mathrm{o}}, T_2^{\mathrm{p}})\right) = \mathcal{S}(T_1^{\mathrm{o}}) \cup \mathcal{S}(T_2^{\mathrm{p}}).
\end{aligned}$$

*Two trees* $T_1$ *and* $T_2$ *are* equivalent wrt the set semantics*, denoted* $T_1 \equiv_{\mathcal{S}} T_2$*, if and only if the two sets* $\mathcal{S}(T_1)$ *and* $\mathcal{S}(T_2)$ *are equal.*

The meaning of a pair $(P, O)$ belonging to $\mathcal{S}(T)$ is that if the proponent executes all actions from $P$ and the opponent does not execute any of the actions from $O$, then the root goal of the tree $T$ is achieved. In particular, if $(P, \emptyset) \in \mathcal{S}(T)$, then the opponent cannot prevent the proponent from achieving the root goal when they execute all actions from $P$.

*Example 3.* The set semantics of the tree in Fig. 1 is the following

$$\begin{aligned}
\mathcal{S}(T) = \big\{&(\{\mathtt{force}, \mathtt{stealCard}, \mathtt{withdrawCash}\}, \emptyset),\\
&(\{\mathtt{camera}, \mathtt{eavesdrop}, \mathtt{stealCard}, \mathtt{withdrawCash}\}, \emptyset),\\
&(\{\mathtt{eavesdrop}, \mathtt{stealCard}, \mathtt{withdrawCash}\}, \{\mathtt{coverKey}\}),\\
&(\{\mathtt{phish}, \mathtt{logIn\&execTrans}\}, \{\mathtt{SMS}\}),\\
&(\{\mathtt{phish}, \mathtt{guessUN}, \mathtt{logIn\&execTrans}\}, \{\mathtt{SMS}\}),\\
&(\{\mathtt{phish}, \mathtt{guessPwd}, \mathtt{logIn\&execTrans}\}, \{\mathtt{strongPWD}, \mathtt{SMS}\}),\\
&(\{\mathtt{guessUN}, \mathtt{guessPwd}, \mathtt{logIn\&execTrans}\}, \{\mathtt{strongPWD}, \mathtt{SMS}\}),\\
&(\{\mathtt{phish}, \mathtt{stealPhone}, \mathtt{logIn\&execTrans}\}, \emptyset),\\
&(\{\mathtt{phish}, \mathtt{guessUN}, \mathtt{stealPhone}, \mathtt{logIn\&execTrans}\}, \emptyset),\\
&(\{\mathtt{phish}, \mathtt{guessPwd}, \mathtt{stealPhone}, \mathtt{logIn\&execTrans}\}, \{\mathtt{strongPWD}\}),\\
&(\{\mathtt{guessUN}, \mathtt{guessPwd}, \mathtt{stealPhone}, \mathtt{logIn\&execTrans}\}, \{\mathtt{strongPWD}\})\big\}.
\end{aligned}$$

Throughout the rest of the paper, by an *attack in an attack–defense tree* $T$ we mean an element of its set semantics $\mathcal{S}(T)$.

Grammar (1) ensures that attack–defense trees are well-typed with respect to the two players, i.e., p and o. However, not every well-typed tree is necessarily well-formed wrt the labels used. In particular, it should be ensured that the usage of repeated labels is consistent throughout the whole tree. For instance, if the action `coverKey`, of covering an ATM's keypad with a hand, can be countered by monitoring with a camera, this countermeasure should also be attached to every other node labeled `coverKey`. Similarly, if execution of the action `logIn&execTrans` contributes to the achievement of the proponent's goal of stealing money via the online banking services, this information should be kept in every subtree rooted in a node labeled `via online banking`. Thus, to ensure that the results of the methods developed further in the paper indeed reflect the intended aspects of a modeled scenario, in the following we assume that subtrees of an attack–defense tree that are rooted in identically labeled nodes are equivalent wrt the set semantics.

## 3 Quantitative analysis using attributes

Among methods for quantitative analysis of scenarios modeled with attack–defense trees are so called *attributes*, introduced intuitively by Schneier in [26] and formalized for attack trees in [21] and [14], and for attack–defense trees in [17]. Attributes represent quantitative aspects of the modeled scenario, such as a *minimal cost* of executing an attack or *maximal damage* caused by an attack. Numerous methods to evaluate the value of an attribute on attack–defense trees exist [1,7], and the most often used approach is based on so called bottom-up evaluation [17]. The idea behind the bottom-up evaluation is to assign attribute values to the basic actions and to propagate them up to the root of the tree using appropriate operations on the intermediate nodes. The notions of attribute and bottom-up evaluation are formalized using attribute domains.

**Definition 3.** *An attribute domain for an attribute $\alpha$ on attack–defense trees is a tuple*

$$A_\alpha = (D_\alpha, \mathtt{OR}^{\mathrm{p}}_\alpha, \mathtt{AND}^{\mathrm{p}}_\alpha, \mathtt{OR}^{\mathrm{o}}_\alpha, \mathtt{AND}^{\mathrm{o}}_\alpha, \mathtt{C}^{\mathrm{p}}_\alpha, \mathtt{C}^{\mathrm{o}}_\alpha),$$

*where $D_\alpha$ is a set, and for* $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$, $\mathtt{OP} \in \{\mathtt{OR}, \mathtt{AND}\}$,

1. $\mathtt{OP}^{\mathrm{s}}_\alpha$ *is an unranked function on $D_\alpha$,*
2. $\mathtt{C}^{\mathrm{s}}_\alpha$ *is a binary function on $D_\alpha$.*

Let $A_\alpha = (D_\alpha, \mathtt{OR}^{\mathrm{p}}_\alpha, \mathtt{AND}^{\mathrm{p}}_\alpha, \mathtt{OR}^{\mathrm{o}}_\alpha, \mathtt{AND}^{\mathrm{o}}_\alpha, \mathtt{C}^{\mathrm{p}}_\alpha, \mathtt{C}^{\mathrm{o}}_\alpha)$ be an attribute domain. A function $\beta_\alpha : \mathbb{B} \to D_\alpha$ that assigns values from the set $D_\alpha$ to basic actions of attack–defense trees is called a *basic assignment* for attribute $\alpha$.

**Definition 4.** *Let $A_\alpha = (D_\alpha, \mathtt{OR}^{\mathrm{p}}_\alpha, \mathtt{AND}^{\mathrm{p}}_\alpha, \mathtt{OR}^{\mathrm{o}}_\alpha, \mathtt{AND}^{\mathrm{o}}_\alpha, \mathtt{C}^{\mathrm{p}}_\alpha, \mathtt{C}^{\mathrm{o}}_\alpha)$ be an attribute domain, $T$ be an attack–defense tree, and $\beta_\alpha$ be a basic assignment for attribute*

$\alpha$. *The value of attribute* $\alpha$ *for* $T$ *obtained via the bottom–up procedure, denoted* $\alpha_B(T, \beta_\alpha)$*, is defined recursively as*

$$\alpha_B(T,\beta_\alpha) = \begin{cases} \beta_\alpha(\mathtt{b}) & \text{if } T = \mathtt{b}, \mathtt{b} \in \mathbb{B}, \\ \mathtt{OP}^{\mathrm{s}}_\alpha(\alpha_B(T_1^{\mathrm{s}},\beta_\alpha),\ldots,\alpha_B(T_n^{\mathrm{s}},\beta_\alpha)) & \text{if } T = \mathtt{OP}^{\mathrm{s}}(T_1^{\mathrm{s}},\ldots,T_n^{\mathrm{s}}), \\ \mathtt{C}^{\mathrm{s}}_\alpha(\alpha_B(T_1^{\mathrm{s}},\beta_\alpha),\alpha_B(T_2^{\bar{\mathrm{s}}},\beta_\alpha)) & \text{if } T = \mathtt{C}^{\mathrm{s}}(T_1^{\mathrm{s}},T_2^{\bar{\mathrm{s}}}), \end{cases}$$

*where* $\mathrm{s} \in \{\mathrm{p},\mathrm{o}\}$, $\mathtt{OP} \in \{\mathtt{OR},\mathtt{AND}\}$. *(In the notation* $\alpha_B(T,\beta_\alpha)$*, the index* $B$ *refers to the "bottom-up" computation.)*

An extensive overview of attribute domains and their classification can be found in [18]. The article [3] contains a case study and guidelines for practical application of the bottom-up procedure. Numerous examples of attributes for attack trees and attack trees extended with additional sequential refinement have been given in [12] and [14]. We gather some the relevant attribute domains for attack–defense trees in Table 1.

Example 4 illustrates the bottom-up procedure on the tree from Fig. 1.

*Example 4.* Consider the tree $T$ given in Fig. 1, and let $\alpha$ be the *minimal attack cost* attribute (see Table 1 for its attribute domain). We fix the basic assignement $\beta_{\mathrm{cost}}$ to be as follows:

| basic action b | $\beta_{\mathrm{cost}}(\mathtt{b})$ | basic action b | $\beta_{\mathrm{cost}}(\mathtt{b})$ |
|---|---|---|---|
| stealCard | 60 | force | 100 |
| camera | 75 | withdrawCash | 10 |
| phish | 70 | eavesdrop | 20 |
| guessPwd | 120 | guessUN | 120 |
| logIn&execTrans | 10 | stealPhone | 60 |

Furthermore, for all the basic actions of the opponent, we set $\beta_{\mathrm{cost}}(\mathtt{b}) = +\infty$. The bottom-up computation of the *minimal cost* on $T$ gives

$$\mathrm{cost}_B(T,\beta_{\mathrm{cost}}) = 165.$$

This value corresponds to monitoring with the camera, eavesdropping on the victim to learn their PIN, stealing the card, and withdrawing money.

As already noticed in [21], the value of an attribute for a tree can also be evaluated directly on its semantic. For our purposes we define this evaluation as follows.

**Definition 5.** *Let* $(D_\alpha, \mathtt{OR}^{\mathrm{p}}_\alpha, \mathtt{AND}^{\mathrm{p}}_\alpha, \mathtt{OR}^{\mathrm{o}}_\alpha, \mathtt{AND}^{\mathrm{o}}_\alpha, \mathtt{C}^{\mathrm{p}}_\alpha, \mathtt{C}^{\mathrm{o}}_\alpha)$ *be an attribute domain and let* $T$ *be an attack–defense tree with a basic assignment* $\beta_\alpha$*. The* value of the attribute $\alpha$ for $T$ evaluated on the set semantics*, denoted* $\alpha_{\mathcal{S}}(T,\beta_\alpha)$*, is defined as*

$$\alpha_{\mathcal{S}}(T,\beta_\alpha) = (\mathtt{OR}^{\mathrm{p}}_\alpha)_{(P,O)\in\mathcal{S}(T)}\left(\mathtt{C}^{\mathrm{p}}_\alpha\big((\mathtt{AND}^{\mathrm{p}}_\alpha)_{\mathtt{b}\in P}\beta_\alpha(\mathtt{b}),(\mathtt{OR}^{\mathrm{o}}_\alpha)_{\mathtt{b}\in O}\beta_\alpha(\mathtt{b})\big)\right).$$

| **Attribute** | $D_\alpha$ | $\mathtt{OR}^{\mathrm{p}}_\alpha$ | $\mathtt{AND}^{\mathrm{p}}_\alpha$ | $\mathtt{OR}^{\mathrm{o}}_\alpha$ | $\mathtt{AND}^{\mathrm{o}}_\alpha$ | $\mathtt{C}^{\mathrm{p}}_\alpha$ | $\mathtt{C}^{\mathrm{o}}_\alpha$ | $\beta_\alpha(\mathtt{b}^{\mathrm{o}})$ |
|---|---|---|---|---|---|---|---|---|
| min. attack cost | $\mathbb{R}_{\geq 0} \cup \{+\infty\}$ | min | $+$ | $+$ | min | $+$ | min | $+\infty$ |
| max. damage | $\mathbb{R}_{\geq 0} \cup \{-\infty\}$ | max | $+$ | $+$ | max | $+$ | max | $-\infty$ |
| min. skill level | $\mathbb{N} \cup \{0, +\infty\}$ | min | max | max | min | max | min | $+\infty$ |
| min. nb of experts | $\mathbb{N} \cup \{0, +\infty\}$ | min | $+$ | $+$ | min | $+$ | min | $+\infty$ |
| satisfiability for p | $\{0, 1\}$ | $\vee$ | $\wedge$ | $\wedge$ | $\vee$ | $\wedge$ | $\vee$ | 0 |

Table 1: Selected attribute domains for attack–defense trees

*(In the notation $\alpha_{\mathcal{S}}(T, \beta_\alpha)$, the index $\mathcal{S}$ refers to the computation on the "set semantics".)*

*Example 5.* Consider again the tree from Fig. 1 and the basic assignment for the *minimal cost* attribute given in Example 4. The cost of all elements of the set semantics for $T$ are as follows

$$
\begin{aligned}
&(\{\mathtt{force}, \mathtt{stealCard}, \mathtt{withdrawCash}\}, \emptyset), && 170\\
&(\{\mathtt{camera}, \mathtt{eavesdrop}, \mathtt{stealCard}, \mathtt{withdrawCash}\}, \emptyset), && 165\\
&(\{\mathtt{eavesdrop}, \mathtt{stealCard}, \mathtt{withdrawCash}\}, \{\mathtt{coverKey}\}) && +\infty\\
&(\{\mathtt{phish}, \mathtt{logIn\&execTrans}\}, \{\mathtt{SMS}\}), && +\infty\\
&(\{\mathtt{phish}, \mathtt{guessUN}, \mathtt{logIn\&execTrans}\}, \{\mathtt{SMS}\}), && +\infty\\
&(\{\mathtt{phish}, \mathtt{guessPwd}, \mathtt{logIn\&execTrans}\}, \{\mathtt{strongPWD}, \mathtt{SMS}\}), && +\infty\\
&(\{\mathtt{guessUN}, \mathtt{guessPwd}, \mathtt{logIn\&execTrans}\}, \{\mathtt{strongPWD}, \mathtt{SMS}\}), && +\infty\\
&(\{\mathtt{phish}, \mathtt{stealPhone}, \mathtt{logIn\&execTrans}\}, \emptyset), && 140\\
&(\{\mathtt{phish}, \mathtt{guessUN}, \mathtt{stealPhone}, \mathtt{logIn\&execTrans}\}, \emptyset), && 260\\
&(\{\mathtt{phish}, \mathtt{guessPwd}, \mathtt{stealPhone}, \mathtt{logIn\&execTrans}\}, \{\mathtt{strongPWD}\}), && +\infty\\
&(\{\mathtt{guessUN}, \mathtt{guessPwd}, \mathtt{stealPhone}, \mathtt{logIn\&execTrans}\}, \{\mathtt{strongPWD}\}) && +\infty.
\end{aligned}
$$

The evaluation of the *minimal cost* attribute on the set semantics for $T$ gives

$$\alpha_{\mathcal{S}}(T, \beta_{\mathrm{cost}}) = \min\{170, 165, +\infty, 140, 260\} = 140,$$

which corresponds to performing the phishing attack to get the user name and their password, stealing the phone, and logging into the online bank application to execute the transfer.

Notice that the values obtained for the same tree in Examples 4 and 5 are different, despite the fact that the same basic assignment and the same attribute domain have been used. This is due to the fact that the tree from Fig. 1 contains cloned nodes which the standard bottom-up evaluation cannot handle properly. In the next section, we provide conditions and develop a method for a proper evaluation of attributes on attack–defense trees with cloned nodes.

## 4 Quantification on attack–defense trees with clones

Depending on what is considered to be an attack in an attack–defense tree, different semantics can be used. Note that a semantics for attack–defense trees naturally introduces an equivalence relation in $\mathbb{T}$. It is thus of great importance to select a method of quantitative analysis that is consistent with a chosen semantics, i.e., a method that for any two trees equivalent wrt the employed semantics returns the same result. This issue was recognized by the authors of [21] for attack trees, and addressed, in the case of attack–defense trees in [17], with the notion of compatibility between an attribute domain and a semantics. Below, we adapt the definition of compatibility from [17] to the bottom-up computation.

**Definition 6.** *Let $A_\alpha = (D_\alpha, \mathtt{OR}^{\mathrm{p}}_\alpha, \mathtt{AND}^{\mathrm{p}}_\alpha, \mathtt{OR}^{\mathrm{o}}_\alpha, \mathtt{AND}^{\mathrm{o}}_\alpha, \mathtt{C}^{\mathrm{p}}_\alpha, \mathtt{C}^{\mathrm{o}}_\alpha)$ be an attribute domain. The bottom-up procedure, defined in Definition 4, is compatible with a semantics $\equiv$ for attack–defense trees, if for every two trees $T_1$, $T_2$ satisfying $T_1 \equiv T_2$, the equality $\alpha_B(T_1, \beta_\alpha) = \alpha_B(T_2, \beta_\alpha)$ holds for any basic assignment $\beta_\alpha$.*

For instance, it is well-known that the bottom-up computation of the *minimal cost* using the domain from Table 1 is not compatible with the propositional semantics. Indeed, consider the trees $T_1 = \mathtt{OR}^{\mathrm{p}}(\mathtt{b}, \mathtt{AND}(\mathtt{b}', \mathtt{b}''))$ and $T_2 = \mathtt{AND}^{\mathrm{p}}(\mathtt{OR}^{\mathrm{p}}(\mathtt{b}, \mathtt{b}'), (\mathtt{b}, \mathtt{b}''))$ whose corresponding propositional formulæ are equivalent. However, for the basic assignment $\beta_{\mathrm{cost}}(\mathtt{b}) = 3$, $\beta_{\mathrm{cost}}(\mathtt{b}') = 4$, $\beta_{\mathrm{cost}}(\mathtt{b}'') = 1$ the values $\alpha_B(T_1, \beta_\alpha) = 3$ and $\alpha_B(T_2, \beta_\alpha) = 4$ are different. Similarly, the bottom-up computation of the *minimal cost* attribute is not compatible with the set semantics. This can be shown by considering trees $T_3 = \mathtt{AND}^{\mathrm{p}}(\mathtt{b}, \mathtt{b})$ and $T_4 = \mathtt{b}$ and will further be discussed in Corollary 1.

This notion of compatibility defined in Definition 6 can be generalized to any computation on attack–defense trees.

**Definition 7.** *Let $\mathcal{D}$ be a set and let $f$ be a function on $\mathbb{T} \times \mathcal{D}$. We say that $f$ is compatible with a semantics $\equiv$ for attack–defense trees, if for every two trees $T_1$, $T_2$ satisfying $T_1 \equiv T_2$ the equality $f(T_1, d) = f(T_2, d)$ holds for any $d \in \mathcal{D}$.*

To illustrate the difference between the compatibility notions defined in Definitions 6 and 7, one can consider the method for computing the so called *attacker's expected outcome*, proposed by Jürgenson and Willemson in [16]. Since this method is not based on an attribute domain, it cannot be simulated using the bottom-up evaluation. However, the authors show that the outcome of their computations is independent from the Boolean representation of an attack tree. This means that the method proposed in [16] is compatible with the propositional semantics for attack trees.

*Remark 1.* Consider an attribute domain $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ with $\oplus$ and $\otimes$ being binary, associative, and commutative operations on $D_\alpha$[2]. Under

[2] Note that a binary and associative operation can be modeled with an unranked operator.

these assumptions, for a tree $T$ and a basic assignment $\beta_\alpha$, we have

$$\alpha_{\mathcal{S}}(T,\beta_\alpha) = \bigoplus_{(P,O)\in\mathcal{S}(T)} \left( \bigotimes \left( \bigotimes_{\mathtt{b}\in P} \beta_\alpha(\mathtt{b}), \bigotimes_{\mathtt{b}\in O} \beta_\alpha(\mathtt{b}) \right) \right) = \\ = \bigoplus_{(P,O)\in\mathcal{S}(T)} \bigotimes_{\mathtt{b}\in P\cup O} \beta_\alpha(\mathtt{b}).$$

Since for any two trees $T_1$ and $T_2$ that are equivalent wrt the set semantics the expressions $\alpha_{\mathcal{S}}(T_1,\beta_\alpha)$ and $\alpha_{\mathcal{S}}(T_2,\beta_\alpha)$ differ only in the order of the terms, they yield the same (numerical) result. In other words, under the above assumptions, the computation $\alpha_{\mathcal{S}}$ is compatible with the set semantics.

As it has been observed in [17] and [18], there is a wide class of attribute domains of the form $(D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$, where $(D_\alpha, \oplus, \otimes)$ constitutes a commutative idempotent semiring. Recall that an algebraic structure $(R, \oplus, \otimes)$ is a commutative idempotent semiring if $\oplus$ is an idempotent operation, both operations $\oplus$ and $\otimes$ are associative and commutative, their neutral elements, denoted here by $\mathtt{e}_\oplus$ and $\mathtt{e}_\otimes$, belong to $R$, operation $\otimes$ distributes over $\oplus$, and the absorbing element of $\otimes$, denoted $\mathtt{a}_\otimes$, is equal to $\mathtt{e}_\oplus$.

*Remark 2.* In order for the computations performed using the bottom-up evaluation to be consistent with the intuition, the basic actions of the opponent are assigned a specific value. In the case of an attribute domain $(D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ based on a commutative idempotent semiring $(D_\alpha, \oplus, \otimes)$ this value is equal to $\mathtt{a}_\otimes$. One of the consequences of this choice is that if for every attack $(P, O) \in \mathcal{S}(T)$ the set $O$ is not empty, then $\alpha_{\mathcal{S}}(T, \beta_\alpha) = \mathtt{a}_\otimes = \mathtt{e}_\oplus$, indicating the fact that the proponent cannot achieve the root goal if the opponent executes all of their actions present in the tree. Note that this is closely related to the choice of the functions $\mathtt{C}^{\mathrm{p}}_\alpha = \otimes$ and $\mathtt{C}^{\mathrm{o}}_\alpha = \oplus$.

*Example 6.* For instance, in the case of the *minimal cost* attribute domain (cf. Table 1), which is based on the idempotent commutative semiring $(\mathbb{R}_{\geq 0} \cup \{+\infty\}, \min, +)$, the basic actions of the opponent are originally assigned $+\infty$, which is both a neutral element for the min operation, and the absorbing element for the addition. This implies that, if on a certain path, there is an opponent's action which is not countered by the proponent, the corresponding branch will result in the value $+\infty$, which models that it is impossible (since too costly) for the proponent. This is due to the fact that $\mathtt{C}^{\mathrm{p}}_{\mathrm{cost}} = +$. However, if the opponent's action is countered by the proponent's action, the corresponding branch will yield a real value different from $+\infty$, because the min operator, used for $\mathtt{C}^{\mathrm{o}}_{\mathrm{cost}}$, will be applied between a real number assigned to the proponent's counter and the $+\infty$.

The first contribution of this work is presented in Theorem 1. It establishes a relation between the evaluation of attributes via the bottom–up procedure and their evaluation on the set semantics. Its proof is postponed to Section 5.

**Theorem 1.** *Let $T$ be an attack–defense tree generated by grammar* (1) *and let $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ be an attribute domain such that the operations $\oplus$ and $\otimes$ are associative and commutative, $\oplus$ is idempotent, and $\otimes$ distributes over $\oplus$. If*

- *there are no repeated labels in $T$, or*
- *the operator $\otimes$ is idempotent,*

*then the equality $\alpha_B(T, \beta_\alpha) = \alpha_{\mathcal{S}}(T, \beta_\alpha)$ holds for any basic assignment $\beta_\alpha$.*

Note that the assumptions of Theorem 1 are satisfied by any commutative idempotent semiring, thus the same result also holds for attributes whose attribute domains are based on commutative idempotent semirings. Furthermore, one can compare the assumption on the lack of repeated labels in Theorem 1 with the linearity of an attack–defense tree, considered in [1]. The authors of [1] have proven that under this strong assumption, the evaluation method that they have developed for multi-parameter attributes coincides with their bottom-up evaluation.

*Remark 3.* Consider again the attribute domain specified in Theorem 1. Suppose that the operation $\otimes$ is not idempotent. Then there exists $d \in D_\alpha$, such that $d \otimes d \neq d$. In consequence, for $\beta_\alpha(\mathtt{b}) = d$ and the trees $T_1 = \mathtt{b}$ and $T_2 = \mathtt{AND}^{\mathrm{p}}(\mathtt{b}, \mathtt{b})$ that are equivalent wrt to the set semantics, we have $\alpha_B(T_1, \beta_\alpha) \neq \alpha_B(T_2, \beta_\alpha)$. This shows that, if the operation $\otimes$ is not idempotent, the bottom-up evaluation based on the attribute domain satisfying the remaining assumptions of Theorem 1 is not compatible with the set semantics.

Theorem 1 and Remarks 1 and 3 immediately yield the following corollary.

**Corollary 1.** *Let $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ be an attribute domain such that the operations $\oplus$ and $\otimes$ are associative and commutative, $\oplus$ is idempotent, and $\otimes$ distributes over $\oplus$. The bottom-up procedure based on $A_\alpha$ is compatible with the set semantics if and only if the operation $\otimes$ is idempotent.*

We can also notice that if the assumptions from Corollary 1 are satisfied but the operation $\otimes$ is not idempotent, then the bottom-up procedure is compatible with the so called multiset semantics (introduced for attack trees in [21] and attack–defense trees in [17]) which uses pairs of multisets instead of pairs of sets.

Some of the domains based on idempotent semirings have a specific property that we encapsulate in the notion of *non-increasing* domain.

**Definition 8.** *Let $A_\alpha$ be an attribute domain. We say that $A_\alpha$ is non-increasing if $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$, $(D_\alpha, \oplus, \otimes)$ is a commutative idempotent semiring, and for every $d, c \in D_\alpha$, the inequality $d \otimes c \preceq d$ holds, where $\preceq$ stands for the canonical partial order on $D_\alpha$, i.e., the order defined by $d \preceq c$ if and only if $d \oplus c = c$.*

*Example 7.* From the attribute domains presented in Table 1 all but one are non-increasing. The only one which is not non-increasing is the *maximal damage* domain.

Note that in order to be able to evaluate the value of an attribute on the set semantics $\mathcal{S}(T)$, one needs to construct the semantics itself. This task might be computationally expensive, since, in the worst case, the number of elements of $\mathcal{S}(T)$ is exponential in the number of nodes of $T$. In contrast, the complexity of the bottom-up procedure is linear in the number of nodes of the underlying tree (if the operations performed on the intermediate nodes are linear in the number of arguments). Thus, it is desirable to ensure that $\alpha_B(T, \beta_\alpha) = \alpha_{\mathcal{S}}(T, \beta_\alpha)$. By Theorem 1, this equality holds in a wide class of attributes, provided that there are no clones in $T$. If $T$ contains clones, then the two methods might return different values (as illustrated in Remark 3).

To deal with this issue, we present our second contribution of this work. In Algorithm 1, we propose a method of evaluating the value of attributes having non-increasing domains on attack–defense trees, that takes the repetition of labels into account. The algorithm relies on the following notion of *necessary clones*.

**Definition 9.** *Let* $\mathtt{b}$ *be a cloned basic action of the proponent in an attack–defense tree* $T$*. If* $\mathtt{b}$ *is present in every attack of the form* $(P, \emptyset) \in \mathcal{S}(T)$*, then* $\mathtt{b}$ *is a necessary clone; otherwise it is an optional clone.*

It is easy to see that the tree from Fig. 1 does not contain any necessary clones. Indeed, this tree contains only one clone – `phish` – however, there exists the attack $(\{\mathtt{force}, \mathtt{stealCard}, \mathtt{withdrawCash}\}, \emptyset)$ which does not make use of the corresponding phishing action.

The sets of all necessary and optional clones in a tree $T$ are denoted with $\mathcal{C}_N(T)$ and $\mathcal{C}_O(T)$, respectively. When there is no danger of ambiguity, we use $\mathcal{C}_N$ and $\mathcal{C}_O$ instead of $\mathcal{C}_N(T)$ and $\mathcal{C}_O(T)$. The idea behind Algorithm 1 is to first recognize the set $\mathcal{C}_N$ of necessary clones and temporarily ensure that the values of the attribute assigned to them do not influence the result of the bottom–up procedure. Then the values of the optional clones are also temporarily modified, and the corresponding bottom-up evaluations are performed. Only then the result is adjusted in such a way that the original values of the necessary clones are taken into account. Before explaining Algorithm 1 in detail, we provide a simple method for determining whether a cloned basic action of the proponent is a necessary clone in the following lemma.

**Lemma 1.** *Let* $T$ *be an attack–defense tree generated by grammar* (1) *and* $\mathtt{a} \in \mathbb{B}^{\mathrm{P}}$ *be a cloned action of the proponent in* $T$*. Let* $\alpha$ *be the minimal skill level attribute (cf. Table 1) with the following basic assignment, for* $\mathtt{b} \in \mathbb{B}$

$$\beta_{\mathrm{skill}}(\mathtt{b}) = \begin{cases} 0 \text{ if } \mathtt{b} \neq \mathtt{a} \text{ and } \mathtt{b} \in \mathbb{B}^{\mathrm{P}}, \\ 1 \text{ if } \mathtt{b} = \mathtt{a}, \\ +\infty \text{ otherwise.} \end{cases}$$

*Then,* $\mathtt{a}$ *is a necessary clone in* $T$ *if and only if* $\mathrm{skill}_B(T, \beta_{\mathrm{skill}}) = 1$.

*Proof.* Observe that under the given basic assignment the value of $\mathrm{skill}_{\mathcal{S}}(T, \beta_{\mathrm{skill}})$ is equal to 1 if and only if $\mathtt{a}$ is a necessary clone. Since max is an idempotent operation, $\mathrm{skill}_B(T, \beta_{\mathrm{skill}}) = \mathrm{skill}_{\mathcal{S}}(T, \beta_{\mathrm{skill}})$, by Theorem 1. The lemma follows. □

We now explain our algorithm for evaluating attributes on attack–defense trees with repeated labels. Algorithm 1 takes as input an attack–defense tree $T$

**Algorithm 1** Evaluation of attributes in attack–defense tree with clones

**Input:** Attack–defense tree $T$, attribute domain $(D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$, $\beta_\alpha \colon \mathbb{B} \to D_\alpha$
**Output:** $\alpha_{\mathcal{A}}(T, \beta_\alpha)$
1: $\alpha_{\mathcal{A}}(T, \beta_\alpha) \leftarrow \mathtt{e}_\oplus$
2: initialize $\mathcal{C}_N$, $\mathcal{C}_O$
3: $\beta'_\alpha(\mathtt{b}) \leftarrow \mathtt{e}_\otimes$ for every $\mathtt{b} \in \mathcal{C}_N$
4: $\beta'_\alpha(\mathtt{b}) \leftarrow \beta_\alpha(\mathtt{b})$ for every $\mathtt{b} \in \mathbb{B} \setminus (\mathcal{C}_N \cup \mathcal{C}_O)$
5: **for** every subset $\mathcal{C} \subseteq \mathcal{C}_O$ **do**
6: $\quad \beta'_\alpha(\mathtt{b}) \leftarrow \mathtt{a}_\otimes$ for every $\mathtt{b} \in \mathcal{C}$
7: $\quad \beta'_\alpha(\mathtt{b}) \leftarrow \mathtt{e}_\otimes$ for every $\mathtt{b} \in \mathcal{C}_O \setminus \mathcal{C}$
8: $\quad r^c \leftarrow \alpha_B(T, \beta'_\alpha) \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_O \setminus \mathcal{C}} \beta_\alpha(\mathtt{b})$
9: $\quad \alpha_{\mathcal{A}}(T, \beta_\alpha) \leftarrow \alpha_{\mathcal{A}}(T, \beta_\alpha) \oplus r^c$
10: **end for**
11: $\alpha_{\mathcal{A}}(T, \beta_\alpha) \leftarrow \alpha_{\mathcal{A}}(T, \beta_\alpha) \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_N} \beta_\alpha(\mathtt{b})$
12: **return** $\alpha_{\mathcal{A}}(T, \beta_\alpha)$

generated by grammar (1), an attribute domain $A_\alpha$, and a basic assignment $\beta_\alpha$ for the attribute. Once the sets of necessary and the optional clones have been determined, new basic assignments are created. Under each of these assignments $\beta'_\alpha$, the necessary clones receive value $\mathtt{e}_\otimes$ (in line 3). Intuitively, this ensures two things. First, that when the bottom–up procedure with the assignment $\beta'_\alpha$ is performed (in line 8), the value selected at the nodes corresponding to a choice made by the proponent (e.g., at the $\mathtt{OR}^{\mathrm{p}}$ nodes) is likely to be the one corresponding to a subset of actions of some optimal attack (i.e., a subset containing a necessary clone). The second outcome is that in the final result of the algorithm, the values of $\beta_\alpha$ assigned to the necessary clones are taken into account exactly once (line 11).

In lines 6–7, an assignment $\beta'_\alpha$ is created for every subset $\mathcal{C}$ of the set of optional clones $\mathcal{C}_O$. The clones from $\mathcal{C}$ are assigned $\mathtt{a}_\otimes$, which intuitively ensures that they are ignored by the bottom-up procedure, and the remaining optional clones are assigned $\mathtt{e}_\otimes$ (again, to ensure that their values under $\beta_\alpha$ will eventually be counted exactly once). The result of computations performed in the **for** loop is multiplied (in the sense of performing operation $\otimes$) in line 11 by the product of values assigned to the necessary clones. (Note that the index $\mathcal{A}$ in the notation $\alpha_{\mathcal{A}}(T, \beta_\alpha)$ refers to the evaluation using Algorithm 1.)

*Example 8.* We illustrate Algorithm 1 on the tree $T$ from Fig. 1 and the *minimal cost* attribute domain. Consider the basic assignment of cost given in Example 4. Observe that $\mathcal{C}_N = \emptyset$ and $\mathcal{C}_O = \{\texttt{phish}\}$. The sets $\mathcal{C}$ considered in the **for** loop, their influence on the assignment of cost, and their corresponding results $r^c$ are the following

$$\begin{aligned} &\mathcal{C} = \emptyset, && \beta'_{\mathrm{cost}}(\texttt{phish}) = 0, && r^c = 140 \\ &\mathcal{C} = \{\texttt{phish}\}, && \beta'_{\mathrm{cost}}(\texttt{phish}) = +\infty, && r^c = 165. \end{aligned}$$

The value of $\mathrm{cost}_{\mathcal{A}}(T, \beta_{\mathrm{cost}})$ after the **for** loop is $\min\{140, 165\}$. Since $\mathcal{C}_N = \emptyset$, the algorithm returns $\mathrm{cost}_{\mathcal{A}}(T, \beta_{\mathrm{cost}}) = 140$. This value corresponds to the cost of the attack $(\{\texttt{phish}, \texttt{logIn\&execTrans}, \texttt{stealPhone}\}, \emptyset)$, which is indeed the cheapest attack in the tree under the given basic assignment, as already illustrated in Example 5. Notice furthermore, that $\mathrm{cost}_{\mathcal{A}}(T, \beta_{\mathrm{cost}}) = \alpha_{\mathcal{S}}(T, \beta_{\mathrm{cost}})$ (cf. Example 5).

Now we turn our attention to complexity of Algorithm 1. Let $k$ be the number of distinct clones of the proponent in $T$. Furthermore, let $n$ be the number of nodes in $T$. We assume that the complexity of operations $\oplus$ and $\otimes$ is linear in the number of arguments, which is a reasonable assumption in the view of the existing attribute domains (cf. Table 1). This implies that the result of a single bottom up-procedure in $T$ is obtained in time $\mathcal{O}(n)$. Thus, from the operations performed in lines 1–4, the most complex one is the initialization of the sets $\mathcal{C}_N$ and $\mathcal{C}_O$, the time complexity of which is in $\mathcal{O}(kn)$ (by Lemma 1). Since the **for** loop from line 5 iterates over all of the subsets of the optional clones, and the operations inside the loop are linear in $n$, the overall time complexity of Algorithm 1 is in $\mathcal{O}(n2^k)$.

In Theorem 2 we give sufficient conditions for the result $\alpha_{\mathcal{A}}(T, \beta_\alpha)$ of Algorithm 1 to be equal to the result $\alpha_{\mathcal{S}}(T, \beta_\alpha)$ of evaluation on the set semantics. Its proof is presented in Section 5.

**Theorem 2.** *Let $T$ be an attack–defense tree generated by grammar (1) and $A_\alpha$ be a non–increasing attribute domain. Then the equality $\alpha_{\mathcal{A}}(T, \beta_\alpha) = \alpha_{\mathcal{S}}(T, \beta_\alpha)$ holds for every basic assignment $\beta_\alpha \colon \mathbb{B} \to D_\alpha$ satisfying $\beta_\alpha|_{\mathbb{B}^\circ} \equiv \mathtt{a}_\otimes$.*

Remark 1 and Theorem 2 imply the following corollary.

**Corollary 2.** *Let $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ be a non–increasing attribute domain and let $\beta := \{\beta_\alpha \colon \mathbb{B} \to D_\alpha \ st \ \beta_\alpha|_{\mathbb{B}^\circ} \equiv \mathtt{a}_\otimes\}$. Then, the evaluation procedure $\alpha_A \colon \mathbb{T} \times \beta \to D_\alpha$ specified by Algorithm 1 is compatible with the set semantics (in the sense of Definition 7).*

## 5 Proofs of Theorems 1 and 2

Throughout this section it is assumed that $T$ is an attack–defense tree generated by grammar (1) and $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ is an attribute domain with

the operations $\oplus$ and $\otimes$ that are associative and commutative, $\oplus$ is idempotent, and $\otimes$ distributes over $\oplus$. We begin with examining parallels between attribute domains of this type and the set semantics.

Since the operation $\otimes$ distributes over $\oplus$, the result of the bottom–up procedure for any basic assignment $\beta_\alpha$ of $\alpha$ can be represented as

$$\begin{aligned}\alpha_B(T,\beta_\alpha) = {} & (\beta_\alpha(\mathtt{b}_1^1) \otimes \beta_\alpha(\mathtt{b}_2^1) \otimes \ldots \otimes \beta_\alpha(\mathtt{b}_{k_1}^1)) \oplus \\ & \ldots \\ & \oplus (\beta_\alpha(\mathtt{b}_1^i) \otimes \beta_\alpha(\mathtt{b}_2^i) \otimes \ldots \otimes \beta_\alpha(\mathtt{b}_{k_i}^i)) \oplus \\ & \ldots \\ & \oplus (\beta_\alpha(\mathtt{b}_1^n) \otimes \beta_\alpha(\mathtt{b}_2^n) \otimes \ldots \otimes \beta_\alpha(\mathtt{b}_{k_n}^n)).\end{aligned} \tag{3}$$

Observe that with the set $D_{\mathcal{S}} = \wp\big(\wp(\mathbb{B}^{\mathrm{p}}) \times \wp(\mathbb{B}^{\mathrm{o}})\big)$ and the operation $\odot$ defined by equality (2), the algebraic structure $(D_{\mathcal{S}}, \cup, \odot)$ constitutes a commutative idempotent semiring. Consider the attribute domain $A_{\mathcal{S}} = (D_{\mathcal{S}}, \cup, \odot, \odot, \cup, \odot, \cup)$ and the basic assignment

$$\beta_{\mathcal{S}}(\mathtt{b}) = \begin{cases} \big\{\big(\{\mathtt{b}\}, \emptyset\big)\big\} & \text{if } \mathtt{b} \in \mathbb{B}^{\mathrm{p}}, \\ \big\{\big(\emptyset, \{\mathtt{b}\}\big)\big\} & \text{otherwise.} \end{cases}$$

Clearly, $\mathcal{S}(T) = \mathcal{S}_B(T, \beta_{\mathcal{S}})$. By the previous observations $\mathcal{S}_B(T, \beta_{\mathcal{S}})$ can be represented as

$$\begin{aligned}\mathcal{S}_B(T,\beta_{\mathcal{S}}) = {} & (\beta_{\mathcal{S}}(\mathtt{b}_1^1) \odot \beta_{\mathcal{S}}(\mathtt{b}_2^1) \odot \cdots \odot \beta_{\mathcal{S}}(\mathtt{b}_{k_1}^1)) \cup \\ & \ldots \\ & \cup (\beta_{\mathcal{S}}(\mathtt{b}_1^i) \odot \beta_{\mathcal{S}}(\mathtt{b}_2^i) \odot \cdots \odot \beta_{\mathcal{S}}(\mathtt{b}_{k_i}^i)) \cup \\ & \ldots \\ & \cup (\beta_{\mathcal{S}}(\mathtt{b}_1^n) \odot \beta_{\mathcal{S}}(\mathtt{b}_2^n) \odot \cdots \odot \beta_{\mathcal{S}}(\mathtt{b}_{k_n}^n)).\end{aligned} \tag{4}$$

We chose the representations (3) and (4) in such a way that for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, k_i\}$ the basic action $\mathtt{b}_j^i$ in (3) is the same as $\mathtt{b}_j^i$ in (4), which is possible due to the commutativity of the operations.

From definitions of the basic assignment $\beta_{\mathcal{S}}$ and the operation $\odot$ it follows that for every $i \in \{1, \ldots, n\}$ the $i$th term

$$\beta_{\mathcal{S}}(\mathtt{b}_1^i) \odot \beta_{\mathcal{S}}(\mathtt{b}_2^i) \odot \cdots \odot \beta_{\mathcal{S}}(\mathtt{b}_{k_i}^i)$$

of representation (4) is a set consisting of exactly one pair of sets. Let us denote this term with $\{(P_i, O_i)\}$. Observe that since $\mathcal{S}(T) = \mathcal{S}_B(T, \beta_{\mathcal{S}})$, we have $(P_i, O_i) \in \mathcal{S}(T)$ for every $i$, and, conversely, for every $(P, O) \in \mathcal{S}(T)$ there exists at least one $i$ such that $(P, O) = (P_i, O_i)$.

Finally, we denote the $i$th term of representation (3) with $\alpha_i$. Now we are ready to prove Theorem 1.

*Proof of Theorem 1.* If there are no repeated labels in $T$ or the operator $\otimes$ is idempotent, then for $i \in \{1, \ldots, n\}$ it holds that $\alpha_i = \bigotimes_{\mathtt{b} \in P_i \cup O_i} \beta_\alpha(\mathtt{b})$. Together with the idempotency of $\oplus$ this implies that

$$\alpha_B(T, \beta_\alpha) = \bigoplus_{i=1}^{n} \alpha_i = \bigoplus_{(P,O) \in \mathcal{S}(T)} \bigotimes_{\mathtt{b} \in P \cup O} \beta_\alpha(\mathtt{b}).$$

□

We finish this section by providing the proof of Theorem 2.

*Proof of Theorem 2.* Consider a result $r^c$ of the bottom–up procedure obtained in the line 8 of Algorithm 1 for a set $\mathcal{C} \subseteq \mathcal{C}_O$ of optional clones. Using representation (3), it can be written as

$$\begin{aligned}
r^c = \alpha_B(T, \beta'_\alpha) \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_O \setminus \mathcal{C}} \beta_\alpha(\mathtt{b}) = \\
= \left(\beta'_\alpha(\mathtt{b}^1_1) \otimes \beta'_\alpha(\mathtt{b}^1_2) \otimes \ldots \otimes \beta'_\alpha(\mathtt{b}^1_{k_1})\right) \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_O \setminus \mathcal{C}} \beta_\alpha(\mathtt{b}) \oplus \\
\ldots \\
\oplus \left(\beta'_\alpha(\mathtt{b}^i_1) \otimes \beta'_\alpha(\mathtt{b}^i_2) \otimes \ldots \otimes \beta'_\alpha(\mathtt{b}^i_{k_i})\right) \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_O \setminus \mathcal{C}} \beta_\alpha(\mathtt{b}) \oplus \\
\ldots \\
\oplus \left(\beta'_\alpha(\mathtt{b}^n_1) \otimes \beta'_\alpha(\mathtt{b}^n_2) \otimes \ldots \otimes \beta'_\alpha(\mathtt{b}^n_{k_n})\right) \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_O \setminus \mathcal{C}} \beta_\alpha(\mathtt{b}).
\end{aligned}$$

Let us denote the $i$th term of the above expression with $r^c_i$. Observe that the result of Algorithm 1 is

$$\alpha_{\mathcal{A}}(T, \beta_\alpha) = \left[\bigoplus_{\mathcal{C} \subseteq \mathcal{C}_O} r^c\right] \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_N} \beta_\alpha(\mathtt{b}) = \left(\bigoplus_{i=1}^{n} \left[\bigoplus_{\mathcal{C} \subseteq \mathcal{C}_O} r^c_i\right]\right) \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_N} \beta_\alpha(\mathtt{b}).$$

Due to the values assigned to the optional clones in the **for** loop, the inner expression can be expanded as follows.

$$\bigoplus_{\mathcal{C}\subseteq\mathcal{C}_O} r_i^{\mathcal{C}} = \left[\bigoplus_{\substack{\mathcal{C}\subseteq\mathcal{C}_O\\ \mathcal{C}\cap(P_i\cup O_i)\neq\emptyset}} [\mathtt{a}_\otimes \otimes \bigotimes_{\mathtt{b}\in\mathcal{C}_O\setminus\mathcal{C}} \beta_\alpha(\mathtt{b})]\right] \oplus$$

$$\oplus \bigoplus_{\substack{\mathcal{C}\subseteq\mathcal{C}_O\\ \mathcal{C}\cap(P_i\cup O_i)=\emptyset}} \left[\bigotimes_{\substack{\mathtt{b}\in P_i\cup O_i\\ \mathtt{b}\notin\mathcal{C}_N\cup\mathcal{C}_O}} \beta_\alpha(\mathtt{b}) \otimes \bigotimes_{\substack{\mathtt{b}\in P_i\cup O_i\\ \mathtt{b}\in\mathcal{C}_N\cup\mathcal{C}_O\setminus\mathcal{C}}} \mathtt{e}_\otimes \otimes \bigotimes_{\mathtt{b}\in\mathcal{C}_O\setminus\mathcal{C}} \beta_\alpha(\mathtt{b})\right] =$$

$$= \bigoplus_{\substack{\mathcal{C}\subseteq\mathcal{C}_O\\ \mathcal{C}\cap(P_i\cup O_i)=\emptyset}} \left[\bigotimes_{\substack{\mathtt{b}\in P_i\cup O_i\\ \mathtt{b}\notin\mathcal{C}_N}} \beta_\alpha(\mathtt{b}) \otimes \bigotimes_{\substack{\mathtt{b}\notin P_i\cup O_i\\ \mathtt{b}\in\mathcal{C}_O\setminus\mathcal{C}}} \beta_\alpha(\mathtt{b})\right]$$

Since the attribute domain is non–increasing, the last "sum" is absorbed by the term corresponding to the set $\mathcal{C}$ satisfying $\mathcal{C}_O \setminus \mathcal{C} = (P_i \cup O_i) \cap \mathcal{C}_O$, namely, the term $\bigotimes_{\substack{\mathtt{b}\in P_i\cup O_i\\ \mathtt{b}\notin\mathcal{C}_N}} \beta_\alpha(\mathtt{b})$. Thus,

$$\alpha_{\mathcal{A}}(T,\beta_\alpha) = \left(\bigoplus_{i=1}^{n}\left[\bigotimes_{\substack{\mathtt{b}\in P_i\cup O_i\\ \mathtt{b}\notin\mathcal{C}_N}} \beta_\alpha(\mathtt{b})\right]\right) \otimes \bigotimes_{\mathtt{b}\in\mathcal{C}_N} \beta_\alpha(\mathtt{b}) =$$

$$= \bigoplus_{i=1}^{n} \bigotimes_{\mathtt{b}\in P_i\cup O_i} \beta_\alpha(\mathtt{b}) = \bigoplus_{(P,O)\in\mathcal{S}(T)} \bigotimes_{\mathtt{b}\in P\cup O} \beta_\alpha(\mathtt{b}),$$

where the second equality follows from definition of necessary clones and the fact that $\beta_\alpha|_{\mathbb{B}^o} \equiv \mathtt{a}_\otimes$, and the last one holds by the idempotency of $\oplus$. The proof is complete. □

## 6 Conclusion

The goal of the work presented in this paper was to tackle the issue of quantitative analysis of attack–defense trees in which a basic action can appear multiple times. We have presented conditions ensuring that in this setting the classical, fast bottom-up procedure for attributes evaluation yields valid result. For a subclass of attributes, we have identified necessary and sufficient condition for compatibility of the bottom-up evaluation with the set semantics. A constructive method of evaluation of attributes belonging to a wide and important subclass of attributes, that takes the presence of repeated labels into account, has been presented.

This work addresses only the tip of the iceberg of a much larger problem which is the analysis and quantification of attack–defense trees with dependent

actions. The notion of clones captures the strongest type of dependency between goals, namely where the nodes bearing the same label represent exactly the same instance of the same goal. It is thus obvious that the attribute values for the clones should only be considered once in the attribute computations. However, in practice, weaker dependencies between goals may also be present. For instance, when the attacker has access to a computer with sufficient computation power, the attack consisting in guessing a password becomes de facto the brute force attack and can be performed within a reasonable time, for most of the passwords used in practice. In contrast, if this attack is performed manually, it will, most probably, take much longer to succeed. Similarly, if the attacker knows the victim, guessing their password manually will, in most cases, be faster compared to the situation when the attacker is a stranger to the victim. Of course, this problem can be solved by relabeling the nodes and using differently named goals for the two situations. However, this solution is not in line with the practical usage of attack(–defense) trees whose construction often relies on preexisting libraries of attack patterns where the nodes are already labeled and the labels are as simple as possible. We are currently working on improving the standard bottom-up evaluation procedure for attributes (in the spirit of Algorithm 1) to accommodate such weakly dependent nodes.

Furthermore, it would be interesting to try to generalize Algorithm 1 for the approaches proposed in the past for the restricted class of attack–defense trees without repeated labels. Such approaches include for instance multi-objective optimization defined in [1] and a method for selecting the most suitable set of countermeasures, based on integer linear programing, developed in [20].

***Acknowledgments*** We would like to thank Angèle Bossuat for fruitful discussions on the interpretation of repeated labels in attack–defense trees and on possible approaches to the problem of quantification in the presence of clones.

## 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., Parker, D.: Quantitative Verification and Synthesis of Attack-Defence Scenarios. In: CSF. pp. 105–119. IEEE Computer Society (2016)
3. Bagnato, A., Kordy, B., Meland, P.H., Schweitzer, P.: Attribute decoration of attack–defense trees. IJSSE 3(2), 1–35 (2012)
4. Bossuat, A., Kordy, B.: Evil Twins: Handling Repetitions in Attack–Defense Trees – A Survival Guide. In: GraMSec@CSF 2017. LNCS, vol. 10744, pp. 17–37. Springer (2018)
5. Codetta-Raiteri, D.: BDD based analysis of parametric fault trees. In: Proceedings of the RAMS'06. Annual Reliability and Maintainability Symposium, 2006. pp. 442–449. RAMS'06, IEEE Computer Society, Washington, DC, USA (2006)
6. Fraile, M., Ford, M., Gadyatskaya, O., Kumar, R., Stoelinga, M., Trujillo-Rasua, R.: Using Attack–Defense Trees to Analyze Threats and Countermeasures in an ATM: A Case Study. In: PoEM. LNBIP, vol. 267, pp. 326–334. Springer (2016)

7. 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)
8. Gadyatskaya, O., Jhawar, R., Kordy, P., Lounis, K., Mauw, S., Trujillo-Rasua, R.: Attack trees for practical security assessment: Ranking of attack scenarios with ADTool 2.0. In: QEST. LNCS, vol. 9826, pp. 159–162. Springer (2016)
9. Haasl, D.F., Roberts, N.H., Veselay, W.E., Goldberg, F.F.: Fault tree handbook. Tech. rep., Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Comission (1981)
10. Hermanns, H., Krämer, J., Krcál, J., Stoelinga, M.: The value of attack-defence diagrams. In: POST. LNCS, vol. 9635, pp. 163–185. Springer (2016)
11. Hong, J.B., Kim, D.S., Chung, C.J., Huang, D.: A survey on the usability and practical applications of graphical security models. Computer Science Review 26, 1–16 (2017)
12. Horne, R., Mauw, S., Tiu, A.: Semantics for Specialising Attack Trees based on Linear Logic. Fundam. Inform. 153(1-2), 57–86 (2017)
13. 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)
14. Jhawar, R., Kordy, B., Mauw, S., Radomirovic, S., Trujillo-Rasua, R.: Attack Trees with Sequential Conjunction. In: SEC. IFIP Advances in Information and Communication Technology, vol. 455, pp. 339–353. Springer (2015)
15. Ji, X., Yu, H., Fan, G., Fu, W.: Attack–defense trees based cyber security analysis for CPSs. In: SNPD. pp. 693–698. IEEE Computer Society (2016)
16. Jürgenson, A., Willemson, J.: Computing exact outcomes of multi-parameter attack trees. In: OTM Conferences (2). LNCS, vol. 5332, pp. 1036–1051. Springer (2008)
17. Kordy, B., Mauw, S., Radomirovic, S., Schweitzer, P.: Attack–defense trees. Journal of Logic and Computation 24(1), 55–87 (2014)
18. Kordy, B., Mauw, S., Schweitzer, P.: Quantitative Questions on Attack–Defense Trees. In: ICISC. LNCS, vol. 7839, pp. 49–64. Springer (2012)
19. 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)
20. Kordy, B., Wideł, W.: How well can I secure my system? In: IFM. LNCS, vol. 10510, pp. 332–347. Springer (2017)
21. Mauw, S., Oostdijk, M.: Foundations of Attack Trees. In: ICISC 2005. LNCS, vol. 3935, pp. 186–198. Springer (2005)
22. National Electric Sector Cybersecurity Organization Resource (NESCOR): Analysis of selected electric sector high risk failure scenarios, version 2.0 (2015), `http://smartgrid.epri.com/doc/NESCOR%20Detailed%20Failure%20Scenarios%20v2.pdf`
23. Paja, E., Dalpiaz, F., Giorgini, P.: The socio-technical security requirements modelling language for secure composite services. In: Secure and Trustworthy Service Composition, LNCS, vol. 8900, pp. 63–78. Springer (2014)
24. 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)
25. Ruijters, E., Stoelinga, M.: Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. Computer Science Review 15-16, 29–62 (2015)

26. Schneier, B.: Attack trees. Dr Dobb's Journal of Software Tools (1999)
27. Stecher, K.: Evaluation of large fault-trees with repeated events using an efficient bottom-up algorithm. IEEE Transactions on Reliability pp. 51–58 (1986)
28. Vigo, R., Nielson, F., Nielson, H.R.: Automated generation of attack trees. In: IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014. pp. 337–350 (2014)