Evil Twins: Handling Repetitions in Attack–Defense Trees: A Survival Guide

Angèle Bossuat, Barbara Kordy

To cite this version:

Angèle Bossuat, Barbara Kordy. Evil Twins: Handling Repetitions in Attack–Defense Trees: A Survival Guide. Graphical Models for Security, Aug 2017, Santa Barbara, United States. pp.17-37. hal-01728782

HAL Id: hal-01728782

https://hal.inria.fr/hal-01728782

Submitted on 12 Mar 2018

Evil Twins: Handling Repetitions in Attack–Defense Trees

A Survival Guide

Angèle Bossuat[1,3] and Barbara Kordy[2,3]

[1] University Rennes 1, Rennes, France
[2] INSA Rennes, Rennes, France
[3] IRISA, Rennes, France
{angele.bossuat,barbara.kordy}@irisa.fr

Abstract. Attack–defense trees are a simple but potent and efficient way to represent and evaluate security scenarios involving a malicious attacker and a defender – their adversary. The nodes of attack–defense trees are labeled with goals of the two actors, and actions that they need to execute to achieve these goals. The objective of this paper is to provide formal guidelines on how to deal with attack–defense trees where several nodes have the same label. After discussing typical issues related to such trees, we define the notion of well-formed attack–defense trees and adapt existing semantics to correctly capture the presence of repeated labels.

1 Into the Wild: Introduction

Security analysis and risk assessment are essential to any system facing potential threats. Attack–defense trees allow the security experts to represent and assess the system's security, by illustrating different ways in which it can be attacked and how such attacks could be countered. Formally speaking, attack–defense trees are simple AND-OR trees, but their strength and expressive power relies on intuitive labels that decorate their nodes. These labels describe what the attacker and the defender need to do to achieve their goals, i.e., to attack and defend the system, respectively.

To provide accurate evaluation results, an attack–defense tree must be as precise and versatile as possible. Yet, no hard rules exist on how to label their nodes. To be able to exploit the graphical aspects of attack–defense trees, experts tend to prefer laconic labels which are often too short to fully express the desired meaning. Furthermore, attack–defense trees are frequently reused from one system to another, they may be borrowed from generic libraries of standard attack patterns, and are usually constructed by merging subtrees devised by several experts, not necessarily communicating with each other. Due to all these reasons, it is not rare to find identical labels on separate nodes in an attack–defense tree.

The aim of this work is to formalize attack–defense trees with repeated labels and develop guidelines to handle them properly. We distinguish between simple mislabeling, and cases where the nodes should indeed have the same label. For the latter, we bring out some important differences, and propose solutions more elaborate and less problematic than simply modifying the labels to make them all unique in a given tree. Our specific contributions are as follows

1. *Repeated labels:* we classify repeated labels according to their meaning and propose a new labeling format to properly handle trees with repetitions.
2. *Well-formedness:* we study frequently observed problems related to mislabeling and introduce a notion of well-formed trees to address them.
3. *Formal basis:* we adapt existing formal semantics for attack–defense trees to make sure they are in line with the new labeling scheme that we propose.
4. *Quantification:* we finally show how to preform quantitative analysis using well-formed attack–defense trees.

Related Work. The attack–defense tree's origins lie in attack trees, introduced in 1999 by Schneier to represent attack scenarios in the form of AND-OR trees [15]. Nowadays, there exist numerous variants of attack trees [9], some of which are popular and widely used in the industry to support real-life risk assessment processes [5,13]. Attack–defense trees extend the classical attack trees by complementing them with the defensive point of view [7]. They aim to represent interaction between the attacker and the defender, to give a more precise image of reality.

Although, in practice, attack tree-based models often possess multiple nodes with the same labels, not much fundamental research exists on the topic. From the formal perspective, a semantics based on multisets has been used in [12] and [7] to interpret an attack tree and respectively an attack–defense tree with a set of multisets representing potential ways of attacking a system. This approach supposes that every node with the same label is a separate action to be executed. On the contrary, other works formalize attack tree-based models with propositional formulæ where, due to the use of the logical conjunction and disjunction which are idempotent, all repetitions are ignored [16,10].

Repeated labels might have an impact on quantitative analysis of attack trees. The standard, bottom-up algorithm for quantification, recalled in Sect. 2.3, treats all repeated nodes as separate events [8]. Since quantifying trees with repeated nodes may increase the complexity of the underlying algorithms, some authors, e.g., Aslanyan et all in [1] restrain their considerations to linear trees, i.e., trees with no label repetition, to gain efficiency. The authors of [14] go even further, and provide two variants of an algorithm for the probability computation on attack-countermeasure trees[4] with and without repeated nodes. Finally, most of the works do not consider repetitions explicitly but rather assume that all nodes (including those with the same labels) represent independent events [2,11].

We believe that both approaches – treating all nodes as independent or ignoring repetitions – are too restrictive. The objective of the formalization proposed in this work is to accommodate both of these cases and thus allow for a more faithful modeling of the reality.

[4] Attack-countermeasure trees are yet another security model based on attack trees.

Finally, studying the problem of labels' repetition led us to propose the notion of well-formedness for such trees. Previously, well-formedness of attack(–defense) trees has been addressed in various ways. In [1], attack–defense trees are formalized as typed terms over actions of the attacker and the defender, and well-formed trees are simply identified with the well-typed ones. In [3], Audinot et al. analyze the problem of well-formedness (that they call correctness) of an attack tree with respect to the modeled system. They focus on the well-formedness of the tree refinements by introducing four correctness properties which allow them to express how well an AND/OR combination of the child nodes represents the goal of the parent node. The objective of the well-formedness developed in our work, and defined in Definition 5, is to capture the intuitive construction of a security scenario represented as an attack–defense tree, in a formal way.

2 Know the Flora: Attack–Defense Trees

We start by briefly introducing the attack–defense tree model and summarizing the state of the art on its existing formal foundations. We especially focus the attention on aspects that may influence the meaning and the treatment of trees with repeated labels. For more detailed information on attack–defense trees, their semantics, and their quantitative analysis, we refer the reader to [7].

2.1 The Model

An *attack–defense tree* (ADTree) is a rooted tree with labeled nodes, aiming to describe and evaluate security scenarios involving two (sets of) competing actors: the attacker trying to attack a particular system[5] and the defender trying to protect it against the potential attacks. Labels of the nodes represent the goals that the actors must achieve. Each node has one of two types – *attack* (red circle) or *defense* (green rectangle) – depending on which actor's goal it illustrates. The nodes of an ADTree can have any number of children of the same type. These children represent the *refinement* of the parent's goal into subgoals. The refinement can be *disjunctive* (`OR` node) or *conjunctive* (`AND` node). To achieve the goal represented by an `OR` node, it is necessary and sufficient to achieve at least one of the subgoals represented by its children. To achieve the goal represented by an `AND` node, it is necessary and sufficient to achieve all of the subgoals represented by its children. To graphically distinguish `OR` from `AND` nodes, we use an arc to connect the children of the `AND` nodes. The nodes that do not have any children of the same type are called *non-refined nodes*. Their labels represent the so called *basic actions*, i.e., the actual actions that the actors need to execute to achieve their (sub)goals. Finally, each node of an ADTree can also have at most one child of the opposite type, which represents a *countermeasure*, i.e., a goal of the other actor, the achievement of which disables the goal of the node. Graphically, countermeasures are connected to the nodes they counter by dotted edges. A countermeasure can, in turn, be refined and/or countered.

[5] The system can be an infrastructure, a computer program, an organization, etc.

Remark 1. Note that the root node of an ADTree can be of the attack or the defense type. The actor whose goal is represented by the label of the root node is called the *proponent*, and the other one is called the *opponent*. In practice, the proponent is the attacker, in most cases.

Example 1. Fig. 1 shows a simple example of an ADTree. In this scenario, the attacker (proponent) is a student who wants to pass a multiple choice test examination. To be sure that she will answer all questions correctly, she needs to learn the exam questions and get the solutions in advance, in order to memorize the correct answers. She can get a copy of the exam by accessing the teacher's computer, finding the file containing the questions, and storing it either by printing it or by saving it on a USB stick. She can proceed in a similar way to get a copy of the file with the solutions, which is located on the same computer. However, to better protect his exam, the teacher (opponent) could archive and encrypt the solutions' file using PKZIP [6]. The student would then need to break the encryption, for example using the CrackIt tool [17], to be able to access the solutions.

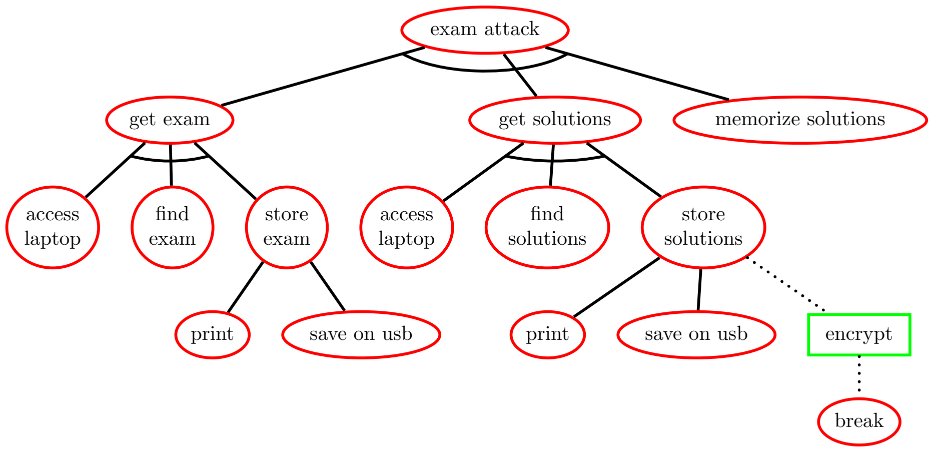


Fig. 1: An ADTree for passing the examination

Since, in ADTrees for real-life scenarios, the number of nodes tends to grow drastically, the graphical representation, as used in Fig. 1, is often not the most appropriate one. To formally describe and manipulate ADTrees, we therefore introduce an alternative, term-based notation.

Let $\mathbb{B}$ be the set of all basic actions. We assume that the elements of $\mathbb{B}$ are typed, i.e., that $\mathbb{B}$ is partitioned into the basic actions of the proponent $\mathbb{B}^{\mathrm{p}}$ and those of the opponent $\mathbb{B}^{\mathrm{o}}$. The ADTrees are generated by the following grammar

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

where $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$, $\bar{\mathrm{p}} = \mathrm{o}$, $\bar{\mathrm{o}} = \mathrm{p}$, and $\mathtt{b}^{\mathrm{s}} \in \mathbb{B}^{\mathrm{s}}$. Whenever the type s of a basic action $\mathtt{b}^{\mathrm{s}}$ is clear from the context, we omit the superscript s to simplify the presentation. Note that, as explained in [7], a term of the form $\mathtt{C}^{\mathrm{s}}(T_1^{\mathrm{s}}, T_2^{\bar{\mathrm{s}}})$ represents the ADTree obtained from attaching (using a dotted edge) the tree $T_2^{\bar{\mathrm{s}}}$ to the root of the tree T_1^{s}.

According to Remark 1, terms of the form T^{p} represent ADTrees, since the label of the root node always illustrates the proponent's goal. In the rest of this paper, we identify an ADTree with its corresponding term. The set of all ADTrees is denoted by $\mathbb{T}$.

Example 2. The term representation of the ADTree from Fig. 1 is the following

$$\begin{aligned}\mathtt{AND}^{\mathrm{p}}\Big(&\mathtt{AND}^{\mathrm{p}}\Big(\mathtt{lapt}, \mathtt{ex}, \mathtt{OR}^{\mathrm{p}}\big(\mathtt{pr}, \mathtt{usb}\big)\Big),\\ &\mathtt{AND}^{\mathrm{p}}\Big(\mathtt{lapt}, \mathtt{sol}, \mathtt{C}^{\mathrm{p}}\big(\mathtt{OR}^{\mathrm{p}}(\mathtt{pr}, \mathtt{usb}), \mathtt{C}^{\mathrm{o}}(\mathtt{enc}, \mathtt{break})\big)\Big), \mathtt{memo}\Big).\end{aligned}$$

2.2 Existing Semantics for ADTrees

It is well-known that two security experts may produce two visually different ADTrees to represent the same security scenario. The simplest (but far from the sole) example of this situation are the two trees $T = \mathtt{AND}^{\mathrm{p}}(\mathtt{card}, \mathtt{pin})$ and $T' = \mathtt{AND}^{\mathrm{p}}(\mathtt{pin}, \mathtt{card})$. Here, the objective of the proponent is to get the necessary credentials to withdraw money from a victim's account. The order in which the proponent obtains the card and the corresponding pin is not relevant – the only thing that matters is that eventually, he gets both: the card and the pin.

To formally capture the notion of equivalent ADTrees, formal semantics for ADTrees have been introduced. Different semantics focus on different aspects (e.g., order of actions, their multiple occurrences, or cause-consequence relationships) and allow to partition the set $\mathbb{T}$ into equivalence classes according to these aspects. This is achieved by assigning mathematical objects to ADTrees, for instance propositional formulæ or multisets of basic actions, in such a way that trees representing the same scenario are interpreted with the same object. It is important to note that two trees may represent the same situation with respect to some aspects, i.e., be equivalent in one semantics, but differ substantially when other aspects are taken into account. Formal semantics also facilitate the reasoning about ADTrees, because they reduce it to the analysis of the corresponding mathematical objects. In general, any equivalence relation on $\mathbb{T}$ can be seen as a semantics for ADTrees.

Definitions 1 and 2 recall two major semantics for ADTrees – the propositional and the multiset semantics. Full details can be found in [7].

Definition 1. *The propositional semantics for ADTrees is a function $\mathcal{P}$ that assigns to each ADTree 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 $x_{\mathtt{b}}$, for $\mathtt{b} \in \mathbb{B}$, is a propositional variable. Two ADTrees are equivalent wrt $\mathcal{P}$ if their interpretations are equivalent propositional formulæ.

The recursive construction from Definition 1 starts by assigning a propositional variable to each basic action $\mathtt{b} \in \mathbb{B}$. This means that if a tree contains two nodes having the same label, these nodes will be interpreted with the same propositional variable. In addition, the logical disjunction ($\vee$) and conjunction ($\wedge$) used to interpret the refined nodes are idempotent. This implies that the propositional semantics $\mathcal{P}$ is unable to take the multiplicity of actions into account, as illustrated in Example 3.

Example 3. Consider the student/teacher scenario from Example 1. In order to access the teacher's laptop, the student would need to access the teacher's office. To do so, she needs to access the building – either by breaking-in through the window or by picking the lock – and then access the office by picking its lock, as illustrated in Fig. 2. When the propositional semantics is used, this tree is equivalent to its simplified form composed of a single node `lock-picking`. This is due to the absorption law which implies that $f_1 = (\mathtt{window} \vee \mathtt{pick}) \wedge \mathtt{pick}$ and $f_2 = \mathtt{pick}$ are equivalent formulæ. We discuss the link between the two trees in more detail in Remark 2.

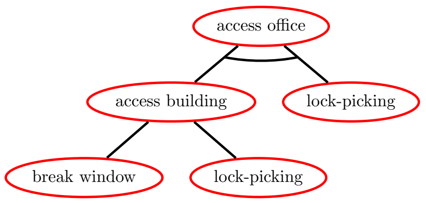


Fig. 2: ADTree for accessing office

Example 3 shows how the propositional semantics models that the execution of one of the repeated actions in the tree activates all other occurrences of this action in the considered scenario. In contrast, the multiset semantics, that we denote with $\mathcal{M}$ and briefly present below, treats each repeated action as a separate event. The multiset semantics has first been introduced in [12] to formalize attack trees and has then been extended to ADTrees in [7]. This semantics interprets an ADTree with a set of pairs of the form (P, O) of *multisets*[6] describing how the proponent can reach the goal represented by the tree: the first multiset P consists of basic actions from $\mathbb{B}^{\mathrm{p}}$ that the proponent has to do, and the second multiset O contains basic actions from $\mathbb{B}^{\mathrm{o}}$ that the proponent must stop the opponent from performing. The construction of $\mathcal{M}$ uses the distributive product $\otimes$ defined for two sets of pairs[7] as:

$$S \otimes Z = \{(P_S \uplus P_Z, O_S \uplus O_Z) | (P_S, O_S) \in S,\ (P_Z, O_Z) \in Z\},$$

where $\uplus$ is the multiset union. Definition 2 formalizes the construction of $\mathcal{M}$.

[6] A multiset is a collection that allows multiple occurrences of an element.

[7] $\otimes$ can be generalized on any finite number of set of pairs, in a natural way.

Definition 2. *The multiset semantics for ADTrees is a function $\mathcal{M}$ that assigns to each ADTree a set of pairs of multisets, as follows*

$$
\begin{aligned}
&\mathcal{M}(\mathtt{b}^{\mathrm{p}}) = \{(\{\!|\mathtt{b}|\!\}, \emptyset)\}, && \mathcal{M}(\mathtt{b}^{\mathrm{o}}) = \{(\emptyset, \{\!|\mathtt{b}|\!\})\},\\
&\mathcal{M}\left(\mathtt{OR}^{\mathrm{p}}(T_1^{\mathrm{p}}, \dots, T_k^{\mathrm{p}})\right) = \bigcup_{i=1}^{k} \mathcal{M}(T_i^{\mathrm{p}}), && \mathcal{M}\left(\mathtt{OR}^{\mathrm{o}}(T_1^{\mathrm{o}}, \dots, T_k^{\mathrm{o}})\right) = \bigotimes_{i=1}^{k} \mathcal{M}(T_i^{\mathrm{o}}),\\
&\mathcal{M}\left(\mathtt{AND}^{\mathrm{p}}(T_1^{\mathrm{p}}, \dots, T_k^{\mathrm{p}})\right) = \bigotimes_{i=1}^{k} \mathcal{M}(T_i^{\mathrm{p}}), && \mathcal{M}\left(\mathtt{AND}^{\mathrm{o}}(T_1^{\mathrm{o}}, \dots, T_k^{\mathrm{o}})\right) = \bigcup_{i=1}^{k} \mathcal{M}(T_i^{\mathrm{o}}),\\
&\mathcal{M}\left(\mathtt{C}^{\mathrm{p}}(T_1^{\mathrm{p}}, T_2^{\mathrm{o}})\right) = \mathcal{M}(T_1^{\mathrm{p}}) \otimes \mathcal{M}(T_2^{\mathrm{o}}), && \mathcal{M}\left(\mathtt{C}^{\mathrm{o}}(T_1^{\mathrm{o}}, T_2^{\mathrm{p}})\right) = \mathcal{M}(T_1^{\mathrm{o}}) \cup \mathcal{M}(T_2^{\mathrm{p}}).
\end{aligned}
$$

Two ADTrees are equivalent wrt $\mathcal{M}$ if they are interpreted with the same set of pairs of multisets.

Due to the use of multisets, the multiset semantics models that the execution of one of the repeated actions has no effect on other occurrences of this action in the considered scenario. In particular, the two trees considered in Example 3 are not equivalent when the multiset semantics is used. Indeed, the tree from Fig. 2 is interpreted with the set $\{(\{\!|\mathtt{window}, \mathtt{pick}|\!\}, \emptyset), (\{\!|\mathtt{pick}, \mathtt{pick}|\!\}, \emptyset)\}$ and the tree composed of a single node `lock-picking` with the set $\{(\{\!|\mathtt{pick}|\!\}, \emptyset)\}$.

Remark 2. The scenario considered in Example 3 shows that both semantics — $\mathcal{P}$ and $\mathcal{M}$ — are useful. When the modeler is interested only in what skills are necessary to perform the access office attack, then the propositional semantics is sufficient. Here, we assume that the attacker who has lock-picking skills will be able to use them at any time. However, if the goal of the security expert is to enumerate and analyze the actual ways of attacking, then the multiset semantics is the correct one to be used.

2.3 Quantitative Evaluation of ADTrees

To complete the overview of formal foundations for ADTrees, we briefly recall the bottom-up procedure for their quantitative evaluation. The simple tree structure of ADTrees can be exploited to easily quantify security scenarios. The security expert's objective might be to find out which way of attacking is the cheapest or the fastest one, whether the proponent's goal can be reached even in the presence of some countermeasures deployed by the opponent, to estimate the probability that the root goal will be achieved, etc. The idea is to assign values to the non-refined nodes and then propagate them all the way up to the root, using functions that depend on the refinement and the type of the node. This process is called *bottom-up attribute's evaluation*, and it is formalized in Definition 3. We refer the reader to [8] for a detailed classification of existing attributes, and to [4] for practical guidelines regarding the attributes' evaluation on ADTrees.

Definition 3. *Let D_α be a set of values. An attribute α is composed of*

– *a basic assignment $\beta_\alpha \colon \mathbb{B} \to D_\alpha$ which assigns a value from D_α to every basic action, and*
– *an attribute domain $A_\alpha = (D_\alpha, \mathtt{OR}^{\mathrm{p}}_\alpha, \mathtt{AND}^{\mathrm{p}}_\alpha, \mathtt{OR}^{\mathrm{o}}_\alpha, \mathtt{AND}^{\mathrm{o}}_\alpha, \mathtt{C}^{\mathrm{p}}_\alpha, \mathtt{C}^{\mathrm{o}}_\alpha)$, where for $\mathtt{OP}^{\mathrm{s}} \in \{\mathtt{OR}^{\mathrm{s}}, \mathtt{AND}^{\mathrm{s}}, \mathtt{C}^{\mathrm{s}}\}$, $\mathtt{OP}^{\mathrm{s}}_\alpha \colon D^k_\alpha \to D_\alpha$ is an internal operation on D_α of the same arity as $\mathtt{OP}^{\mathrm{s}}$.*

The bottom-up algorithm for α assigns values from D_α to ADTrees as follows

$$\alpha(\mathtt{b}) = \beta_\alpha(\mathtt{b}), \qquad \alpha\big(\mathtt{OP}^{\mathrm{s}}(T^{\mathrm{s}}_1, \dots, T^{\mathrm{s}}_k)\big) = \mathtt{OP}^{\mathrm{s}}_\alpha\big(\alpha(T^{\mathrm{s}}_1), \dots, \alpha(T^{\mathrm{s}}_k)\big).$$

Example 4 illustrates the evaluation of the minimal time attribute on the ADTree from Fig. 1.

Example 4. Here, we are interested in the minimal time required for the student to perform the exam attack illustrated in Fig. 1. First, we build the basic assignment function β_α which assigns a value (in minutes) to each basic action.[8]

goal	β_α	goal	β_α	goal	β_α	goal	β_α
`access laptop`	45	`print`	6	`find solutions`	5	`break`	18
`find exam`	5	`save on usb`	1	`memorize`	90	`encrypt`	$+\infty$

Table 1: Basic assignment for the minimal time attribute in the student attack

To propagate the values of minimal attack time up to the root node, the following attribute domain is used $A_{\text{time}} = (\mathbb{N} \cup \{+\infty\}, \min, +, +, \min, +, \min)$. The corresponding bottom-up computation is given in Fig. 3, and shows that the student's attack will take at least 210 minutes.

3 The Root of the Problem: Common Issues

Although constructing an ADTree seems to be a simple and intuitive task, this process may suffer from several issues. They are related to the completeness or correctness of the models. One of the important sources of modeling problems is a presence of multiple nodes having the same label. In this section, we illustrate the most common mistakes made while creating ADTrees and provide hints to avoid them. This is the first step towards the notion of well-formed ADTrees formalized in Sect. 5.

3.1 Incomplete Refinement

As we have explained in Sect. 2, the children of a refined node represent subgoals that need to be achieved so that the goal of the node is achieved. Fig. 4a shows an example of a tree where in order to access the teacher's computer, the student needs to get their username and password. This example presents a problem of

[8] The $+\infty$ value assigned to the basic actions of the defender signifies that the attacker cannot successfully perform these actions, see [8] for a detailed explanation.

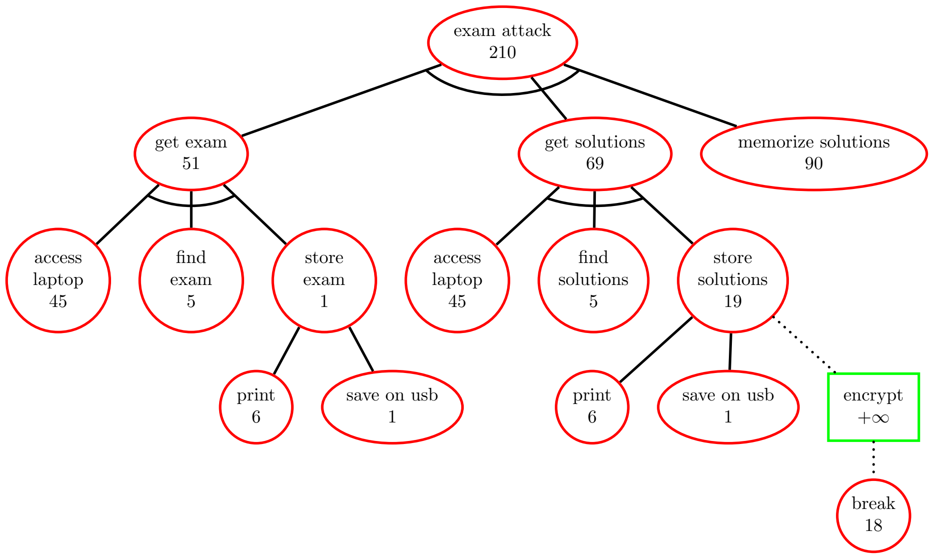


Fig. 3: Minimal time for the student to perform the exam attack

incomplete refinement, as getting the username and the password is not sufficient to access the computer – the action of actually accessing the machine is also necessary. The corrected tree is given in Fig. 4b.

The aim of this example is to illustrate that the label of a refined node cannot represent any additional action to be executed along those already represented by its children. This label is just a short description *replacing but not complementing* the refinement of the node. Note that this is due to the way in which the formal semantics for ADTrees work: the meaning of a refined node is fully expressed as the combination of its children. Similarly, in the case of the bottom-up quantification of ADTrees, the value of a refined node is computed as the combination of the values of its children.

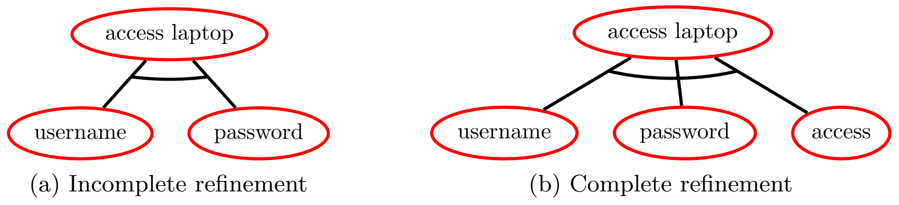
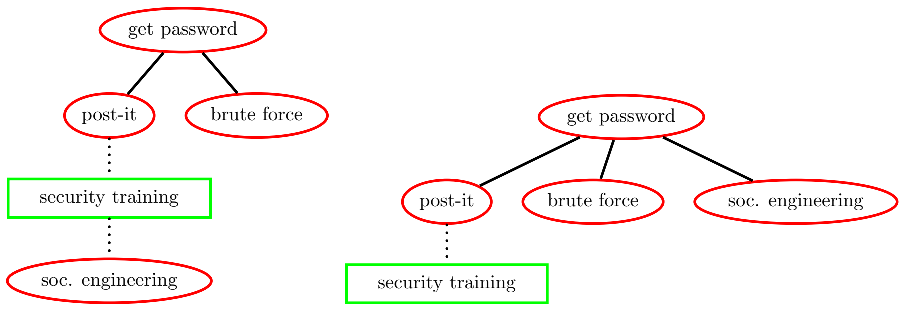


Fig. 4: Well-formedness and refinements

Hint 1: One can easily check whether all nodes of an ADTree are fully refined by hiding the labels of the refined nodes and judging whether the corresponding goal is achieved, or if additional children nodes need to be added.

3.2 Misplaced Counter

A second frequent mistake is to misplace a counter node. Consider the tree from Fig. 5a, where in order to get the teacher's password, the student can find a post-it where the password has been written, or perform a brute force attack. In order to prevent the first attack, a security training could be offered to the teachers to advise them against writing down their passwords. The student could in turn overcome the security training by social engineering the teacher to reveal the password. Note, however, that by social engineering the teacher, the student would already achieve the `get password` goal. In the case of this tree, the `social engineering` node is not a counterattack to security training. It is actually yet another option to get the teacher's password. The correct tree should therefore look like the one in Fig. 5b. A simple analysis of the two trees from Fig. 5 shows that they are not equivalent in any semantics. Similarly, the bottom-up algorithm would give different quantitative results on these two trees.



(a) Incorrect counterattack (b) Correct tree

Fig. 5: Well-formedness and counter placement

Hint 2: A countermeasure node of type s is correctly placed if its achievement disables the goal of the node of type $\bar{s}$ it is supposed to counter.

3.3 Repeated Labels

Due to the fact that ADTrees are often reused to model similar situations and that libraries of standard attacks might be used to ease the creation of an ADTree, it is not rare to see trees that contain several nodes having the same label. In this case, the modeler needs to ensure that the same labels really represent the same goals to be achieved. As a consequence, subtrees rooted in the nodes having the same label need to have the same refining subtrees. More precisely, the subtrees rooted in these nodes, and obtained by removing all nodes of the other type, should be equivalent with respect to the semantics that is used.

Hint 3: If two nodes in an ADTree have the same labels but their refining subtrees are not equivalent with respect to the considered semantics, then

– If the subgoals represented by the refinement of each of these nodes actually apply to the other one as well, then the refining nodes present in only one of the subtrees should be added to the other one and the labels of the two nodes stay unchanged.
– If at least one subgoal does not apply to both nodes, the corresponding goals are not identical, and at least one of the labels should be modified. Note that it does not, however, mean that these goals cannot have common subgoals.

3.4 Repeated Basic Actions

Finally, we need to ensure that nodes labeled with the same basic action represent the same actions to be executed. This implies that, if two non-refined nodes have the same label, then there must not exist any attribute for which they have a different value. If there is at least one attribute that differentiates them, then the basic actions they represent are not identical, and should therefore be modeled with different labels. For instance, if we assume that the exam considered in the tree from Fig. 1 is a multiple choice test composed of five pages of questions, the two `print` nodes should not have the same label, because printing the exam would take more time than printing just the solutions that fit on one page.

Hint 4: To decide whether the non-refined nodes are correctly labeled, the modeler should ensure that, for every attribute α that will be considered, all nodes with the same label are indeed getting the same value by the assignment β_α.

4 Poisonous or Edible: ADTrees with Repeated Labels

As we have seen in Sect. 3, ADTrees may contain several nodes with identical labels. In this section, we propose a methodology to handle such trees properly. We first discuss different origins of repeated labels, and then propose solutions to avoid an incorrect labeling that could lead to miscalculations.

4.1 Meaning of Repeated Labels

While analyzing ADTrees for real-life scenarios, one can observe that there are two kinds of nodes with repeated labels. We call them *cloned* nodes and *twin* nodes. We explain the difference between them below. To clarify the explanation, we say that the node has been *activated* if the goal represented by its label is achieved by the corresponding actor. When the goal of a node has been countered by the other actor, we say that the node has been *deactivated.*

Cloned nodes – when activating one node means activating another one having the same label, we say that the two nodes are cloned. This means that cloned nodes represent exactly the same instance of an action, so deactivating one of the cloned nodes deactivates all its clones.

Twin nodes – when activating one node does not activate another one having the same label, we say that the two nodes are twins. This means that each individual twin node represents a separated instance of the same action, thus all twin nodes having the same label need to be deactivated separately.

Example 5. Consider the scenario from Example 1 illustrated in Fig. 1. The two `access laptop` nodes are cloned: since the exam and the solution files are stored on the same laptop, accessing the laptop needs to be done only once. In contrast, the two `save on usb` nodes are twins: obviously, saving the exam file on a usb stick does not result in saving the solution file, and vice versa. Note that one could dispute the fact that the two `save on usb` nodes have the same label, but according to *Hint 4*, this is correct. Even though the solution and the exam are two different files, they have roughly the same size and will therefore take the same time to be copied to the usb stick. Every other attribute gives unquestionably the same value.

Existing semantics for ADTree have a rather restrictive view on ADTrees with repeated labels. The propositional semantics acts as if all nodes having the same labels were cloned: the labels of non-refined nodes are interpreted as propositional variables and idempotent logical operators ($\vee, \wedge$) are used to interpret the refinements. In contrast, the multiset semantics assumes that all nodes with repeated labels are twins: due to the use of the multisets, where the multiplicity of elements in the collection is relevant, each node is viewed as representing a separate instance of an action. In practice, however, the same scenario may contain both cloned and twin nodes, as illustrated in Example 5. To overcome this issue and accommodate cloned and twin nodes, we propose a more precise labeling scheme, that we present in Sect. 4.2.

4.2 Extended Labeling for ADTrees with Repetitions

A naive solution would be to relabel all twin nodes to remove repetitions. This would make the use of the propositional semantics possible, but this solution is not preferred due to the following issues

- It would prohibit the re-use of models created for similar scenarios, and make the use of libraries of standard attacks more complex.
- Since the number of possible labels, i.e., the size of $\mathbb{B}$, would increase, the effort of defining β_α would be (unnecessary but inevitably) greater. E.g., instead of providing one single value to quantify the complexity of brute forcing a 15 char password, one would need to define two values: for `brute force a 15 char pwd for a laptop` and for `brute force a 15 char pwd for a smartphone`. However, these two values would obviously be the same.
- Relabeling could result in peculiar, non-intuitive labels, which could have a disadvantageous influence on the tree analysis, especially regarding the estimation of the values for basic actions, i.e., definition of β_α.
- Finally, the cloned nodes would still be considered multiple times when the multiset semantics would be used.

To bypass the above issues, we propose a solution which relies on labels being pairs in $\mathbb{G} \times \Gamma$, where $\mathbb{G}$ is a typed set of goals containing $\mathbb{B}$ and Γ is a finite set of indices. Instead of label $\mathtt{g}$, a pair $(\mathtt{g}, \gamma)$ is used. Its first component $\mathtt{g} \in \mathbb{G}$ describes the goal to be achieved and the second component $\gamma \in \Gamma$ is an index which allows us to distinguish between cloned and twin nodes.

Definition 4. *Let T be an ADTree whose nodes are labeled with the elements of $\mathbb{G} \times \Gamma$, and consider two nodes having the same goal $\mathtt{g}$, i.e., labeled with $(\mathtt{g}, \iota)$ and $(\mathtt{g}, \gamma)$, respectively. If $\iota = \gamma$, then we say that the two nodes are cloned; if $\iota \neq \gamma$, then we say that the two nodes are twins (or twin nodes).*

From now on, the word *label* stands for the pair of the form $(\mathtt{g}, \gamma)$ and $\mathtt{g}$ is called its *goal*. Note that, since goals are typed, the set $\mathbb{G}$ is partitioned into goals of the proponent's type ($\mathbb{G}^{\mathrm{p}}$) and those of the opponent's type ($\mathbb{G}^{\mathrm{o}}$). However, if this does not lead to confusion, we omit the superscript denoting the goal's type.

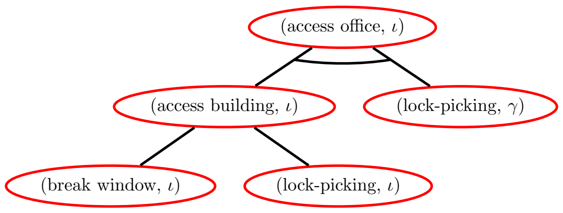


Fig. 6: Extended labeling of ADTree nodes

Example 6. While using the extended labeling based on pairs, the tree from Fig. 2 is relabeled as shown in Fig. 6. It is now clear that $(\mathtt{lock\text{-}picking}, \iota)$ and $(\mathtt{lock\text{-}picking}, \gamma)$ represent two separate instances of picking a lock, as $\iota \neq \gamma$.

5 Survival Kit: Well-formed ADTrees

The purpose of ADTrees is to represent and analyze the security scenarios in a rigorous way. In order to obtain meaningful analysis results, an ADTree must model the reality in the most faithful way possible. To achieve this, we have developed a set of rules that guide the security expert in creating well-formed trees. This is feasible thanks to the new labeling introduced in Sect. 4, which undoubtedly increases the expressive power of ADTrees. In this section, we formalize the notion of well-formed ADTrees that overcome the typical problems illustrated in Sect. 3. Then, we explain how to use them correctly, by adapting previously seen formal semantics and the quantification algorithm to our pair-based labeling.

5.1 Definition of Well-formed ADTrees

To be able to address the issues presented in Sect. 3, we first extend the grammar (1) so that the generated terms capture the labels of the refined nodes:

$$T^{\mathrm{s}}\colon (\mathtt{b}^{\mathrm{s}}, \gamma) \;\mid\; \mathtt{OR}^{\mathrm{s}}[(\mathtt{g}, \gamma)](T^{\mathrm{s}}, \dots, T^{\mathrm{s}}) \;\mid\; \mathtt{AND}^{\mathrm{s}}[(\mathtt{g}, \gamma)](T^{\mathrm{s}}, \dots, T^{\mathrm{s}}) \;\mid\; \mathtt{C}^{\mathrm{s}}(T^{\mathrm{s}}, T^{\bar{\mathrm{s}}}), \quad (2)$$

where $s, \bar{s} \in \{p, o\}$, $\mathtt{b}^s \in \mathbb{B}$, and $\mathtt{g} \in \mathbb{G}$ are goals of refined nodes. The term starting with C does not mention any label, since the goal of the root node expressed by the term $\mathtt{C}^s(T_1^s, T_2^{\bar{s}})$ is contained in the label of the root node of T_1^s.

The presence of extended labels in the terms generated by grammar (2) allows us to explain the meaning of refinement and counter, formalize the hints from Sect. 3.1–3.3, and differentiate between the cloned and the twin nodes. This is captured by the notion of well-formed ADTrees that we introduce in Definition 5. Note that, identifying well-formed ADTrees with the well-typed ones, as in [1], is not sufficient for our work, in particular because it does not capture the problems described in Sect. 3. Indeed, all trees considered in that section are well-typed, but we have shown that they suffer from multiple construction drawbacks that could hinder the security analysis. We therefore believe that the definition of well-formed ADTrees needs to take into account the labels of every node (not only the non-refined ones), the semantics that will be used for the tree analysis, and the attribute domains for the considered attributes.

Definition 5. *Let T be an ADTree generated by grammar (2). ADTree T is said to be well-formed if and only if the type of its root node is* p *(proponent) and all of the following conditions are satisfied for all of its subtrees Y, where $\mathtt{g}_i$ denotes the goal of the root node of Y_i.*

1. ***The meaning of* OR**
   Let $Y = \mathtt{OR}^s[(\mathtt{g}, \gamma)](Y_1^s, \ldots, Y_k^s)$. The goal $\mathtt{g}$ is achieved if and only if at least one of the subgoals $\mathtt{g}_i$ is achieved.
2. ***The meaning of* AND**
   Let $Y = \mathtt{AND}^s[(\mathtt{g}, \gamma)](Y_1^s, \ldots, Y_k^s)$. The goal $\mathtt{g}$ is achieved if and only if all of the subgoals $\mathtt{g}_i$ are achieved.
3. ***The meaning of* C**
   Let $Y = \mathtt{C}^s(Y_1^s, Y_2^{\bar{s}})$. If $\mathtt{g}_2$ is achieved then $\mathtt{g}_1$ cannot be achieved.
4. ***Cloned and twin nodes***
   Let $\mathcal{I}$ be a semantics that will be used for the analysis of T and assume that T contains two subtrees of the form $Y_i = \mathtt{OP}_i^s[(\mathtt{g}_i, \gamma_i)](Y_{i_1}, \ldots, Y_{i_k})$, where $\mathtt{OP}_i^s \in \{\mathtt{OR}^s, \mathtt{AND}^s\}$, for $i \in \{1, 2\}$. Let $Y_{i|s}$ denote the term obtained from Y_i by recursively replacing all of its subterms of the form $\mathtt{C}^s(U_{i_1}, U_{i_2})$ by U_{i_1}.[9] If $\mathtt{g}_1 = \mathtt{g}_2$, then $\mathcal{I}(Y_{1|s}) = \mathcal{I}(Y_{2|s})$, i.e., the subtrees refining $\mathtt{g}_1$ and $\mathtt{g}_2$ are equivalent wrt $\mathcal{I}$. Moreover, if $(\mathtt{g}_1, \gamma_1) = (\mathtt{g}_2, \gamma_2)$, i.e., the corresponding nodes are cloned, then $\mathcal{I}(Y_1) = \mathcal{I}(Y_2)$.
5. ***Correct labeling***
   Let α be an attribute that will be used for the analysis of T and assume that T contains two subtrees of the form $Y_i = \mathtt{OP}_i^s[(\mathtt{g}_i, \gamma_i)](Y_{i_1}, \ldots, Y_{i_k})$, where $\mathtt{OP}_i^s \in \{\mathtt{OR}^s, \mathtt{AND}^s\}$, for $i \in \{1, 2\}$. Additionally, let $Y_{i|s}$ be as in the previous item. If $\mathtt{g}_1 = \mathtt{g}_2$, then $\alpha(Y_{1|s}) = \alpha(Y_{2|s})$. Moreover, if $(\mathtt{g}_1, \gamma_1) = (\mathtt{g}_2, \gamma_2)$, i.e., the corresponding nodes are cloned, then $\alpha(Y_1) = \alpha(Y_2)$.

Rules 1 and 2 guarantee the correctness and completeness of refinements. They implement *Hint 1* from Sect. 3. For instance, the tree from Fig. 4a is not

[9] In other words, $Y_{i|s}$ is the tree Y_i in which all countermeasures have been disregarded.

well-formed, because it does not satisfy rule 2. Rule 3 is related to *Hint 2*. The tree from Fig. 5a does not satisfy rule 3 because a successful social engineering attack does not counter the security training. Rule 4 formalizes *Hint 3*. It makes sure that nodes with the same goals (in particular the twin nodes) have equivalent refining subtrees and nodes with the same labels (goals and indices), i.e., the cloned nodes, have equivalent subtrees (including countermeasures). In particular, rule 4 forbids two cloned nodes from being placed on the same path to the root node. Rule 5 corresponds to *Hint 4*. It ensures that nodes with the same labels and non-countered nodes with the same goals always get the same value when the bottom-up quantitative analysis is performed.

In Example 7, we modify the tree from Fig. 1 by extending its labels with the second component, and renaming some of the goals to ensure the well-formedness of the tree. We remark that the unique purpose of the index from Γ is to allow the distinction between the cloned and the twin nodes having the same goal. If two nodes have different goals, the fact that they have the same index does not model any additional relationship between them.

Example 7. As already discussed in Example 5, the two `access laptop` nodes are cloned. They therefore get the same index ι. Since printing the exam will be substantially longer than printing the solutions, the two `print` nodes cannot have the same goal. We therefore rename them to `print exam` and `print sol`. The exam and the solutions differ in terms of the number of pages, nevertheless the size of the corresponding pdf files is practically the same. Therefore, the two `save on usb` nodes may keep the same goal, but their indices must be different, as these nodes are twins. The updated well-formed ADTree is given in Fig 7.

5.2 Formal Semantics for Well-formed ADTrees

We now discuss the formal semantics for well-formed ADTrees labeled with pairs from $\mathbb{G} \times \Gamma$.

Propositional semantics. We require that the propositional variables are associated with labels (i.e., pairs) and not only with goals. We therefore have $\mathcal{P}((\mathtt{b}, \gamma)) = x_{(\mathtt{b},\gamma)}$, and the rest of Definition 1 stays unchanged. If the pair-based labeling is adopted, then cloned nodes are represented with the same variable and are only counted once in the semantics; twin nodes, in turn, correspond to different variables, say $x_{(\mathtt{b},\gamma)}$ and $x_{(\mathtt{b},\gamma')}$, and will thus be treated as separated actions to be performed. The propositional semantics of a well-formed ADTree T can be expressed as a formula in a minimized disjunctive normal form

$$\mathcal{P}(T) = \bigvee_{i=1}^{l} \Big(\big(\bigwedge_{j=1}^{n_i} x_{(\mathtt{p}_{ij},\gamma_{ij})} \big) \wedge \big(\bigwedge_{j=1}^{m_i} \neg x_{(\mathtt{o}_{ij},\gamma_{ij})} \big) \Big), \tag{3}$$

where $\mathtt{p}_{ij} \in \mathbb{B}^{\mathtt{p}}, \mathtt{o}_{ij} \in \mathbb{B}^{\mathtt{o}}$, and $\forall i, \forall j$, if $j \neq j'$, then $(\mathtt{p}_{ij}, \gamma_{ij}) \neq (\mathtt{p}_{ij'}, \gamma_{ij'})$ and $(\mathtt{o}_{ij}, \gamma_{ij}) \neq (\mathtt{o}_{ij'}, \gamma_{ij'})$.

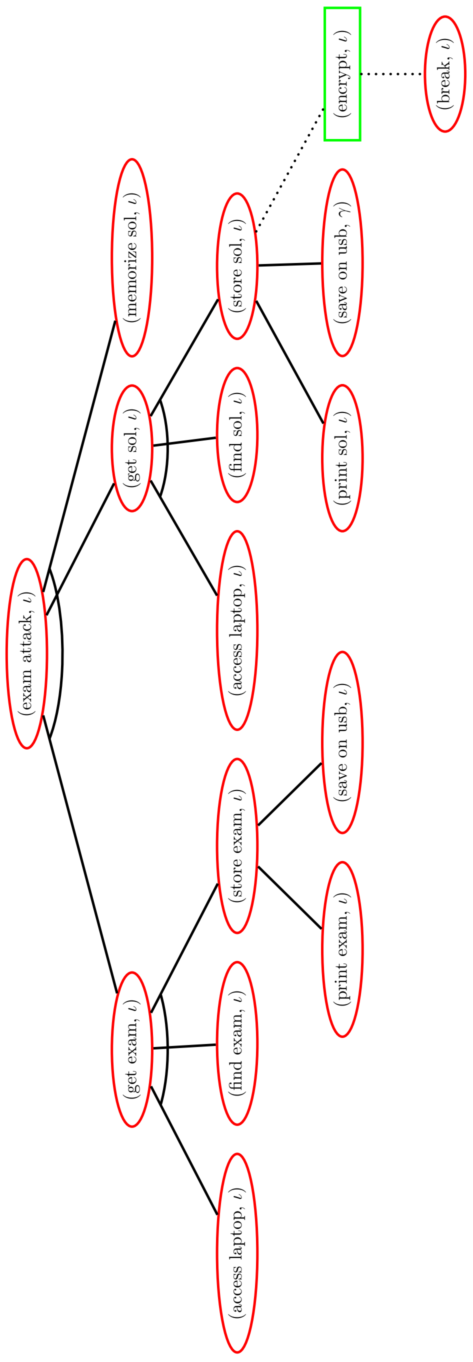


Fig. 7: Well-formed ADTree for passing the examination

Set semantics. The objective of using multisets in the semantics introduced in Definition 2 was to be able to recall the multiplicity of the same goal. When the labeling from $\mathbb{G} \times \Gamma$ is used, the indices already take care of storing information about which actions need to be repeated several times (twin nodes get different indices) and which ones are only executed once (cloned nodes get the same index). We therefore replace the multisets from Definition 2 with regular sets. The corresponding semantics is formally defined in Definition 6, where

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

for $S, Z \subseteq \mathbb{B}^{\mathrm{p}} \times \mathbb{B}^{\mathrm{o}}$.

Definition 6. *The set semantics for ADTrees labeled with pairs from* $\mathbb{G} \times \Gamma$ *is a function* $\mathcal{S}\colon \mathbb{T} \to \mathcal{P}\big(\mathcal{P}(\mathbb{B}^{\mathrm{p}} \times \Gamma) \times \mathcal{P}(\mathbb{B}^{\mathrm{o}} \times \Gamma)\big)$ *that assigns to each ADTree a set of pairs of sets of labels, as follows*

$$\mathcal{S}\left((\mathtt{b}^{\mathrm{p}}, \gamma)\right) = \big\{\big(\{(\mathtt{b}^{\mathrm{p}}, \gamma)\}, \emptyset\big)\big\}, \qquad \mathcal{S}\left((\mathtt{b}^{\mathrm{o}}, \gamma)\right) = \big\{\big(\emptyset, \{(\mathtt{b}^{\mathrm{o}}, \gamma)\}\big)\big\}$$

$$\mathcal{S}\left(\mathtt{OR}^{\mathrm{p}}(T_1^{\mathrm{p}}, \ldots, T_k^{\mathrm{p}})\right) = \bigcup_{i=1}^{k} \mathcal{S}(T_i^{\mathrm{p}}), \qquad \mathcal{S}\left(\mathtt{OR}^{\mathrm{o}}(T_1^{\mathrm{o}}, \ldots, T_k^{\mathrm{o}})\right) = \bigodot_{i=1}^{k} \mathcal{S}(T_i^{\mathrm{o}})$$

$$\mathcal{S}\left(\mathtt{AND}^{\mathrm{p}}(T_1^{\mathrm{p}}, \ldots, T_k^{\mathrm{p}})\right) = \bigodot_{i=1}^{k} \mathcal{S}(T_i^{\mathrm{p}}), \qquad \mathcal{S}\left(\mathtt{AND}^{\mathrm{o}}(T_1^{\mathrm{o}}, \ldots, T_k^{\mathrm{o}})\right) = \bigcup_{i=1}^{k} \mathcal{S}(T_i^{\mathrm{o}})$$

$$\mathcal{S}\left(\mathtt{C}^{\mathrm{p}}(T_1^{\mathrm{p}}, T_2^{\mathrm{o}})\right) = \mathcal{S}(T_1^{\mathrm{p}}) \odot \mathcal{S}(T_2^{\mathrm{o}}), \qquad \mathcal{S}\left(\mathtt{C}^{\mathrm{o}}(T_1^{\mathrm{o}}, T_2^{\mathrm{p}})\right) = \mathcal{S}(T_1^{\mathrm{o}}) \cup \mathcal{S}(T_2^{\mathrm{p}}).$$

The set semantics of a well-formed ADTree T can be expressed as follows

$$\mathcal{S}(T) = \bigcup_{i=1}^{l} \Big\{ \Big(\bigcup_{j=1}^{n_i} \{(\mathtt{p}_{ij}, \gamma_{ij})\}, \bigcup_{j=1}^{m_i} \{(\mathtt{o}_{ij}, \gamma_{ij})\} \Big) \Big\}. \tag{4}$$

Note that expressions (3) and (4) correspond to the canonical form of an ADTree in the respective semantics, i.e., the form that explicitly enumerates possible ways to achieve the tree's root goal, giving the minimum amount of information necessary to reconstruct an equivalent ADTree. Example 8 illustrates the use of the set semantics on a well-formed ADTree.

Example 8. While interpreting the tree from Fig. 7 with the set semantics, we obtain the following eight ways of performing the student attack:

$$\begin{aligned}
&\big\{\big(\{(\mathtt{lapt}, \iota), (\mathtt{ex}, \iota), (\mathtt{sol}, \iota), (\mathtt{memo}, \iota), (\mathtt{break}, \iota), (\mathtt{pr\,ex}, \iota), (\mathtt{pr\,sol}, \iota)\}, \emptyset\big),\\
&\ \big(\{(\mathtt{lapt}, \iota), (\mathtt{ex}, \iota), (\mathtt{sol}, \iota), (\mathtt{memo}, \iota), (\mathtt{break}, \iota), (\mathtt{pr\,ex}, \iota), (\mathtt{usb}, \gamma)\}, \emptyset\big),\\
&\ \big(\{(\mathtt{lapt}, \iota), (\mathtt{ex}, \iota), (\mathtt{sol}, \iota), (\mathtt{memo}, \iota), (\mathtt{break}, \iota), (\mathtt{usb}, \iota), (\mathtt{pr\,sol}, \iota)\}, \emptyset\big),\\
&\ \big(\{(\mathtt{lapt}, \iota), (\mathtt{ex}, \iota), (\mathtt{sol}, \iota), (\mathtt{memo}, \iota), (\mathtt{break}, \iota), (\mathtt{usb}, \iota), (\mathtt{usb}, \gamma)\}, \emptyset\big),\\
&\ \big(\{(\mathtt{lapt}, \iota), (\mathtt{ex}, \iota), (\mathtt{sol}, \iota), (\mathtt{memo}, \iota), (\mathtt{pr\,ex}, \iota), (\mathtt{pr\,sol}, \iota)\}, \{\mathtt{enc}\}\big),\\
&\ \big(\{(\mathtt{lapt}, \iota), (\mathtt{ex}, \iota), (\mathtt{sol}, \iota), (\mathtt{memo}, \iota), (\mathtt{pr\,ex}, \iota), (\mathtt{usb}, \gamma)\}, \{\mathtt{enc}\}\big),\\
&\ \big(\{(\mathtt{lapt}, \iota), (\mathtt{ex}, \iota), (\mathtt{sol}, \iota), (\mathtt{memo}, \iota), (\mathtt{usb}, \iota), (\mathtt{pr\,sol}, \iota)\}, \{\mathtt{enc}\}\big),\\
&\ \big(\{(\mathtt{lapt}, \iota), (\mathtt{ex}, \iota), (\mathtt{sol}, \iota), (\mathtt{memo}, \iota), (\mathtt{usb}, \iota), (\mathtt{usb}, \gamma)\}, \{\mathtt{enc}\}\big)\big\}.
\end{aligned}$$

The different attack options implement distinct ways of storing the exam and the solution files, and depend on whether the solution file is encrypted or not. The first four options correspond to the situation where the teacher can use the encryption, because the student is prepared to break it. The last four cases model that, in addition to the actions listed in the first set, the student should also stop the teacher from encrypting the solution file.

The use of sets (instead of multisets) ensures that accessing the laptop is performed only once. However, thanks to the use of different indices (ι and γ) saving the exam and the solution files on a usb represent two different actions.

Quantitative analysis. If α is an attribute, the function β_α is still of the form $\beta_\alpha \colon \mathbb{B} \to D_\alpha$, i.e., it does not take the indices into account, so two twin nodes having the same goal will get the same value by β_α. This way, the computational burden of estimating values for similar basic actions is omitted. To ensure a correct handling of the cloned nodes, the value of attribute α must now be evaluated on the semantics of the tree. 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 for α. If the propositional semantics (resp. the set semantics) is used, then the evaluation of the tree whose interpretation is given by formula (3) (resp. (4)) proceeds as follows

$$\alpha(T) = (\mathtt{OR}^{\mathrm{p}}_\alpha)_{i=1}^{l}\Big(\mathtt{C}^{\mathrm{p}}_\alpha\big((\mathtt{AND}^{\mathrm{p}}_\alpha)_{j=1}^{n_i}\beta_\alpha(\mathtt{p}_{ij}), (\mathtt{OR}^{\mathrm{o}}_\alpha)_{j=1}^{m_i}\beta_\alpha(\mathtt{o}_{ij})\big)\Big). \tag{5}$$

Example 9. Let us make use of the set semantics to evaluate the minimal time for the exam attack on the well-formed tree from Fig. 7. We use the basic assignment from Example 4, except for the two `print` nodes that now represent distinct basic actions. We set $\beta_\alpha(\mathtt{print\ exam}) = 6$ and $\beta_\alpha(\mathtt{print\ sol}) = 2$, to model that printing a longer document will take more time. The minimal time corresponding to each attack option is as follows

$$\begin{aligned}
&\big(\{(\mathtt{lapt},\iota),(\mathtt{ex},\iota),(\mathtt{sol},\iota),(\mathtt{memo},\iota),(\mathtt{break},\iota),(\mathtt{pr\,ex},\iota),(\mathtt{pr\,sol},\iota)\},\emptyset\big) \mapsto 171,\\
&\big(\{(\mathtt{lapt},\iota),(\mathtt{ex},\iota),(\mathtt{sol},\iota),(\mathtt{memo},\iota),(\mathtt{break},\iota),(\mathtt{pr\,ex},\iota),(\mathtt{usb},\gamma)\},\emptyset\big) \mapsto 170,\\
&\big(\{(\mathtt{lapt},\iota),(\mathtt{ex},\iota),(\mathtt{sol},\iota),(\mathtt{memo},\iota),(\mathtt{break},\iota),(\mathtt{usb},\iota),(\mathtt{pr\,sol},\iota)\},\emptyset\big) \mapsto 166,\\
&\big(\{(\mathtt{lapt},\iota),(\mathtt{ex},\iota),(\mathtt{sol},\iota),(\mathtt{memo},\iota),(\mathtt{break},\iota),(\mathtt{usb},\iota),(\mathtt{usb},\gamma)\},\emptyset\big) \mapsto 165,\\
&\big(\{(\mathtt{lapt},\iota),(\mathtt{ex},\iota),(\mathtt{sol},\iota),(\mathtt{memo},\iota),(\mathtt{pr\,ex},\iota),(\mathtt{pr\,sol},\iota)\},\{\mathtt{enc}\}\big) \mapsto +\infty,\\
&\big(\{(\mathtt{lapt},\iota),(\mathtt{ex},\iota),(\mathtt{sol},\iota),(\mathtt{memo},\iota),(\mathtt{pr\,ex},\iota),(\mathtt{usb},\gamma)\},\{\mathtt{enc}\}\big) \mapsto +\infty,\\
&\big(\{(\mathtt{lapt},\iota),(\mathtt{ex},\iota),(\mathtt{sol},\iota),(\mathtt{memo},\iota),(\mathtt{usb},\iota),(\mathtt{pr\,sol},\iota)\},\{\mathtt{enc}\}\big) \mapsto +\infty,\\
&\big(\{(\mathtt{lapt},\iota),(\mathtt{ex},\iota),(\mathtt{sol},\iota),(\mathtt{memo},\iota),(\mathtt{usb},\iota),(\mathtt{usb},\gamma)\},\{\mathtt{enc}\}\big) \mapsto +\infty.
\end{aligned}$$

According to the formula from equation (5), we obtain that the time of the shortest attack is $\min\{171, 170, 166, 165, +\infty\} = 165$, in contrast to 210 minutes obtained for the non well-formed tree in Fig. 3. The reason is that the action of accessing the laptop is now counted only once. This comparison shows that distinguishing between different types of repeated nodes improves the accuracy of the attack–defense tree analysis. We finally remark that every attack option where the second set is not empty gets value $+\infty$. This models that these options do not represent successful attacks, as they cannot be performed in finite time.

6 Back to Civilization: Conclusion

The goal of the work presented in this paper was to provide guidelines for properly handling attack–defense trees where several nodes have the same label. A thorough analysis of numerous examples of such trees resulted in a classification of the repeated nodes into two categories: cloned nodes and twin nodes. These two kinds of nodes must be treated differently, because activating a cloned node activates all other ones having the same label, while activating one of the repeated twin nodes does not have any influence on the other ones.

To formally capture the difference between the two cases, we have proposed a new labeling scheme which complements the node's goal with the information regarding which repeated nodes are cloned, and which ones are twins. Furthermore, we have extended the classical grammar that generates ADTrees in a way that includes the labels of the refined nodes. This enabled us to define well-formed ADTrees, and formally specify their semantics. The definition of well-formedness ensures that the trees are not only well-typed (with respect to the actions of the proponent and the opponent), but also that they do not suffer from common mistakes or omissions often made during the tree creation process.

Since attack trees are special cases of ADTrees, the solution elaborated in this work directly applies to classical attack trees. We therefore hope that our survival kit will be a valuable and practical help to security experts making use of attack(–defense) trees to model and evaluate the security of their systems.

Repeated labels are just a special case of a much larger problem of dependencies between nodes in ADTrees. In practice, different attacks may share some but not all of the necessary actions, they may involve temporal or causal dependencies between the actions of the two actors, etc. We are currently working on extending the ADTree model with such dependencies in order to be able to analyze scenarios involving sequences (instead of sets of) actions, as well as distinguishing between preventive and reactive countermeasures.

Another aspect that we would like to study is the formulation of the goals in ADTrees. Due to their conciseness, the labels are often imprecise or misleading. In addition, several formulations in the natural language might correspond to the same goal. A methodology to devise precise labels and to decide which formulations are equivalent should be developed, so that the nodes with different but equivalent labels can be treated in the same, and if possible automated, way.

Acknowledgments We would like to thank Wojciech Wideł for the very fruitful discussions on the meaning of countermeasures in ADTrees, which allowed us to improve the approach developed in this paper.

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. Audinot, M., Pinchinat, S., Kordy, B.: Is my attack tree correct? In: ESORICS. LNCS, vol. 10492, pp. 83–102. Springer (2017)
4. Bagnato, A., Kordy, B., Meland, P.H., Schweitzer, P.: Attribute Decoration of Attack–Defense Trees. IJSSE 3(2), 1–35 (2012)
5. Gadyatskaya, O., Harpes, C., Mauw, S., Muller, C., Muller, S.: Bridging two worlds: Reconciling practical risk assessment methodologies with theory of attack trees. In: GraMSec@CSF. LNCS, vol. 9987, pp. 80–93. Springer (2016)
6. Katz, P.: PKZIP 6.0 Command Line User's Manual. PKWare, Inc. (2002), `https://pkware.cachefly.net/webdocs/manuals/win6_cli-usersguide.pdf`
7. Kordy, B., Mauw, S., Radomirovic, S., Schweitzer, P.: Attack–defense trees. J. Log. Comput. 24(1), 55–87 (2014), `http://dx.doi.org/10.1093/logcom/exs029`
8. Kordy, B., Mauw, S., Schweitzer, P.: Quantitative Questions on Attack–Defense Trees. In: ICISC. LNCS, vol. 7839, pp. 49–64. Springer (2012)
9. Kordy, B., Piètre-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)
10. Kordy, B., Pouly, M., Schweitzer, P.: Computational aspects of attack–defense trees. In: SIIS. LNCS, vol. 7053, pp. 103–116. Springer (2011)
11. Kordy, B., Wideł, W.: How well can I secure my system? In: iFM 2017. LNCS, vol. 10510, pp. 332–347. Springer International Publishing (2017)
12. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: ICISC. pp. 186–198 (2005)
13. Paul, S.: Towards automating the construction & maintenance of attack trees: a feasibility study. In: GraMSec@ETAPS. EPTCS, vol. 148, pp. 31–46 (2014)
14. 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)
15. Schneier, B.: Attack trees. Dr Dobb's Journal of Software Tools (1999)
16. Vigo, R., Nielson, F., Nielson, H.R.: Automated generation of attack trees. In: CSF. pp. 337–350. IEEE Computer Society (2014)
17. Wesley, K.J., Anbiah, R.R.J.: Cracking PKZIP files' password. A to Z of C pp. 610–615 (2008)