Quantitative Questions on Attack–Defense Trees

Barbara Kordy, Sjouke Mauw and Patrick Schweitzer

University of Luxembourg, SnT
{barbara.kordy, sjouke.mauw, patrick.schweitzer}@uni.lu

Abstract Attack–defense trees are a novel methodology for graphical security modeling and assessment. The methodology includes visual, intuitive tree models whose analysis is supported by a rigorous mathematical formalism. Both, the intuitive and the formal components of the approach can be used for quantitative analysis of attack–defense scenarios. In practice, we use intuitive questions to ask about aspects of scenarios we are interested in. Formally, a computational procedure, defined with the help of attribute domains and a bottom-up algorithm, is applied to derive the corresponding numerical values.
This paper bridges the gap between the intuitive and the formal way of quantitatively assessing attack–defense scenarios. We discuss how to properly specify a question, so that it can be answered unambiguously. Given a well specified question, we then show how to derive an appropriate attribute domain which constitutes the corresponding formal model. Since any attack tree is in particular an attack–defense tree, our analysis is also an advancement of the attack tree methodology.

Keywords: attack trees, attack–defense trees, attributes, quantitative analysis, quantitative questions.

1 Introduction

In graphical security modeling, the main focus lies on the visual representation of a scenario. A common requirement of graphical models is their user friendliness, and hence their intuitiveness. However, intuitive models are prone to be ambiguous. While this in itself may already be undesirable, ambiguity is detrimental for computer supported processing. Contrary to intuitive models, formal frameworks prevent ambiguity and are able to support automated quantitative evaluation. A disadvantage of formal frameworks is however that they are, not seldom, more difficult to understand.

Attack–defense trees [16] form a systematic, graphical methodology for analysis of attack–defense scenarios. They represent a game between an attacker, whose goal is to attack a system, and a defender who tries to protect the system. The widespread formalism of attack trees is a subclass of attack–defense trees, where only the actions of the attacker are considered. The attack–defense tree methodology combines intuitive and formal components. On the one hand, the intuitive visual attack–defense tree representation is used in practice to answer qualitative and quantitative questions, such as "What are the minimal costs to protect a server?", or "Is the scenario satisfiable?" On the other hand, there exist attack–defense terms and a precise mathematical framework for quantitative analysis using a recursive bottom-up procedure introduced for attack trees in [26] and extended to attack–defense trees in [15].

Several case studies performed using the attack–defense tree methodology showed that there exist a significant discrepancy between users focusing on the intuitive components of the model and users working with the formal components. This is due to the fact that intuitive models are user friendly but often ambiguous. In contrast, formal models are rigorous and mathematically sound. This, however, makes them difficult to understand for users without a formal background. This discrepancy between the two worlds is especially visible in the case of quantitative analysis. Correct numerical evaluation can only be performed when all users have precise and consistent understanding of considered quantities also called attributes.

Contributions. This work is an attempt to bridge the gap between the intuitive and the formal components of the attack–defense tree methodology for quantitative security analysis. Our goal is to provide a precise relation between intuitive questions and their formal models called attribute domains. We elaborate which kind of intuitive questions occurring in practical security analysis can be answered with the help of the bottom-up procedure on attack–defense trees. We empirically classify questions that were

collected during case studies and literature reviews. We distinguish and formally analyze three different classes of questions: those referring to one player, those where answers for both players can be deduced from each other and those relating to an outside third party. For each class we provide detailed guidelines how the questions should be specified, so that they are unambiguous and can be answered correctly. Simultaneously, we discuss templates of the attribute domains corresponding to each class.

Related work. An excellent historical overview on graphical security modeling, starting from fault trees [28], over threat trees [3] and privilege graphs [9] leading up to Schneier's attack trees [26], was given by Piètre-Cambacédès and Bouissou in [22]. When Schneier introduced the attack trees formalism in [26], he proposed how to evaluate, amongst others, attack costs, success probability of an attack, and whether there is a need for special equipment. Since then, many authors have not only extended the attack tree formalism syntactically, but also followed in his footsteps and included the possibility of quantitative analysis in their extended formalisms. Baca and Petersen [4], for example, have extended attack trees to countermeasure graphs and quantitatively analyzed an open-source application development. Bistarelli et al. [6], Edge et al. [10] and Roy et al. [24] have augmented attack trees with a notion of defense or mitigation nodes. They all analyze specific types of risk using particular risk formulas, adjusted to their models. Willemson and Jürgenson [30] introduced an order on the leaves of attack trees to be able to optimize the computation of the expected outcome of the attacker. There also exist a number of case studies and experience reports that quantitatively analyze real-life systems. Notable examples are Henniger et al. [12], who have conducted a study using attack trees for vehicular communications systems, Abdulla et al. [1], who analyzed the GSM radio network using attack jungles, and Tanu and Arreymbi [27], who assessed the security of mobile SCADA system for a tank and pump facility. Since all previously mentioned papers focus on specific attributes, they do not address the general problem of the relation between intuitive and formal quantitative analysis.

The formalism of attack–defense trees considered in this work was introduced by Kordy et al. in [15]. Formal aspects of the attack–defense methodology have been discussed in [14] and [17]. In [5], Bagnato et al. provided guidelines for how to use attack–defense trees in practice. They analyzed a DoS attack scenario on an RFID-based goods management system by evaluating a number of relevant attributes, including cost, time, detectability, penalty, required skill level, impact, difficulty and profitability.

Paper structure. The necessary background concerning the attack–defense tree methodology is briefly explained in Section 2. The relation between intuitive and formal quantitative analysis of attack–defense scenarios is presented in Section 3. This section also introduces our classification of questions that can be answered on attack–defense trees with the help of a bottom-up procedure. The classification contains three classes of questions which are treated in Sections 4, 5 and 6. Section 8 presents a software tool, that has been developed to support quantitative analysis of attack–defense scenarios. Section 9 concludes the paper.

2 Attack–Defense Scenarios Intuitively and Formally

2.1 The Intuitive Model

An *attack–defense tree* (ADTree) constitutes an intuitive graphical model describing the measures an attacker might take in order to attack a system and the defenses that a defender can employ to protect the system. An ADTree is a node-labeled rooted tree having nodes of two opposite types: *attack nodes* represented with circles and *defense nodes* represented with rectangles. The root node of an ADTree depicts the main goal of one of the players. Each node of an ADTree may have one or more children of the same type which *refine* the node's goal into subgoals. The refinement relation is indicated by solid edges and can be either disjunctive or conjunctive. The goal of a *disjunctively* refined node is achieved when *at least one* of its children's goals is achieved. The goal of a *conjunctively* refined node is achieved when *all* of its children's goals are achieved. To distinguish between the two refinements we indicate the conjunctive refinement with an arc. A node which does not have any children of the same type is called a *non-refined* node. Non-refined nodes represent *basic actions*, i.e., actions which can be easily understood and quantified. Every node in an ADTree may also have one child of the opposite type, representing a *countermeasure*. The countermeasure relation is indicated by dotted edges. Nodes

representing countermeasures can again be refined into subgoals and countered by a node of the opposite type.

Example 1. An example of an ADTree is given in Figure 1. The root of the tree represents an attack on a server. Three ways to accomplish this attack are depicted: insider attack, outsider attack (OA) and stealing the server (SS). To achieve his goal, an insider needs to be internally connected (IC) and have the correct user credentials (UC). To not be caught easily, an insider uses a colleague's and not his own credentials. Attack by an outsider can be prevented if a properly configured firewall (FW) is installed.

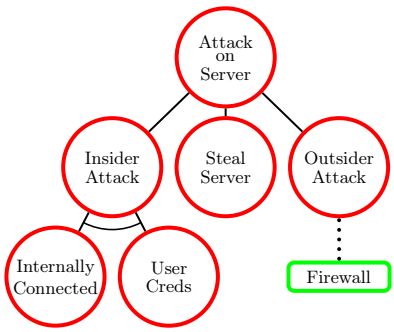

Figure 1. An ADTree for how to attack a server

Graphical visualization of potential attacks and possible countermeasures constitutes a first step towards a systematic security analysis. The next step is to assign numerical values to ADTree models, i.e., to perform a quantitative analysis. Intuitively speaking, performing a quantitative security analysis means *answering questions related to specific aspects or properties influencing the security of a system or a company.* These questions may be of Boolean type, e.g., "Is the attack satisfiable?", or may concern physical or temporal aspects, e.g., "What are the minimal costs of attacking a system?", or "How long does it take to detect the attack?" In order to facilitate and automate the analysis of vulnerability scenarios using ADTrees, the formal model of ADTerms and their quantitative analysis have been introduced. We briefly describe them in the next section.

2.2 The Formal Model

In this section we recall formal definitions related to our methodology. For more details and explanatory examples we refer the reader to [16]. To formally represent and analyze ADTrees, typed terms over a particular typed signature, called the AD–signature, have been introduced in [15]. To be able to capture ADTrees rooted in an attacker's node as well as those rooted in a defender's node, we distinguish between the *proponent* (denoted by p), which refers to the root player, and the *opponent* (denoted by o), which is the other player. For instance, for the ADTree in Figure 1, the proponent is the attacker and the opponent is the defender. Conversely, if the root of an ADTree is a defense node, the proponent is the defender and the opponent is the attacker.

Furthermore, given a set S, we denote by S^* the set of all finite strings over S, and by ε the empty string. For $s \in S$, we denote by s^+ a string composed of a finite number of symbols s.

Definition 1. *The* AD–signature *is a pair* $\Sigma = (S, F)$, *where*

- $S = \{p, o\}$ *is a set of types, and*
- $F = B^p \cup B^o \cup \{\underline{\ }^p, \wedge^p, \vee^p, \underline{\ }^o, \wedge^o, \vee^o, c^p, c^o\}$ *is a set of function symbols, such that the sets* B^p, B^o *and* $\{\underline{\ }^p, \wedge^p, \vee^p, \underline{\ }^o, \wedge^o, \vee^o, c^p, c^o\}$ *are pairwise disjoint.*

Every function symbol $F \in \Sigma$ *is equipped with a mapping* $\mathrm{rank}\colon F \to S^* \times S$, *where* $\mathrm{rank}(F)$ *is defined as a pair* $(\mathrm{in}(F); \mathrm{out}(F))$. *The first component of the pair describes the type of the arguments of* F *and*

the second component describes the type of the values of F. We have

$$\begin{aligned}
&\mathrm{rank}(b) = (\varepsilon; \mathrm{p}), \text{ for } b \in B^{\mathrm{p}}; && \mathrm{rank}(b) = (\varepsilon; \mathrm{o}), \text{ for } b \in B^{\mathrm{o}};\\
&\mathrm{rank}(_^{\mathrm{p}}) = (\mathrm{p}^+; \mathrm{p}); && \mathrm{rank}(_^{\mathrm{o}}) = (\mathrm{o}^+; \mathrm{o});\\
&\mathrm{rank}(\wedge^{\mathrm{p}}) = (\mathrm{p}^+; \mathrm{p}); && \mathrm{rank}(\wedge^{\mathrm{o}}) = (\mathrm{o}^+; \mathrm{o});\\
&\mathrm{rank}(\mathrm{c}^{\mathrm{p}}) = (\mathrm{p}\,\mathrm{o}; \mathrm{p}); && \mathrm{rank}(\mathrm{c}^{\mathrm{o}}) = (\mathrm{o}\,\mathrm{p}; \mathrm{o}).
\end{aligned}$$

Given $F \in \mathcal{F}$ and $s \in S$, we say that F is of type s, if $\mathrm{out}(F) = s$. The elements of B^{p} and B^{o} are typed constants, which represent basic actions of the proponent's and opponent's type, respectively. By B we denote the union $B^{\mathrm{p}} \cup B^{\mathrm{o}}$. The functions[1] $\_^{\mathrm{p}}; \wedge^{\mathrm{p}}; \_^{\mathrm{o}}$, and $\wedge^{\mathrm{o}}$ represent disjunctive and conjunctive refinement operators for the proponent and the opponent, respectively. We set $\overline{\mathrm{p}} = \mathrm{o}$ and $\overline{\mathrm{o}} = \mathrm{p}$. The binary functions c^s, for $s \in S$, represent countermeasures and are used to connect components of type s with components of the opposite type $\overline{s}$.

Definition 2. Typed ground terms over the AD signature are called attack defense terms (ADTerms). The set of all ADTerms is denoted by T.

For $s \in \{\mathrm{p}; \mathrm{o}\}$, we denote by T^s the set of all ADTerms with the head symbol of type s. We have $T = T^{\mathrm{p}} \cup T^{\mathrm{o}}$. The elements of T^{p} and T^{o} are called ADTerms of the proponent's and of the opponent's type, respectively. The ADTerms of the proponent's type constitute formal representations of ADTrees.

Example 2. Consider the ADTree given in Figure 1. The corresponding ADTerm is

$$t = \_^{\mathrm{p}}(\wedge^{\mathrm{p}}(\mathrm{IC}; \mathrm{UC}); \mathrm{SS}; \mathrm{c}^{\mathrm{p}}(\mathrm{OA}; \mathrm{FW})).$$

The entire ADTerm, as well as its six subterms $\wedge^{\mathrm{p}}(\mathrm{IC}; \mathrm{UC})$, $\mathrm{c}^{\mathrm{p}}(\mathrm{OA}; \mathrm{FW})$, IC, UC, SS, and OA, are of the proponent's type. Term t also contains a subterm of the opponent's type, namely FW.

In order to facilitate and automate quantitative analysis of vulnerability scenarios, the notion of an attribute for ADTerms has been formalized in [15]. An attribute expresses a particular property, quality, or characteristic of a scenario, such as the minimal costs of an attack or the expected impact of a defensive measure. A specific bottom-up procedure for evaluation of attribute values on ADTerms ensures that the user, for instance a security expert, only needs to quantify the basic actions. From these, the value for the entire scenario is deduced automatically. Attributes are formally modeled using attribute domains.

Definition 3. An attribute domain for ADTerms is a tuple

$$A = (D; \_^{\mathrm{p}}; \wedge^{\mathrm{p}}; \_^{\mathrm{o}}; \wedge^{\mathrm{o}}; \mathrm{c}^{\mathrm{p}}; \mathrm{c}^{\mathrm{o}}),$$

where D is a set of values and, for $s \in \{\mathrm{p}; \mathrm{o}\}$,

- $\_^s$, $\wedge^s$ are unranked operations on D,
- c^s are binary operations on D.

Let $A = (D; \_^{\mathrm{p}}; \wedge^{\mathrm{p}}; \_^{\mathrm{o}}; \wedge^{\mathrm{o}}; \mathrm{c}^{\mathrm{p}}; \mathrm{c}^{\mathrm{o}})$ be an attribute domain for ADTerms. The bottom-up computation of attribute values on ADTerms is formalized as follows. First, a value from D is assigned to each basic action, with the help of function $\alpha : B \to D$, called a basic assignment. Then, a recursively defined function $\hat{\alpha} : T \to D$ assigns a value to every ADTerm t, as follows

$$\hat{\alpha}(t) = \begin{cases} \alpha(t); & \text{if } t \in B,\\ \_^s(\hat{\alpha}(t_1); \ldots; \hat{\alpha}(t_k)); & \text{if } t = \_^s(t_1; \ldots; t_k);\\ \wedge^s(\hat{\alpha}(t_1); \ldots; \hat{\alpha}(t_k)); & \text{if } t = \wedge^s(t_1; \ldots; t_k);\\ \mathrm{c}^s(\hat{\alpha}(t_1); \hat{\alpha}(t_2)); & \text{if } t = \mathrm{c}^s(t_1; t_2); \end{cases} \tag{1}$$

where $s \in \{\mathrm{p}; \mathrm{o}\}$ and $k > 0$.

The example below illustrates the bottom-up procedure for an attribute called satisfiability.

[1] In fact, symbols $\_^{\mathrm{p}}; \wedge^{\mathrm{p}}; \_^{\mathrm{o}}$, and $\wedge^{\mathrm{o}}$ represent unranked functions, i.e., they stand for families of functions $(\_^{\mathrm{p}}_k)_{k \in \mathbb{N}}; (\wedge^{\mathrm{p}}_k)_{k \in \mathbb{N}}; (\_^{\mathrm{o}}_k)_{k \in \mathbb{N}}; (\wedge^{\mathrm{o}}_k)_{k \in \mathbb{N}}$.

Example 3. The question "Is the considered scenario satisfiable?" is formally modeled using the satisfiability attribute. The corresponding attribute domain is $A_{sat} = (\{0,1\}, \wedge, \vee, \neg, \bot, \top)$, where $\neg(x) = x^{\vee} = y$, for all $x, y \in \{0,1\}$. The basic assignment $\beta_{sat} : B \to \{0,1\}$ assigns the value 1 to every basic action which is satisfiable and the value 0 to every basic action which is not satisfiable. Using the recursive evaluation procedure defined by Equation (1), we evaluate the satisfiability attribute on the ADTerm from Example 2. Assuming that all basic actions are satisfied, i.e., that $\beta_{sat}(X) = 1$ for $X \in \{IC, UC, SS, OA, FW\}$, we obtain

$$\begin{aligned} &sat(\vee^{p}(\wedge^{p}(IC, UC), SS, c^{p}(OA, FW))) = \\ &\quad \vee(\wedge(\beta_{sat}(IC), \beta_{sat}(UC)), \beta_{sat}(SS), \neg(\beta_{sat}(OA), \beta_{sat}(FW))) = \\ &\quad \vee(\wedge(1,1), 1, \neg(1,1)) = \vee(1,1,0) = 1. \end{aligned}$$

The satisfiability attribute, as introduced in the previous example, allows us to define which player is the winner of the considered attack defense scenario. If the satisfiability value calculated for an ADTerm is equal to 1, the winner of the corresponding scenario is the proponent, otherwise the winner is the opponent. In Example 3, the root attack is satisfied, so the winner is the attacker.

3 Classification of Questions

One of the goals of this paper is to describe how to correctly specify a question for an ADTree. This allows us to construct the corresponding formal model and deduce an answer using the bottom-up procedure. Let us motivate our approach with the following example.

Example 4. "What are the costs of the considered scenario?" seems to be a valid question on an ADTree. However, this question is underspecified, because we do not know whether we should quantify the attacker's costs, the defender's costs or both. Clarifying this information is necessary to correctly define the corresponding basic assignment. We improve the question and ask "What are the costs of the attacker?" The new question is still underspecified, since it is not clear whether we are interested in the minimal, maximal, average or other costs. Making also this information explicit is necessary to correctly define the way how to aggregate the values for disjunctively refined nodes of the attacker.

In this paper, we provide a pragmatic taxonomy of quantitative questions that can be asked about ADTrees. The presented classification results from case studies, e.g., [5,10,27], as well as from a detailed literature overview concerning quantitative analysis of security. Our study allowed us to identify three main classes of empirical questions, as described below.

Class 1: Questions referring to one player. Most of the typical questions for ADTrees have an explicit or implicit reference to one of the players which we call owner of the question. This is motivated by the fact that the security model is usually analyzed from the point of view of one player only. Examples of questions referring to one player are "What are the minimal costs of the attacker?" (here the owner is the attacker) or "How much does it cost to protect the system?" (here implicitly mentioned owner is the defender). When we ask a question of Class 1, we assume that its owner does not have extensive information concerning his adversary. Thus, we always consider the worst case scenario with respect to the actions of the other player. Most of the questions usually asked for attack trees can be adapted so that they can be answered on ADTrees as well. Thus, questions related to attributes such as attacker's/defender's costs [26,7,27,4,25,21,31,1,24,8,2,29,10], attack/defense time [12,26,29], attack detectability [27,8], attacker's special skill [21,1,26], difficulty of attack/protection [8,11,27,12,21,1,3,29], penalty [7,13,29], impact of the attack [26,27,12,19,25,21,3,1,23,10,29], attacker's profit [3,13,6,24], etc., all belong to Class 1. We analyze questions of this class in Section 4.

Class 2: Questions where answers for both players can be deduced from each other. Exemplary questions belonging to Class 2 are "Is the scenario satisfiable?", or "How probable is it that the scenario will succeed?". We observe that if the scenario is satisfied for the attacker, then it is not satisfied for the defender, and vice versa. Similarly, knowing that one player succeeds with probability p, we also know that the other player succeeds with probability $1 - p$. The foremost goal of attack trees and all their extensions is

to represent whether attacks are possible. Thus, the *satisfiability* attribute is considered, either explicitly or implicitly, in all works concerning attack trees and their extensions. As for *probability*[2], the attribute has been extensively studied in [6,7,12,19,20,31,1,24,8,10,29]. We perform a detailed analysis of questions of Class 2 in Section 5.

Class 3: Questions referring to an outside third party. Questions belonging to Class 3 relate to a universal property which is influenced by actions of both the attacker and the defender. They quantify attack defense scenarios from the point of view of an outside third party which is neither the attacker nor the defender. For instance, one could ask about "How much *data traffic is involved in the attack–defense scenario*?". In this case, we do not need to distinguish between traffic resulting from the attacker's and the defender's actions, as both players contribute to the total amount. Another example of a question of Class 3 is "What is the *global environmental impact* of the scenario?". Instances of environmental impact could be CO_2 emission or water pollution. Attributes corresponding to questions in Class 3 have not been addressed in the attack tree literature, since attack trees focus on a single player. The importance of those questions becomes apparent when actions of two opposing parties are considered. The case study [5] that we have performed using the attack defense tree methodology showed that such attributes relate to essential properties which should not be disregarded by the security assessment process. Questions of Class 3 are discussed in Section 6.

The following three sections set up guidelines for how to correctly specify quantitative questions of all three classes. The guidelines' main purpose is to enable us to find a corresponding attribute domain in order to correctly compute an answer using the bottom-up procedure. Figure 2 depicts the three classes of questions, as well as general templates for the corresponding attribute domains, as introduced in Definition 3. Symbols $\heartsuit$, $\diamondsuit$ and $\bar{\heartsuit}$ serve as placeholders for specific operators. Corresponding symbols within a tuple indicate that the functions coincide. For instance, $(D, \heartsuit, \diamondsuit, \heartsuit, \diamondsuit, \heartsuit, \diamondsuit)$ means that $\heartsuit^p = \diamondsuit^o = c^o$ and that $\diamondsuit^p = \heartsuit^o = c^p$. We motivate these equalities and give possible instantiations of $\heartsuit$, $\diamondsuit$ and $\bar{\heartsuit}$ in the following three sections.

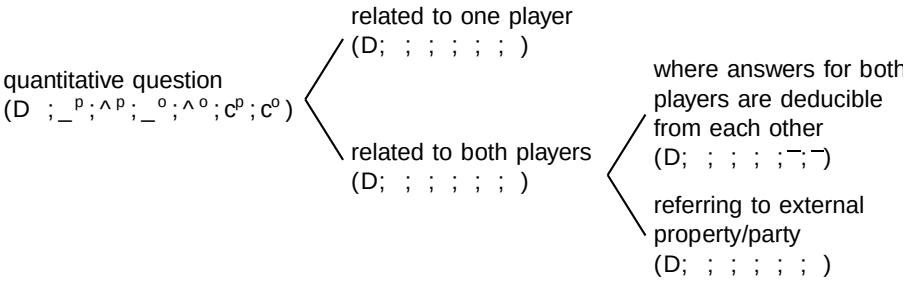

Figure 2. Classification of questions and attribute domains' templates

4 Questions Referring to One Player

4.1 Defining a Formal Model for Questions of Class 1

Questions belonging to Class 1 refer to exactly one player, which we call the question's *owner*. As we explain below, in the attack defense tree setting, only two situations occur for a question's owner: either he needs to choose *at least one* option or he needs to execute *all* options. Therefore, two operators suffice to answer questions of Class 1 and the generic attribute domain is of the form $(D, \heartsuit, \diamondsuit, \heartsuit, \diamondsuit, \heartsuit, \diamondsuit)$. Furthermore, if we change a question's owner, the attribute domain changes from $(D, \heartsuit, \diamondsuit, \heartsuit, \diamondsuit, \heartsuit, \diamondsuit)$ into $(D, \diamondsuit, \heartsuit, \diamondsuit, \heartsuit, \diamondsuit, \heartsuit)$.

We illustrate the construction of the formal model for Class 1 using the question "What are the minimal costs of the attacker?", where the owner is the attacker. In the case of Class 1, all values assigned

[2] We would like to point out that, the probability attribute can only be evaluated using the bottom-up procedure given by Equation (1), if the ADTree does not contain any dependent actions.

to nodes and subtrees express the property that we are interested in from the perspective of the question's owner. In the minimal costs example, this means that even subtrees rooted in defense nodes have to be quantified from the attacker's point of view, i.e., a value assigned to the root of a subtree expresses what is the minimal amount of money that the attacker needs to invest in order to be successful in the current subtree.

Subtrees rooted in uncountered attacker's nodes can either be disjunctively or conjunctively refined. In the first case the attacker needs to ensure that he is successful in at least one of the refining nodes, in the second case he needs to be successful in all refining nodes. The situation for subtrees rooted in uncountered defender's nodes is reciprocal. If a defender node is disjunctively refined, the attacker needs to successfully counteract all possible defenses to ensure that he is successful at the subtree's root node; if the defender's node is conjunctively refined, successfully counteracting at least one of the refining nodes already suffices for the attacker to be successful at the subtree's root node.

This reciprocality explains that two different operators suffice to quantify all possible uncountered trees: The operator that we use to combine attribute values for disjunctively refined nodes of one player is the same as the operator we use for conjunctively refined nodes of the other player.

Furthermore, the same two operators can also be used to quantify all remaining subtrees. If a subtree is rooted in a countered attacker's node, the attacker needs to ensure that he is successful at the action represented by the root node and that he successfully counteracts the existing defensive measure. Dually, for the attacker to be successful in a subtree rooted in a defender's countered node, it is sufficient to successfully overcome the defensive action or to successfully perform the attack represented by the countering node. This implies that we can use the same operator as for conjunctively refined attacker's nodes in the first case and the same operator as for disjunctively refined attacker's nodes in the second case.

4.2 Pruning

For attributes in Class 1, we are only interested in one player, the owner of a question. Therefore for this class, we should disregard subtrees that do not lead to a successful scenario for the owner. We achieve this with the help of the pruning procedure illustrated in the following example.

Example 5. Consider the ADTree in Figure 1 and assume that we are interested in calculating the minimal costs of the attacker. In this case, there is no need to consider the subtree rooted in Outsider Attack, because it is countered by the defense Firewall and thus does not lead to a successful attack. The subtree rooted in Outsider Attack therefore should be removed. This simultaneously eliminates having to provide values for the non-refined nodes Outsider Attack and `Firewall. The computation of the minimal costs is then executed on the term corresponding to the tree in the right of Figure 3.

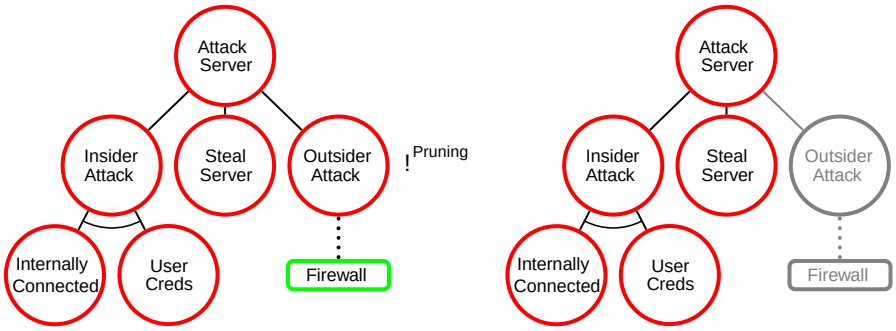

Figure 3. Pruning the attack server scenario for questions of Class 1 owned by the attacker

To motivate the use of the pruning procedure, let us distinguish two situations. If a non-refined node of the non-owner is countered, its assigned value should not influence the result of the computation. If

a non-owner's node is not countered, its value should indicate that the owner does not have a chance to successfully perform this subscenario. Mathematically, this means that the value assigned to the non-refined nodes of the non-owner needs to be neutral with respect to one operator and simultaneously absorbing with respect to the other. Since, in general, such an element may not exist, we use pruning to eliminate one of the described situations. which results in elimination of the absorption condition.

Below we explain how to intuitively prune an ADTree and how to model the pruning in a mathematical way.

Pruning intuitively. Let us consider a question of Class 1 and its owner. In order to graphically prune an ADTree, we perform the following procedure. Starting from a leaf of the non-owner, we traverse the tree towards the root until we reach the first node v satisfying one of the following conditions, as illustrated in Figures 4, 5, 6, and 7.

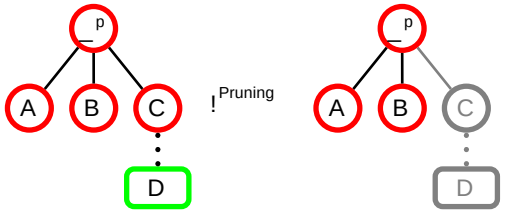

Figure 4. Pruning a proper disjunctive refinement

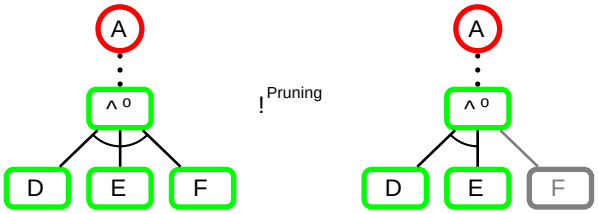

Figure 5. Pruning a proper conjunctive refinement

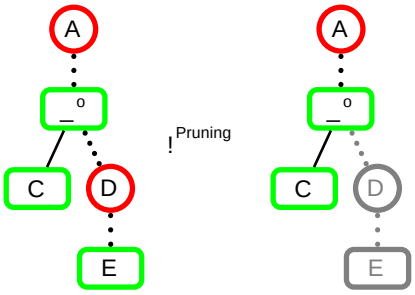

Figure 6. Pruning a countermeasure

- v is a node of the owner and part of a proper[3] disjunctive refinement (see Figure 3);
- v is a node of the non-owner and part of a proper conjunctive refinement (see Figure 5);

[3] A refinement is called proper if it contains at least two refining nodes.

Figure 7. Pruning an entire ADTree

v is a node of the owner that counteracts a refined node of the non-owner (see Figure 6);
v is the root of the ADTree (see Figure 7).

The subtree rooted in node v is removed from the ADTree. The procedure is repeated, starting from all leaves of the non-owner. We note that the order in which we perform the procedure does not influence the final result. Also, in some cases the pruning procedure results in the removal of the entire ADTree. This is the case when the owner of the question does not have any way of successfully achieving his goal.

Pruning formally. Let Q be a question of Class 1 and let own denotes the owner of Q. In order to model the pruning procedure in a mathematical way, we construct the formal model answering the question Can the owner of Q succeed in a considered attack defense scenario? The idea is to assign the Boolean value 1 to all subtrees in which the owner of Q can succeed and the value 0 to the subtrees in which he cannot. Formally, we evaluate an attribute that we denote by sat_{own}, defined as follows. First we set the basic assignment

$$sat_{own}(b) = \begin{cases} 1 & \text{if } b \in B^{own} \\ 0 & \text{if } b \in B^{\overline{own}}. \end{cases} \tag{2}$$

Then, given an ADTerm t, we use the following attribute domain[4] to derive the values of the attribute sat_{own} at all subterms of t:

$$A_{sat_{own}} = \begin{cases} (\{0,1\}, \wedge, \vee, \wedge, \vee, \wedge, \vee) & \text{if own} = p \\ (\{0,1\}, \vee, \wedge, \vee, \wedge, \vee, \wedge) & \text{if own} = o. \end{cases} \tag{3}$$

The following theorem shows that sat_{own} models the pruning procedure soundly and correctly.

Theorem 1. Consider a question Q of Class 1, its owner own, an ADTree T and the corresponding ADTerm t. Furthermore, let $A_{sat_{own}}$ and sat_{own} be defined by equations (2) and (3). The intuitive pruning procedure presented in Section 4.2 removes a subtree T' of T if and only if the evaluation of the sat_{own} attribute on the corresponding subterm t' of t results in the value 0.

Proof. We need to show that

1. if a subtree is removed by pruning, the evaluation of sat_{own} on the corresponding term results in 0.
2. if a subtree is not removed by pruning, evaluation of sat_{own} on the corresponding term results in 1.

1) Let u be a leaf of the non-owner, from which we start the current step of the pruning procedure. We show that if a tree rooted in a node v is removed by pruning, then all subterms corresponding to subtrees rooted in the nodes on the path from u to v (including u and v) evaluate to 0.

We prove by contraposition. Assume that there exists a node w on the path between u and v, such that the term corresponding to the tree rooted in w evaluates to 1. Moreover, let w be the first node with such property encountered when starting from u. Note that $w \neq u$, because the basic assignment sat_{own} assigns the value 0 to every non-refined node of the non-owner. This means that there exists a node w_1 which is a child of w lying on the path from u to v. According to our assumptions, the term corresponding to the tree rooted in w evaluates to 1 while the term corresponding to the subtree rooted in w_1 evaluates to 0. This implies that operator $\wedge$ has been used. According to the attribute domain given by (3), there are only three situations where the logical disjunction is used:

[4] Note that the question Can the owner of Q succeed in the scenario? also falls into Class 1, as it is referring to a specific player. This explains why the corresponding attribute domain conforms to the template deduced in Section 4.1.

either w is a properly, disjunctively re ned node of the owner;
or w is a properly, conjunctively re ned node of the non-owner;
or w is a re ned node of the non-owner and w_1 is its countermeasure.

It is now su cient to notice that in all the three cases, the pr uning procedure should have had stopped at node w_1. Contradiction.

2) First, let us remark that the pruning procedure stops at node v if the value of the term corresponding to the tree rooted in the parent node of v is not uniquely determined by the value of the subterm corresponding to the tree rooted in v. This is because, in all three cases where pruning stops at v, the calculation of the sat_{own} attribute for the subterm corresponding to the tree rooted in the parent of v uses operator _ which is applied to the value 0 (quantifying the term corresponding to the tree rooted in v)

constitutes a formal model allowing us to correctly evaluate attribute , and thus answer Q, without requiring any prior pruning.

Theorem 2. Let Q, A , , A and c be as defined in this section. For every ADTerm t, we have

b(t) = (d; 0), for some d 2 D, if the tree corresponding to t is removed by the pruning procedure related to Q;
b(t) = ((t); 1), if the tree corresponding to t is not removed by the pruning procedure related to Q.

Proof. First observe that the calculation of the second component of the pair b(t) corresponds to the calculation of attribute sat_{own} formalized in Section 4.2. Theorem 1 ensures that the second component of b(t) is 0 if and only if the corresponding ADTerm is removed by the pruning procedure related to Q. In this case, the first component of b(t) does not have any conclusive meaning. This corresponds to the fact that answering the question Q for the pruned subtrees of an ADTree does not make any sense, since these subtrees do not contribute to the success of the owner of Q.

Let t be a term corresponding to a subtree which is not removed by pruning related to Q. In order to prove that b(t) = ((t); 1), it is sufficient to notice that, according to Theorem 1, the evaluation of sat_{own} on all subterms of t results in the value 1. This means that, when calculating b(t) we perform operations of the form

$$\begin{aligned} b((d_1; 1); : : : ; (d_k; 1)) &= ((d_1 \quad 1; : : : ; d_k \quad 1); 1) \\ &= ((d_1; : : : ; d_k); 1); \end{aligned}$$

where 2 f ; g. This obviously leads to the desired result. tu

We illustrate the use of the extended attribute domain introduced in this section on the following example.

Example 6. As in Example 7, we would like to answer the question What are the minimal costs of the proponent, assuming that reusing tools is infeasible? , on the tree in the left of Figure 3. From Example 7, we know that the corresponding attribute domain is $A_{co} = (R; min; +; +; min; +; min)$. We extend A_{co} to the attribute domain $A_{co} = (R; min; +; +; min; +; min)$, as defined in this section. Since + 1 is the neutral element with respect to min on R, operation is defined as x 0 = + 1 , for every x 2 R. We evaluate the minimal costs attribute on the ADTerm corresponding to the non-pruned ADTree from Figure 3, as follows:

$$\begin{aligned} &co(\_^p(\wedge^p(IC; UC); SS; c^p(OA; FW))) = \\ &min(+(c_{co}(IC); (c_{co}(UC))); c_{co}(SS); +(c_{co}(OA); c_{co}(FW))) = \\ &min(+((\ _{co}(IC); 1); (\ _{co}(UC); 1)); (\ _{co}(SS); 1); +((\ _{co}(OA); 1); (\ _{co}(FW); 0))) = \\ &min((+(\ _{co}(IC); \ _{co}(UC)); 1); (\ _{co}(SS); 1); (+(\ _{co}(OA); + 1); 0)) = \\ &min((+(100\,e; 200\,e); 1); (400\,e; 1); (+ 1 ; 0)) = \\ &min((300\,e; 1); (400\,e; 1); (+ 1 ; 0)) = \\ &(min f 300\,e; 400\,e; + 1g ; 1) = \\ &(300\,e; 1): \end{aligned}$$

This result shows that the scenario is satisfiable for the proponent and that his minimal costs are 300e. It is the same as the result obtained in Example 7.

4.4 From a Question to an Attribute Domain

In this section we analyze how a question of Class 1 should look like, in order to be able to instantiate the attribute domain template A = (D; ; ; ; ; ;) with specific value set and operators. To correctly instantiate A, we need a value domain D, two operators (for all and at least one) and we need to know

which of those operators instantiates and which . Thus, a well specified question of Class 1 contains exactly four parts, as illustrated on the following question:

Modality: What are the minimal
Notion: costs
Owner: of the proponent
Execution: assuming that all actions are executed one after another?

Each of the four parts has a specific purpose in determining the attribute domain.

Notion. The notion used by the question influences the choice of the value domain. The notions in Class 1, identified during our study, are:

attack potential,	impact,	resources,
attack time,	insider required,	severity,
consequence,	mitigation success,	skill level,
costs,	outcome,	special equipment
detectability,	penalty,	needed,
difficulty level,	profit,	special skill needed,
elapsed time,	response time,	survivability.

From the notion we determine the value domain, e.g., $\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}_0$, etc. The choice of the value domain influences the basic assignments, as well as the operators to be determined by the modality and the execution style. The selected value domain needs to include all values that we want to use to quantify the owner's actions. It also must contain a neutral element with respect to , if own = p, and with respect to , if own = o. This neutral element is assigned to all non-refined nodes of the non-owner, as argued in Section 4.2.

Modality. The modality of a question clarifies how options are treated. Thus, it determines the characteristic of the *at least one* operator. Different notions are accompanied with different modalities. In the case of costs, interesting modalities are minimal, maximal and average.

Execution. The question also needs to specify an execution style. Its value determines the treatment when all actions need to be executed. Thus, it describes the characteristic of the *all* operator. Exemplary execution styles are: simultaneously/sequentially (for time) or with reuse/without reuse (for resources).

Owner. The owner of a question determines how the modality and the execution are mapped to and . In case the owner of the question is the root player, i.e., the proponent, is instantiated with the *at least one* operator and with the *all* operator. In case the root player is not the owner, the instantiations are reciprocal.

Given all four parts, we can then construct the appropriate attribute domain. For the notion of continuous time, also called duration, possible combinations of the modality, the execution style and the owner have been determined in Table 1. We instantiate the attribute domain template $(D, \ , \ , \ , \ , \ )$ with the elements of the algebraic structure $(D, \ , \ )$, and use the value indicated in the last column of the table as the basic assignment for all non-refined nodes of the non-owner. The table can be used in the case of other notions as well, as shown in the next example

Example 7. The question What are the minimal costs of the proponent, assuming that reusing tools is infeasible? can be answered using the attribute domain $\mathbb{A}_{co} = (\mathbb{R}, \min, +, +, \min, +, \min)$. Here the notion is *cost*, which has the same value domain as duration, i.e., $\mathbb{R}$. The modality is *minimum*, the owner is *the proponent* and the execution style is *without reuse*, which corresponds to sequential. Hence, we use the structure $(\mathbb{R}, \min, +)$, as specified in Line 1 of Table 1. In order to answer the question on the tree in the left of Figure 3, we first prune it, as shown on the right of Figure 3. The only basic actions that are left are Internally connected, User Creds and Steal Server. Suppose the costs are 100e, 200e, and 400e, respectively. We use those values as basic assignment α_{co} and apply the bottom-up computation to the ADTerm $\wedge^p(\vee^p(IC, UC), SS)$:

$$co(\wedge^p(\vee^p(IC, UC), SS)) = \wedge^p_{co}(\vee^p_{co}(\alpha_{co}(IC), \alpha_{co}(UC)), \alpha_{co}(SS)) = \min\{+(100e, 200e), 400e\} = 300e.$$

	Notion	Modality	Owner	Execution	Structure $(D, \sqcap, \sqcup)$	Basic assignment for $\overline{own}$
1	duration	min	p	sequential	$(\mathbb{R}, \min, +)$	$+\infty$
2	duration	avg	p	sequential	$(\mathbb{R}, avg, +)$	e_{avg}
3	duration	max	p	sequential	$(\mathbb{R}, \max, +)$	$-\infty$
4	duration	min	o	sequential	$(\mathbb{R}, +, \min)$	0
5	duration	avg	o	sequential	$(\mathbb{R}, +, avg)$	0
6	duration	max	o	sequential	$(\mathbb{R}, +, \max)$	0
7	duration	min	p	parallel	$(\mathbb{R}, \min, \max)$	$+\infty$
8	duration	avg	p	parallel	$(\mathbb{R}, avg, \max)$	e_{avg}
9	duration	max	p	parallel	$(\mathbb{R}, \max, \max)$	$-\infty$
10	duration	min	o	parallel	$(\mathbb{R}, \max, \min)$	$-\infty$
11	duration	avg	o	parallel	$(\mathbb{R}, \max, avg)$	$-\infty$
12	duration	max	o	parallel	$(\mathbb{R}, \max, \max)$	$-\infty$

Table 1. Determining instantiation of the structure in Class 1, where e_{avg} denotes the neutral element with respect to avg.

We would like to remark that if the structure $(D, \sqcap, \sqcup)$ forms a semi-ring, it is not necessary to prune the ADTree to correctly answer a question Q of Class 1. This is due to the fact that in a semi-ring the neutral element[5] for the first operator is at the same time absorbing for the second operator. Such element can then be assigned to all subtrees which do not yield a successful scenario for the owner of Q, in particular to the uncountered basic actions of the non-owner.

5 Questions Where Answers for Both Players Can Be Deduced From Each Other

We illustrate the construction of the attribute domain for Class 2 using the question *What is the success probability of a scenario, assuming that all actions are independent?* In case of questions of Class 2, values assigned to a subtree quantify the considered property from the point of view of the root player of the subtree. This means that, if a subtree rooted in an attack node is assigned the value 0.2, the corresponding attack is successful with probability 0.2. If a subtree rooted in a defense node is assigned the value 0.2, the corresponding defensive measure is successful with probability 0.2. Thus, in Class 2, conjunctive and disjunctive refinements for the proponent and the opponent have to be treated in the same way: in both cases, they refer to the *at least one* option (here modeled with $\sqcup$) and the *all* options (modeled with $\sqcap$), of the player whose node is currently considered.

Questions in Class 2 have the property that, given a value for one player, we can immediately deduce a corresponding value for the other player. For example, if the attacker succeeds with probability 0.2 the defender succeeds with probability 0.8. This property is modeled using a value domain with a predefined unary negation operation $\overline{\ }$. Negation allows us to express the operators for both countermeasures using the *all* operator where the second argument is negated, which we represent by $\overline{\sqcap}$. Formally, $\overline{\sqcap}(x, y) = x \sqcap \overline{y}$. Hence attribute domains of Class 2 follow the template $(D, \sqcap, \sqcup, \overline{\ }, \overline{\sqcap})$.

Below we discuss three aspects that questions in Class 2 need to address.

Notion. Questions of Class 2 refer to notions for which the value domains contain a unary negation operation. This allows us to transform values of one player into values of the other player. Identified notions for Class 2 are:

feasibility,	probability of success,	needs electricity.
satisfiability,	probability of occurrence,	

Modality. Modality specifies the operator for at least one option. For the notions enumerated above, this will either be the logical OR $(\vee)$ or the probabilistic addition of independent events $P_{\vee}(A, B) = P(A) + P(B) - P(A)P(B)$, for a given probability distribution P and events A and B.

[5] Such an element is usually called zero of the semi-ring. For instance, $+\infty$ is the zero element of the semi-ring $(\mathbb{R}, \min, +)$.

Execution. Finally, we need to know what is the execution style, so that we can specify the operator for *all* options. In the above notions, this will either be the logical AND ($\wedge$) or the probabilistic multiplication of independent events $P_{\setminus}(A, B) = P(A)P(B)$.

Example 8. We calculate the success probability of the scenario given in Figure 1, assuming that all actions are independent. First we set the success probability of all basic actions to $_{pb} = 0.4$ and then we use the attribute domain $A_{pb} = ([0, 1], P_{[}, P_{\setminus}, P_{[}, P_{\setminus}, \overline{P_{\setminus}}, \overline{P_{\setminus}})$, where $\overline{P_{\setminus}}(A, B) = P_{\setminus}(A, \overline{B})$ to compute

$$P_{[}(P_{\setminus}(\ _{pb}(IC),\ _{pb}(UC)),\ (SS)), P_{\setminus}(\ _{pb}(OA), 1\ \ _{pb}(FW))) =$$
$$P_{[}(P_{\setminus}(0.4, 0.4), 0.4, P_{\setminus}(0.4, 1\ \ 0.4)) = P_{[}(0.16, 0.4, 0.24) = 0.61696$$

6 Questions Relating to an Outside Third Party

Suppose an outsider is interested in the overall maximal power consumption of the scenario. As in the previous section, disjunctive refinements of both players should be treated with one operator and conjunctive refinements of both players with another operator. Indeed, for a third party the important information is whether *all* or *at least one* option need to be executed and not who performs the actions. Also countermeasures lose their opposing aspect and their values are aggregated in the same way as conjunctive refinements. Regarding the question, this is plausible since both the countered and the countering action contribute to the overall power consumption. These observations result in the following template for an attribute domain in Class 3: $(D, , , , , ,)$.

We specify relevant parts of the questions in Class 3 on the following example.

Modality: What is the maximal
Notion: energy consumption
Execution: knowing that sharing of power is impossible?

Notion. In Class 3, we use notions that express universal properties covering both players. Found examples are:

social costs,	environmental costs,	combined execution time,
global costs,	environmental damage,	required network traffic,
third party costs,	information flow,	energy consumption.

Modality. The question should also contain enough information to allow us to specify how to deal with *at least one* option. In general, modalities used in Class 3 are the same as those in Class 1, e.g., minimal, maximal and average.

Execution. Finally, we need to know what is the execution style, so that we can define the correct operator for *all* options. The choices for execution style in Class 3 are again the same as in Class 1.

These three parts now straightforwardly define an algebraic structure $(D, ,)$ that we use to construct the attribute domain $(D, , , , , ,)$.

Example 9. Consider the question *What is the maximal energy consumption* for the scenario depicted in Figure 1, knowing that sharing of power is impossible? Both, the proponent's as well as the opponent's actions may require energy. We assume that being *Internally Connected*, performing an *Outsider Attack* and running a *Firewall* all consume 20kWh. Obtaining *User Creds* requires 1kWh, whereas *Stealing Server* does not require any energy. These numbers constitute the basic assignment for the considered attribute. From the question we know that, when we have a choice, we should consider the option which consumes the most energy. Furthermore, since sharing of power is impossible, values for actions which require execution of several subactions should be added. Thus, we use the attribute domain $A_{erg_{max}} = (\mathbb{R}, max, +, max, +, +, +)$ and compute the maximal possible energy consumption in the scenario as

$$erg_{max}((\_^{p}(\wedge^{p}(IC, UC)), SS)) =$$
$$max\{+(20\,kWh, 1kWh), 0kWh, +(20\,kWh, 20kWh)\} = 40\,kWh.$$

Due to similarities for modality and execution style for questions of Class 1 and Class 3, we can make use of Table 1, to choose the structure $(D, ,)$ that determines an attribute domain for a question of Class 3. The table corresponds to the case where the owner is the proponent.

7 Methodological Advancements for Attack Trees

ADTrees extend the well-known formalism of attack trees [26] by incorporating defensive measures to the model. Hence, every attack tree is in particular an ADTree. As visible in Example 4, underspecified questions are not a new phenomenon of ADTrees, but already occur in the case of pure attack trees. Thus, the formalization of quantitative questions, proposed in this paper, is not only useful in the attack defense tree methodology but, more importantly, it helps users of the more widely spread formalism of attack trees.

Given a well specified question on ADTrees and the corresponding attribute domain, we can answer the question on attack trees as well. Formally, attack trees are represented with terms involving only operators $\wedge^p$ and $\vee^p$. If $A = (D, \wedge^p, \vee^p, \wedge^o, \vee^o, c^p, c^o)$ is an attribute domain for ADTerms, the corresponding attribute domain for attack trees is $A = (D, \wedge^p, \vee^p)$, which corresponds to the formalization introduced in [21]. Furthermore, due to the fact that attack trees involve only one player (the attacker), the notions of attacker, proponent, and question's owner coincide in this simplified model. This in turn implies that, in the case of attack trees, the three classes of questions considered in this paper form in fact one class.

8 Prototype Tool

In order to automate the analysis of security scenarios using the attack defense methodology, we have developed a prototype software tool, called ADTool. It is written in Java and is compatible with multiple platforms (Windows, Linux, MAC OS). ADTool is publicly available [18]. Its main functionalities include the possibility of creation and modification of ADTree and ADTerm models as well as attributes evaluation on ADTrees.

ADTool combines the features offered by graphical tree representations with mathematical functionalities provided by ADTerms and attributes. The user can choose whether to work with intuitive ADTrees or with formal ADTerms. When one of these models is created or modified, the other one is generated automatically. The possibility of modular display of ADTrees makes ADTool suitable for dealing with large industrial case studies which may correspond to very complex scenarios and may require large models.

The software supports attribute evaluation on ADTrees, as presented in this paper. A number of predefined attribute domains allow the user to answer questions of Classes 1, 2 and 3. Implemented attributes include: costs, satisfiability, time and skill level, for various owners, modalities and execution styles; scenario's satisfiability and success probability; reachability of the root goal in less than x minutes, where x can be customized by the user; and the maximal energy consumption.

9 Conclusions

A useful feature of the attack defense tree methodology is that it combines an intuitive representation and algorithms with formal mathematical modeling. In practice we model attack defense scenarios in a graphical way and we ask intuitive questions about aspects and properties that we are interested in. To formally analyze the scenarios, we employ attack defense terms and attribute domains. In this paper, we have guided the user in how to properly formulate a quantitative question on an ADTree and how to then construct the corresponding attribute domain. Since attack trees are a subclass of attack defense trees, our results also advance the practical use of quantitative analysis of attack trees.

We are currently applying the approach presented in this paper to analyze socio-technical weaknesses of real-life scenarios, such as Internet web filtering, which involve trade offs between security and usability. In the future, we also plan to investigate the relation between attribute domains of all three classes and the problem of equivalent representations of the same scenario, formalized in [16].

Acknowledgments: We would like to thank Piotr Kordy for his contributions to the development of ADTool. This work was supported by the Fonds National de la Recherche Luxembourg under the grants C08/IS/26 and PHD-09-167.

References

1. Abdulla, P.A., Cederberg, J., Kaati, L.: Analyzing the Se curity in the GSM Radio Network Using Attack Jungles. In: Margaria, T., Ste en, B. (eds.) ISoLA (1). LNCS , vol. 6415, pp. 60 74. Springer (2010)
2. Amenaza: SecurITree. http://www.amenaza.com/ , accessed October 5, 2012
3. Amoroso, E.G.: Fundamentals of Computer Security Techno logy. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1994), http://portal.acm.org/citation.cfm?id=179237#
4. Baca, D., Petersen, K.: Prioritizing Countermeasures th rough the Countermeasure Method for Software Security (CM-Sec). In: Babar, M.A., Vierimaa, M., Oivo, M. (ed s.) PROFES. LNIBP, vol. 6156, pp. 176 190. Springer (2010)
5. Bagnato, A., Kordy, B., Meland, P.H., Schweitzer, P.: Att ribute Decoration of Attack Defense Trees. International Journal of Secure Software Engineering (IJSSE) 3(2), 1 35 (2012)
6. Bistarelli, S., Dall'Aglio, M., Peretti, P.: Strategic G ames on Defense Trees. In: Dimitrakos, T., Martinelli, F., Ryan, P.Y.A., Schneider, S.A. (eds.) FAST. LNCS, vol. 46 91, pp. 1 15. Springer (2006), http://www.springerlink.com/content/83115122h9007685/
7. Buldas, A., Laud, P., Priisalu, J., Saarepera, M., Willem son, J.: Rational Choice of Security Measures Via Multi-parameter Attack Trees. In: López, J. (ed.) CRITIS. L NCS, vol. 4347, pp. 235 248. Springer (2006)
8. Byres, E.J., Franz, M., Miller, D.: The Use of Attack Trees in Assessing Vulnerabilities in SCADA Systems. In: International Infrastructure Survivability Workshop (IISW'04), Institute of Electrical and Electronics Engineers, Lisbon (Dec 2004)
9. Dacier, M., Deswarte, Y.: Privilege graph: An extension t o the typed access matrix model. In: Gollmann, D. (ed.) ESORICS'94, LNCS, vol. 875, pp. 319 334. Springer Ber lin / Heidelberg (1994), http://dx.doi.org/10.1007/3-540-58618-0_72
10. Edge, K.S., Dalton II, G.C., Raines, R.A., Mills, R.F.: U sing Attack and Protection Trees to Analyze Threats and Defenses to Homeland Security. In: MILCOM. pp. 1 7. IEEE (2006)
11. Fung, C., Chen, Y.L., Wang, X., Lee, J., Tarquini, R., And erson, M., Linger, R.: Survivability analysis of distributed systems using attack tree methodology. In: Procee dings of the 2005 IEEE Military Communications Conference. vol. 1, pp. 583 589 (Oct 2005)
12. Henniger, O., Apvrille, L., Fuchs, A., Roudier, Y., Rudd le, A., Weyl, B.: Security requirements for automotive on-board networks. In: 9th International Conference on Int elligent Transport Systems Telecommunications (ITST'09). pp. 641 646. Lille (Oct 2009)
13. Jürgenson, A., Willemson, J.: Computing Exact Outcomes of Multi-parameter Attack Trees. In: Meersman, R., Tari, Z. (eds.) OTM Conferences (2). LNCS, vol. 5332, pp. 1036 1051. Springer (2008)
14. Kordy, B., Mauw, S., Melissen, M., Schweitzer, P.: Attac k Defense Trees and Two-Player Binary Zero-Sum Extensive Form Games Are Equivalent. In: Alpcan, T., Buttyá n, L., Baras, J.S. (eds.) GameSec. LNCS, vol. 6442, pp. 245 256. Springer (Nov 2010)
15. Kordy, B., Mauw, S., Radomirovi¢, S., Schweitzer, P.: Fo undations of Attack Defense Trees. In: Degano, P., Etalle, S., Guttman, J.D. (eds.) FAST. LNCS, vol. 6561, pp. 8 0 95. Springer (Sep 2010)
16. Kordy, B., Mauw, S., Radomirovi¢, S., Schweitzer, P.: At tack Defense Trees. Journal of Logic and Computation pp. 1 33 (2012), available online http://logcom.oxfordjournals.org/content/early/2012 /06/21/logcom.exs029.short?rss=1
17. Kordy, B., Pouly, M., Schweitzer, P.: Computational Asp ects of Attack Defense Trees. In: Security & Intelligent Information Systems. LNCS, vol. 7053, pp. 103 116. Springer (2011)
18. Kordy, P., Schweitzer, P.: The ADTool. http://satoss.uni.lu/members/piotr/adtool/index.php (2012), accessed October 12, 2012
19. Li, X., Liu, R., Feng, Z., He, K.: Threat modeling-orient ed attack path evaluating algorithm. Transactions of Tianjin University 15(3), 162 167 (2009), http://www.springerlink.com/content/v76g87255878721 4/
20. Manikas, T.W., Thornton, M.A., Feinstein, D.Y.: Using M ultiple-Valued Logic Decision Diagrams to Model System Threat Probabilities. In: 41st IEEE International S ymposium on Multiple-Valued Logic (ISMVL-11). pp. 263 267 (2011)
21. Mauw, S., Oostdijk, M.: Foundations of Attack Trees. In: Won, D., Kim, S. (eds.) ICISC. LNCS, vol. 3935, pp. 186 198. Springer (2005), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10 .1.1.97.1056
22. Pièvre-Cambacédès, L., Bouissou, M.: Beyond Attack Tre es: Dynamic Security Modeling with Boolean Logic Driven Markov Processes (BDMP). In: European Dependable Co mputing Conference. pp. 199 208. IEEE Computer Society, Los Alamitos, CA, USA (2010)
23. Roy, A., Kim, D.S., Trivedi, K.S.: Cyber security analys is using attack countermeasure trees. In: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research. pp. 28:1 28:4. CSIIRW '10, ACM, New York, NY, USA (2010), http://doi.acm.org.proxy.bnl.lu/10.1145/1852666.185 2698

24. Roy, A., Kim, D.S., Trivedi, K.S.: Attack countermeasur e trees (ACT): towards unifying the constructs of attack and defense trees. Security and Communication Networks 5(8), 929 943 (2012), http://dx.doi.org/10.1002/sec.299
25. Saini, V., Duan, Q., Paruchuri, V.: Threat Modeling Usin g Attack Trees. J. Computing Small Colleges 23(4), 124 131 (2008), http://portal.acm.org/citation.cfm?id=1352100
26. Schneier, B.: Attack Trees. Dr. Dobb's Journal of Softwa re Tools 24(12), 21 29 (1999), http://www.ddj.com/security/184414879
27. Tanu, E., Arreymbi, J.: An examination of the security im plications of the supervisory control and data acquisition (SCADA) system in a mobile networked environment : An augmented vulnerability tree approach. In: Proceedings of Advances in Computing and Technology, (A C&T) The School of Computing and Technology 5th Annual Conference. pp. 228 242. University of Ea st London, School of Computing, Information Technology and Engineering (2010), http://hdl.handle.net/10552/994
28. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F .: Fault Tree Handbook. Tech. Rep. NUREG-0492, U.S. Regulatory Commission (1981), http://www.nrc.gov/reading-rm/doc-collections/nureg s/staff/sr0492/sr0492.pdf
29. Wang, J., Whitley, J.N., Phan, R.C.W., Parish, D.J.: Uni ed Parametrizable Attack Tree. International Journal for Information Security Research 1(1), 20 26 (201 1), http://www.infonomics-society.org/IJISR/Unified%20Parametrizable%20Attack%20Tree.pdf
30. Willemson, J., Jürgenson, A.: Serial Model for Attack Tr ee Computations. In: Lee, D., Hong, S. (eds.) ICISC. LNCS, vol. 5984, pp. 118 128. Springer (2010), http://research.cyber.ee/~jan/publ/serialattack.pdf
31. Yager, R.R.: OWA trees and their role in security modelin g using attack trees. Inf. Sci. 176(20), 2933 2959 (2006)