Foundations of Attack–Defense Trees*

Barbara Kordy**, Sjouke Mauw, Saša Radomirović, Patrick Schweitzer***
{barbara.kordy, sjouke.mauw, sasa.radomirovic, patrick.schweitzer}@uni.lu

CSC and SnT, University of Luxembourg, 6, rue Coudenhove–Kalergi, L–1359 Luxembourg

Abstract. We introduce and give formal definitions of attack–defense trees. We argue that these trees are a simple, yet powerful tool to analyze complex security and privacy problems. Our formalization is generic in the sense that it supports different semantical approaches. We present several semantics for attack–defense trees along with usage scenarios, and we show how to evaluate attributes.

1 Introduction

It is a well-known fact that the security of any sufficiently valuable system is not static. In order to keep a system secure, it has to be defended against a growing number of attacks. As better defensive measures get deployed, more sophisticated attacks are developed, leading to an endless arms race and an increasingly complex system.

A mature, large, and complex system poses several challenges. How can it be decided whether a costly defensive measure implemented in the distant past is still necessary today? What are the best defensive measures worth currently investing in? How can newly discovered attacks and implemented defenses be efficiently and systematically documented?

These types of challenges are not new. Similar challenges had to be overcome by safety-critical systems. Moreover, the complexity of safety-critical systems in aviation and nuclear power plants easily rivals the most complex security applications. In the 1960s, fault tree analysis [1] was developed to evaluate the safety of systems. Since the 1990s, similar structures have been used to support system security engineering. One such structure are *attack trees*. The root of an attack tree corresponds to an attacker's goal. The children of a node in the tree are refinements of the node's goal into sub-goals. The leaves of the tree are the attacker's actions. Attack trees were popularized by Schneier [2] as a tool for evaluating the security of systems and subsequently formalized by Mauw and Oostdijk [3].

An obvious limitation of attack trees is that they cannot capture the interaction between attacks carried out on a system and the defenses put in place

* The original publication is available at: www.springerlink.com
** B. Kordy was supported by the grant No. C08/IS/26 from FNR Luxembourg
*** P. Schweitzer was supported by the grant No. PHD−09−167 from FNR Luxembourg

to fend off the attacks. This consequently limits the precision with which the best defensive strategies can be analyzed, since it does not take into account the effects of existing defensive measures which may have been overcome by new attacks. Similarly, a pure attack tree does not allow for the visualization and consideration of the *evolution* of a system's security, since the evolution can only be understood in view of both, the attacker's, as well as the defender's, actions.

These limitations can be overcome by introducing defensive actions as countermeasures to attacks. In order to model the ongoing arms race between attacks and defenses, it is necessary to allow for arbitrary alternation between these two types of actions.

We therefore introduce and formalize *attack–defense trees* (ADTrees) as a graphical representation of possible measures an attacker might take in order to attack a system and the defenses that a defender can employ to protect the system. Our formalization of ADTrees extends attack trees as defined in [3], in two ways. It introduces defenses as described above and it generalizes the interpretations and semantics of attack trees based on [3]. Consequently, our formalism provides a single framework covering the attributes and semantics of attack trees used in [4–6], including the notion of defense trees from [7, 8].

The main contribution of this paper is the development of a complete attack–defense language. Our design of this language includes an algebra of attack–defense terms (ADTerms), a graphical syntax (ADTrees), semantics derived from first order models, semantics derived from algebraic specifications, and a methodology to analyze properties of attack–defense terms (attributes). Aside from the specific semantics that we present here, several additional semantics are expected to be useful for the analysis of an attack–defense tree. Two such examples would be a temporal and a probabilistic semantics. The variety of semantics is a consequence of different interpretations of what an attack–defense tree represents. In the existing attack trees literature, several distinct interpretations have been made. The present approach accommodates all these interpretations rather than prescribing a single one.

Related Work. Fault trees have been introduced in the 1960s, and a formal framework was given by Vesely et al. [1]. They have been used by Moore et al. [9] to model attacks on systems, and by Cervesato and Meadows [10] to graphically represent attacks on security protocols. Amoroso [11] considered so-called threat trees on hospital computer systems and a simplified aircraft computer system. Schneier [2] used the fault tree approach to describe security vulnerabilities of systems coining the term *attack trees*. In [3], Mauw and Oostdijk have formalized attack trees.

Edge et al. [5] have shown how to compute the probability, cost, risk, and impact of an attacker's goal using attack trees. They have shown how to translate this information into *protection trees* in order to evaluate probability and cost for protection against the attacker's goal. Morais et al. [12] applied attack trees to test an implementation of WTLS, a wireless security protocol. Sheyner et al. [13]

have automatically generated and analyzed attack graphs from the output of a model checker for an intrusion detection system.

Willemson and Jürgenson [4] have extended the attack trees framework by introducing an order on the set of leaves which helps to solve the optimization problem of selecting the best attack given an attack tree. Bistarelli et al. [14] have generalized attack trees, considering defensive measures against basic actions. They have shown which measures to employ by using answer set programming, a form of declarative programming oriented towards difficult (primarily NP-hard) search problems.

In [15] it was shown that attack–defense trees and binary zero-sum two-player extensive form games can be converted into each other. Both formalisms have their advantages: attack–defense trees are more intuitive, because refinements can be explicitly modeled, whereas the game theoretical approach benefits from the well-studied methodology used in games.

Structure. The paper is organized as follows. In Section 2 we introduce attack–defense trees, give an example and define attack–defense terms which are a formal representation of attack–defense trees. Section 3 introduces semantics for attack–defense trees. In Section 4 we evaluate attributes on attack–defense trees.

2 Attack–Defense Trees

We start by defining the terminology used throughout this paper. Then we describe an example of an attack–defense scenario. We end this section by defining attack–defense terms.

2.1 Terminology

An ADTree is a node-labeled rooted tree 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 has nodes of two *opposite types*: *attack* nodes and *defense* nodes.

The two key features of an ADTree are the representation of *refinements* and *countermeasures*. Every node may have one or more children of the same type representing a *refinement* into *sub-goals* of the node's goal. If a node does not have any children of the same type, it is called a *non-refined* node. Non-refined nodes represent *basic actions*.

Every node may also have one child[1] of opposite type, representing a *countermeasure*. Thus, an attack node may have several children which refine the attack and one child which defends against the attack. The defending child in turn may have several children which refine the defense and one child that is an attack node and counters the defense.

[1] Note that allowing any number of children of opposite type leads to an equally expressive, but more complicated formalism.

The refinement of a node of an ADTree is 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.

The purpose of ADTrees is to analyze an attack–defense scenario. An attack–defense scenario is a game between two players, the *proponent* (denoted by p) and the *opponent* (denoted by o). The root of an ADTree represents the main goal of the proponent. When the root is an attack node, the proponent is an attacker and the opponent is a defender. Conversely, when the root is a defense node, the proponent is a defender and the opponent is an attacker.

In ADTrees, we depict attack nodes by circles and defense nodes by rectangles, as shown in Figure 1. Refinement relations are indicated by solid edges between nodes, and countermeasures are indicated by dotted edges. We depict a conjunctive refinement of a node by connecting the edges going from this node to its children of the same type with an arc.

2.2 Example

To demonstrate the features of ADTrees, we consider the following fictitious scenario covering a collection of attacks on bank accounts.

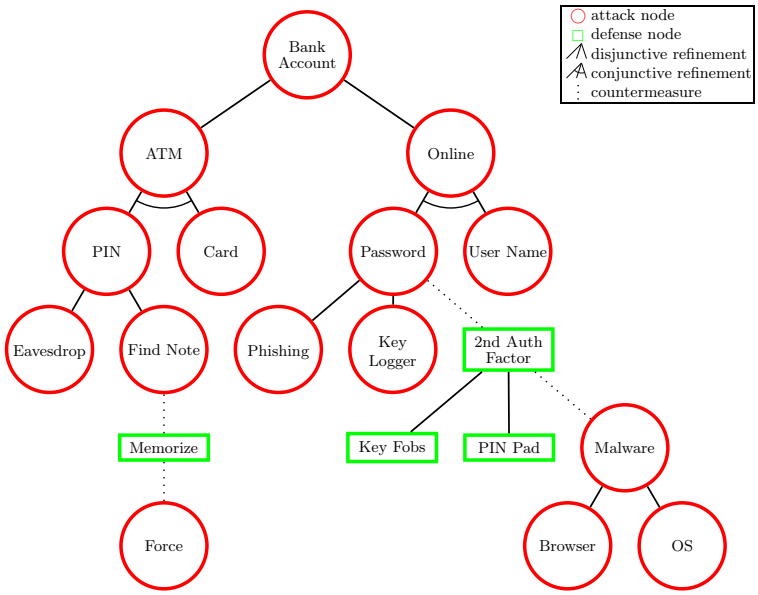


Fig. 1. Example of an ADTree: an attack on a bank account.

A bank wants to protect its customers' bank accounts from theft. Two forms of attacks on an individual's bank account are considered. An attacker can steal

money from the account either by attacking online or through an ATM. In order to steal money through an ATM, the attacker needs both, a PIN and a bank card. We ignore how an attacker might obtain a bank card and focus on the PIN. The PIN could be observed by eavesdropping on a customer while the customer types her PIN. Alternatively, the attacker might acquire a note on which the customer's PIN is written up. A simple defensive measure against exposing a note to an attacker is to memorize one's PIN. This defense is of little use if an attacker forces the customer to reveal the PIN.

For online banking attacks, an attacker needs to acquire a customer's online banking credentials, consisting of a user name and a password. While the user name can be retrieved easily, retrieving a user's password requires either a phishing email or a key logger. To defend against lost or stolen passwords, the bank introduces a second authentication factor. Two factor authentication can be implemented using key fobs which have a built-in cryptographic token with a pre-shared key known only to the token and the customer's bank. An alternative to key fobs are PIN pads. Two factor authentication, however, is useless, if the customer has malware on her computer. To install malware on a victim's computer, the attacker could attack either the browser or the operating system.

The ADTree representing the described state is shown in Figure 1. While this ADTree is obviously incomplete, it is also clear that it would be very simple to extend the ADTree with new attacks and defenses.

2.3 Formal Representation

In order to formally represent attack–defense scenarios, we define attack–defense terms. We make use of the notion of an unranked function. An *unranked function* f with domain D and range R denotes a family of functions $(f_k)_{k\in\mathbb{N}}$, where $f_k\colon D^k \to R$ for $k \in \mathbb{N}$. Given a set S, we denote by S^* the set of all strings over S, where ε stands for the empty string. We use $\dot\cup$ to denote a disjoint set union.

Definition 1. *An* AD–signature *is a pair $\Sigma = (S, F)$, such that*

- $S = \{\mathrm{p}, \mathrm{o}\}$ *is a set of types, (we set* $\overline{\mathrm{p}} = \mathrm{o}$ *and* $\overline{\mathrm{o}} = \mathrm{p}$*),*
- $F = \{(\vee^{\mathrm{p}}_k)_{k\in\mathbb{N}}, (\wedge^{\mathrm{p}}_k)_{k\in\mathbb{N}}, (\vee^{\mathrm{o}}_k)_{k\in\mathbb{N}}, (\wedge^{\mathrm{o}}_k)_{k\in\mathbb{N}}, \mathrm{c}^{\mathrm{p}}, \mathrm{c}^{\mathrm{o}}\} \mathbin{\dot\cup} \mathbb{B}^{\mathrm{p}} \mathbin{\dot\cup} \mathbb{B}^{\mathrm{o}}$ *is a set of function symbols, equipped with a mapping* $\mathrm{type}\colon F \to S^* \times S$*, which expresses the type of each function symbol, as follows. For every* $k \in \mathbb{N}$*,*

$$
\begin{aligned}
&\mathrm{type}(\vee^{\mathrm{p}}_k) = (\mathrm{p}^k, \mathrm{p}), && \mathrm{type}(\vee^{\mathrm{o}}_k) = (\mathrm{o}^k, \mathrm{o}),\\
&\mathrm{type}(\wedge^{\mathrm{p}}_k) = (\mathrm{p}^k, \mathrm{p}), && \mathrm{type}(\wedge^{\mathrm{o}}_k) = (\mathrm{o}^k, \mathrm{o}),\\
&\mathrm{type}(\mathrm{c}^{\mathrm{p}}) = (\mathrm{p}\,\mathrm{o}, \mathrm{p}), && \mathrm{type}(\mathrm{c}^{\mathrm{o}}) = (\mathrm{o}\,\mathrm{p}, \mathrm{o}),\\
&\mathrm{type}(b) = (\varepsilon, \mathrm{p}),\ \textit{for } b \in \mathbb{B}^{\mathrm{p}}, && \mathrm{type}(b) = (\varepsilon, \mathrm{o}),\ \textit{for } b \in \mathbb{B}^{\mathrm{o}}.
\end{aligned}
$$

The elements of $\mathbb{B}^{\mathrm{p}}$ and $\mathbb{B}^{\mathrm{o}}$ are typed constants, which we call proponent's (p) and opponent's (o) basic actions, respectively. By $\mathbb{B}$ we denote the union $\mathbb{B}^{\mathrm{p}} \dot\cup \mathbb{B}^{\mathrm{o}}$. The unranked functions $\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}, \vee^{\mathrm{o}}$, and $\wedge^{\mathrm{o}}$ represent disjunctive ($\vee$) and conjunctive ($\wedge$) refinement operators for a proponent and an opponent, respectively. The

binary function c^s, where $s \in S$, connects actions of type s with actions of the opposite type $\overline{s}$. Intuitively, $\mathrm{c}^s(a, d)$ expresses that there exists an action d (of type $\overline{s}$) that counteracts the action a (of type s).

By T_Σ we denote the set of all typed ground terms over the AD–signature Σ. The elements of T_Σ are called *attack–defense terms* (ADTerms). We have $T_\Sigma = T^{\mathrm{p}}_\Sigma \cup T^{\mathrm{o}}_\Sigma$, where T^{p}_Σ is the set of ADTerms of the proponent's type, and T^{o}_Σ is the set of ADTerms of the opponent's type. The terms from T^{p}_Σ constitute formal representations of ADTrees. Tables 1 and 2 show how to obtain the ADTerm corresponding to an ADTree, and vice versa. Given an ADTree T, we denote by $\iota(T)$ the ADTerm representing T. Given an ADTerm t, we denote by I(t) the corresponding ADTree. In Tables 1 and 2 we assume that the proponent is an attacker. If the proponent is a defender, circle nodes have to be replaced with square nodes and vice versa. To condense the presentation even further, we leave out the arcs, denoting conjunctions, in the cases where $f = \wedge^s$, for $s \in \{\mathrm{p}, \mathrm{o}\}$. Since the computations, as well as the internal structure of ADTrees, do not depend on the names of the refined sub-goals, we represent the refined nodes with the associated function symbols, only. Note that it is always possible to also decorate the refined nodes with intuitive labels.

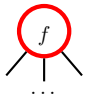
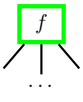
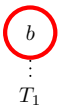
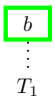
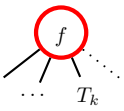
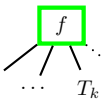
T				
	where $b \in \mathbb{B}^{\mathrm{p}}$	where $b \in \mathbb{B}^{\mathrm{o}}$	where $f \in \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}\}$, $k \geq 1$	where $f \in \{\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$, $k \geq 1$
$\iota(T)$	b	b	$f(\iota(T_1), \ldots, \iota(T_k))$	$f(\iota(T_1), \ldots, \iota(T_k))$
T				
	where $b \in \mathbb{B}^{\mathrm{p}}$	where $b \in \mathbb{B}^{\mathrm{o}}$	where $f \in \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}\}$, $k \geq 1$	where $f \in \{\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$, $k \geq 1$
$\iota(T)$	$\mathrm{c}^{\mathrm{p}}(b, \iota(T_1))$	$\mathrm{c}^{\mathrm{o}}(b, \iota(T_1))$	$\mathrm{c}^{\mathrm{p}}(f(\iota(T_1), \ldots, \iota(T_k)), \iota(T'))$	$\mathrm{c}^{\mathrm{o}}(f(\iota(T_1), \ldots, \iota(T_k)), \iota(T'))$

Table 1. Transformation from ADTrees to ADTerms

Example 1. The ADTerm representing the ADTree from Figure 1 is the following

$$\vee^{\mathrm{p}}\Big[\wedge^{\mathrm{p}}\Big(\vee^{\mathrm{p}}\Big(\mathrm{Eavesdrop}, \mathrm{c}^{\mathrm{p}}\big(\mathrm{FindNote}, \mathrm{c}^{\mathrm{o}}(\mathrm{Memorize}, \mathrm{Force})\big)\Big), \mathrm{Card}\Big),$$
$$\wedge^{\mathrm{p}}\Big(\mathrm{c}^{\mathrm{p}}\Big(\vee^{\mathrm{p}}(\mathrm{Phish}, \mathrm{KLog}), \mathrm{c}^{\mathrm{o}}\big(\vee^{\mathrm{o}}(\mathrm{KFob}, \mathrm{PPad}), \vee^{\mathrm{p}}(\mathrm{Br}, \mathrm{OS})\big)\Big), \mathrm{UName}\Big)\Big].$$

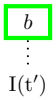
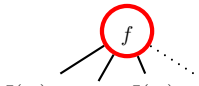
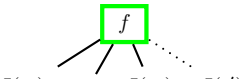
t I(t)	$b \in \mathbb{B}^{\mathrm{p}}$ 	$b \in \mathbb{B}^{\mathrm{o}}$ 	$f(t_1, \ldots, t_k)$, where $f \in \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}\}$, $k \geq 1$ 	$f(t_1, \ldots, t_k)$, where $f \in \{\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$, $k \geq 1$ 
t I(t)	$\mathrm{c}^{\mathrm{p}}(b, t')$, $b \in \mathbb{B}^{\mathrm{p}}$ 	$\mathrm{c}^{\mathrm{o}}(b, t')$, $b \in \mathbb{B}^{\mathrm{o}}$ 	$\mathrm{c}^{\mathrm{p}}(t_0, t')$, where $t_0 = f(t_1, \ldots, t_k)$ and $f \in \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}\}$, $k \geq 1$ 	$\mathrm{c}^{\mathrm{o}}(t_0, t')$, where $t_0 = f(t_1, \ldots, t_k)$ and $f \in \{\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$, $k \geq 1$ 
t I(t)			$\mathrm{c}^{\mathrm{p}}(t_0, t')$, where $t_0 = \mathrm{c}^{\mathrm{p}}(t_1, t_2)$ 	$\mathrm{c}^{\mathrm{o}}(t_0, t')$, where $t_0 = \mathrm{c}^{\mathrm{o}}(t_1, t_2)$ 

Table 2. Transformation from ADTerms to ADTrees

3 Semantics for Attack–Defense Terms

The main purpose of an ADTree is to specify and analyze an attack–defense scenario. It is possible that two distinct ADTrees describe the same scenario. In order to deal with such situations, we introduce the notion of equivalent ADTrees. For instance, the ADTrees represented by $\wedge^{\mathrm{p}}(a, b)$ and $\wedge^{\mathrm{p}}(b, a)$ are equivalent, if and only if the order of executing the sub-goals is irrelevant for the achievement of the parent goal. Hence, we consider ADTerms modulo an equivalence relation. This makes it possible to, e.g., transform an ADTerm into a simpler or more appealing form.

Definition 2. *A semantics for ADTerms is an equivalence relation on T_Σ that preserves types.*

3.1 Models for Attack–Defense Terms

Several different approaches, like propositional logics or multiset interpretations, were proposed in the literature to define semantics for attack trees [4, 16, 3]. In this section, we extend them to attack–defense trees. This is achieved with the help of first order models, cf. [17].

Definition 3. *Consider the AD–signature $\Sigma = (S, F)$. A model for ADTerms is a pair $\mathfrak{M} = (M, I_{\mathfrak{M}})$ consisting of a non-empty set M and a function $I_{\mathfrak{M}}$ defined on F, called interpretation, such that*

– $I_{\mathfrak{M}}(b) \in M$, *for* $b \in \mathbb{B}$,
– $I_{\mathfrak{M}}(f_k)\colon M^k \to M$, *for* $f_k \in \{\vee_k^s, \wedge_k^s \mid s \in S\}$,
– $I_{\mathfrak{M}}(\mathrm{c}^s)\colon M^2 \to M$, *for* $s \in S$.

Let $\mathfrak{M} = (M, I_{\mathfrak{M}})$ be a model for ADTerms, and let t, t' be ADTerms. The *interpretation* of t in $\mathfrak{M}$ is an element $I_{\mathfrak{M}}(t) \in M$, defined as follows

$$I_{\mathfrak{M}}(t) = \begin{cases} I_{\mathfrak{M}}(b), & \text{if } t = b \in \mathbb{B} \\ I_{\mathfrak{M}}(f_k)(I_{\mathfrak{M}}(t_1), \ldots, I_{\mathfrak{M}}(t_k)), & \text{if } t = f_k(t_1, \ldots, t_k). \end{cases}$$

Since the objective of introducing semantics is to decide which ADTerms are indistinguishable, we are interested in formulas of the form $t = t'$. By definition, the formula $t = t'$ is satisfied in $\mathfrak{M}$ if $I_{\mathfrak{M}}(t) = I_{\mathfrak{M}}(t')$. In this case we write $\mathfrak{M} \models t = t'$.

Definition 4. *Let* $\mathfrak{M} = (M, I_{\mathfrak{M}})$ *be a model for ADTerms. The equivalence relation* $\equiv_{\mathfrak{M}}$ *on* T_Σ*, defined by*

$$t \equiv_{\mathfrak{M}} t' \text{ iff } \mathfrak{M} \models t = t',$$

is called the semantics induced by the model $\mathfrak{M}$*. The equivalence class defined by an ADTerm* t *under this relation is denoted by* $[t]_{\mathfrak{M}}$*.*

Propositional Model. The most commonly used model for attack trees is based on propositional logic, cf. [4, 16]. Here, we propose an extension of this model to attack–defense trees. To every basic action $b \in \mathbb{B}$, we associate a propositional variable x_b, and we denote by $\mathbb{F}$ the set of propositional formulas over these variables. Recall that two propositional formulas ψ and ψ' are equivalent ($\psi \approx \psi'$) iff for every valuation ν of propositional variables, we have $\nu(\psi) = \nu(\psi')$. By $\mathbb{F}/_{\approx}$ we denote the quotient space defined by the relation $\approx$, and $[\psi]_{\approx} \in \mathbb{F}/_{\approx}$ stands for the equivalence class defined by a formula ψ.

Definition 5. *The propositional model for ADTerms is the pair* $\mathcal{P} = (\mathbb{F}/_{\approx}, I_{\mathcal{P}})$*, such that, for* $b \in \mathbb{B}$ *and* $s \in \{\mathrm{p}, \mathrm{o}\}$

$$I_{\mathcal{P}}(b) = [x_b]_{\approx}, \qquad I_{\mathcal{P}}(\vee^s) = \vee, \qquad I_{\mathcal{P}}(\wedge^s) = \wedge, \qquad I_{\mathcal{P}}(\mathrm{c}^s) = \star,$$

where $[\psi]_{\approx} \vee [\psi']_{\approx} = [\psi \vee \psi']_{\approx}$*,* $[\psi]_{\approx} \wedge [\psi']_{\approx} = [\psi \wedge \psi']_{\approx}$ *and* $[\psi]_{\approx} \star [\psi']_{\approx} = [\psi \wedge \neg\psi']_{\approx}$*, for all* $\psi, \psi' \in \mathbb{F}$*.*

The semantics $\equiv_{\mathcal{P}}$ induced by the propositional model $\mathcal{P}$ is called the *propositional semantics*. The interpretation of an ADTerm in $\mathcal{P}$ is a set of equivalent propositional formulas expressing the satisfiability of the proponent's goal.

Example 2. Consider $t = \mathrm{c}^{\mathrm{p}}(b, \wedge^{\mathrm{o}}(d, e))$ and $t' = \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(b, b), \wedge^{\mathrm{o}}(d, e))$, where $b \in \mathbb{B}^{\mathrm{p}}$ and $d, e \in \mathbb{B}^{\mathrm{o}}$. The interpretation of t in the propositional model is the equivalence class $I_{\mathcal{P}}(t) = [x_b \wedge \neg(x_d \wedge x_e)]_{\approx}$. The propositional formulas defining this equivalence class are all satisfied iff variable x_b is set to *true* and at least one

of the variables x_d or x_e is set to *false*. It models the fact that in order to achieve his goal, the proponent has to execute the action depicted by b and at least one counteraction, depicted by d or e, must not be executed by the opponent. Since, $x_b \wedge \neg(x_d \wedge x_e) \approx (x_b \wedge x_b) \wedge \neg(x_d \wedge x_e)$, we have $I_{\mathcal{P}}(t) = I_{\mathcal{P}}(t')$, i.e., $t \equiv_{\mathcal{P}} t'$. Thus, t and t' are indistinguishable with respect to the propositional semantics.

As shown in Example 2, the propositional model assumes that the multiplicity of a sub-goal is irrelevant (by idempotency of $\vee$ and $\wedge$). This assumption, however, might not be intended in all applications of ADTrees. It might, for instance, depend on whether parallel or sequential execution of sub-goals is modeled. In the multiset semantics, which we introduce next, we distinguish multiple occurrences of the same actions.

Multiset Model. The model introduced in this paragraph extends the attack trees model defined in [3]. It is suitable for analyzing scenarios in which multiple occurrences of the same sub-goal are significant.

Given a set X, we use 2^X to denote the power set of X, and $\mathcal{M}(X)$ to denote the set of all multisets of elements in X. We use $\{\!\{a_1, \ldots, a_n\}\!\}$ to denote a multiset composed of elements $a_1, \ldots, a_n$. The symbol $\uplus$ stands for the multiset union. A pair $(P, O) \in \mathcal{M}(\mathbb{B}^{\mathrm{p}}) \times \mathcal{M}(\mathbb{B}^{\mathrm{o}})$ is called a *bundle*, and it encodes how the proponent can achieve his goal. A bundle (P, O) represents the following situation: in order to reach his goal, the proponent must perform all actions in P while the opponent must not perform any of the actions in O. In the multiset model we interpret terms with sets of bundles, i.e., with elements from $2^{\mathcal{M}(\mathbb{B}^{\mathrm{p}}) \times \mathcal{M}(\mathbb{B}^{\mathrm{o}})}$. Sets of bundles represent alternative possibilities for the proponent to achieve his goal. A term representing a basic action b of the proponent's type is thus interpreted as a singleton $\{(\{\!\{b\}\!\}, \emptyset)\}$, because in order to achieve his goal it is sufficient for the proponent to execute the action b. A term representing a basic action b of the opponent's type is interpreted as $\{(\emptyset, \{\!\{b\}\!\})\}$, because in order for the proponent to be successful, the counteraction b must not be executed by the opponent. To define the set of bundles that interprets a disjunctive proponent's goal, it is sufficient to take the union of the sets of bundles corresponding to its sub-goals. To define the set of bundles that interprets a conjunctive proponent's goal we introduce the *distributive product*. The distributive product of two sets of bundles S and Z is the following set of bundles

$$S \otimes Z = \{(P_S \uplus P_Z, O_S \uplus O_Z) \mid (P_S, O_S) \in S \text{ and } (P_Z, O_Z) \in Z\}.$$

The distributive product can be extended to any finite number of sets of bundles.

The construction given in the preceding paragraph leads to the multiset model for ADTerms.

Definition 6. *The multiset model for ADTerms is the following pair* $\mathcal{MS} = (2^{\mathcal{M}(\mathbb{B}^{\mathrm{p}})\times\mathcal{M}(\mathbb{B}^{\mathrm{o}})}, I_{\mathcal{MS}})$*, where*

$$I_{\mathcal{MS}}(b) = \{(\{\!\{b\}\!\}, \emptyset)\},\ \textit{for } b \in \mathbb{B}^{\mathrm{p}}, \qquad I_{\mathcal{MS}}(b) = \{(\emptyset, \{\!\{b\}\!\})\},\ \textit{for } b \in \mathbb{B}^{\mathrm{o}},$$
$$I_{\mathcal{MS}}(\vee^{\mathrm{p}}) = \cup, \qquad I_{\mathcal{MS}}(\vee^{\mathrm{o}}) = \otimes,$$
$$I_{\mathcal{MS}}(\wedge^{\mathrm{p}}) = \otimes, \qquad I_{\mathcal{MS}}(\wedge^{\mathrm{o}}) = \cup,$$
$$I_{\mathcal{MS}}(\mathrm{c}^{\mathrm{p}}) = \otimes, \qquad I_{\mathcal{MS}}(\mathrm{c}^{\mathrm{o}}) = \cup.$$

The equivalence relation on ADTerms induced by the multiset model is called the *multiset semantics*, and is denoted by $\equiv_{\mathcal{MS}}$. Note that $I_{\mathcal{MS}}(\vee^s) = I_{\mathcal{MS}}(\wedge^{\overline{s}})$, for $s \in \{\mathrm{p}, \mathrm{o}\}$. This can be explained as follows: in order to achieve his disjunctive goal, the proponent has to achieve *only one* among the corresponding sub-goals, whereas in order to successfully prevent a disjunctive countermeasure of the opponent, the proponent has to prevent *all* the corresponding sub-countermeasures. A similar reasoning holds for conjunctive goals.

Example 3. Consider the ADTerms t and t' introduced in Example 2, that have been shown equivalent with respect to the propositional semantics. We have, $I_{\mathcal{MS}}(t) = \{(\{\!\{b\}\!\}, \{\!\{d\}\!\}), (\{\!\{b\}\!\}, \{\!\{e\}\!\})\}$ and $I_{\mathcal{MS}}(t') = \{(\{\!\{b, b\}\!\}, \{\!\{d\}\!\}), (\{\!\{b, b\}\!\}, \{\!\{e\}\!\})\}$. Since, $I_{\mathcal{MS}}(t) \neq I_{\mathcal{MS}}(t')$, we have $\mathcal{MS} \not\models t = t'$. Thus, the ADTerms t and t' are not equivalent with respect to the multiset semantics.

3.2 Comparing Semantics

In order to compare two semantics, we define what it means that one semantics is finer than a second one. Such a relation allows us to import results about a semantics into any semantics which is coarser.

Definition 7. *Let* $\equiv_1$ *and* $\equiv_2$ *be two semantics for ADTerms. The semantics* $\equiv_1$ *is finer than the semantics* $\equiv_2$ *iff* $\equiv_1 \subseteq \equiv_2$*, i.e., for* $t, t' \in T_\Sigma$*,* $t \equiv_1 t' \Rightarrow t \equiv_2 t'$*.*

The propositional semantics, as opposed to the multiset semantics, does not distinguish multiple occurrences of a basic action. Thus, for instance, absorption laws hold in the propositional but not in the multiset semantics. The relationship between these two semantics is captured by the following proposition.

Proposition 1. *The multiset semantics for ADTerms is finer than the propositional semantics for ADTerms.*

In order to prove the proposition, we use the following lemma.

Lemma 1. *Consider a function* $f\colon 2^{\mathcal{M}(\mathbb{B}^{\mathrm{p}})\times\mathcal{M}(\mathbb{B}^{\mathrm{o}})} \to \mathbb{F}$ *defined as follows*

$$f(\{(\{\!\{p_1^1, \ldots, p_1^{k_1}\}\!\}, \{\!\{o_1^1, \ldots, o_1^{m_1}\}\!\}), \ldots, (\{\!\{p_n^1, \ldots, p_n^{k_n}\}\!\}, \{\!\{o_n^1, \ldots, o_n^{m_n}\}\!\})\}) =$$
$$(x_{p_1^1} \wedge \cdots \wedge x_{p_1^{k_1}} \wedge \neg x_{o_1^1} \wedge \cdots \wedge \neg x_{o_1^{m_1}}) \vee \cdots \vee (x_{p_n^1} \wedge \cdots \wedge x_{p_n^{k_n}} \wedge \neg x_{o_n^1} \wedge \cdots \wedge \neg x_{o_n^{m_n}}),$$

and let t *be an ADTerm. Then*

$$f(I_{\mathcal{MS}}(t)) \in \begin{cases} I_{\mathcal{P}}(t), & \textit{if } t \in T_\Sigma^{\mathrm{p}}, \\ \neg I_{\mathcal{P}}(t), & \textit{if } t \in T_\Sigma^{\mathrm{o}}, \end{cases} \tag{1}$$

where $\neg[\psi]_\approx = [\neg\psi]_\approx$*, for all* $\psi \in \mathbb{F}$*.*

Proof. The proof is by induction on the structure of t. Suppose $t = b \in \mathbb{B}$, then

- if $b \in \mathbb{B}^{\mathrm{p}}$, we get: $f(I_{\mathcal{MS}}(b)) = f(\{(\{\!\{b\}\!\}, \emptyset)\}) = x_b \in [x_b]_{\approx} = I_{\mathcal{P}}(b)$,
- if $b \in \mathbb{B}^{\mathrm{o}}$, we get: $f(I_{\mathcal{MS}}(b)) = f(\{(\emptyset, \{\!\{b\}\!\})\}) = \neg x_b \in \neg[x_b]_{\approx} = \neg I_{\mathcal{P}}(b)$.

Let us now assume that $t \notin \mathbb{B}$, and suppose that (1) holds for all the sub-terms of t. The following observations, resulting from the definition of f, are crucial for the remaining part of the proof. For all ADTerms $t_1, \ldots, t_k$, we have

$$f\big(\bigcup_{j=1}^{k} I_{\mathcal{MS}}(t_j)\big) = \bigvee_{j=1}^{k} f\big(I_{\mathcal{MS}}(t_j)\big), \quad f\big(\bigotimes_{j=1}^{k} I_{\mathcal{MS}}(t_j)\big) = \mathit{DNF}\Big(\bigwedge_{j=1}^{k} f\big(I_{\mathcal{MS}}(t_j)\big)\Big),$$

where $\mathit{DNF}(\psi)$ denotes a disjunctive normal form of the formula ψ. We give a complete proof for an ADTerm of the form $t = \mathrm{c}^{\mathrm{o}}(\vee^{\mathrm{o}}(t_1, \ldots, t_k), t')$. The proofs for the remaining cases are similar. Since t is of the opponent's type, we show that $f(I_{\mathcal{MS}}(t)) \in \neg I_{\mathcal{P}}(t)$. From the type of the function symbol c^{o}, we deduce that $t_1, \ldots, t_k \in T^{\mathrm{o}}_{\Sigma}$ and $t' \in T^{\mathrm{p}}_{\Sigma}$. We have $f(I_{\mathcal{MS}}(t)) =$

$$= f\Big(\big(\bigotimes_{j=1}^{k} I_{\mathcal{MS}}(t_j)\big) \cup I_{\mathcal{MS}}(t')\Big) = f\big(\bigotimes_{j=1}^{k} I_{\mathcal{MS}}(t_j)\big) \vee f\big(I_{\mathcal{MS}}(t')\big)$$

$$= \mathit{DNF}\Big(\bigwedge_{j=1}^{k} f\big(I_{\mathcal{MS}}(t_j)\big)\Big) \vee f\big(I_{\mathcal{MS}}(t')\big)$$

$$\in \big(\bigwedge_{j=1}^{k} \neg I_{\mathcal{P}}(t_j)\big) \vee I_{\mathcal{P}}(t') = \neg\big(\bigvee_{j=1}^{k} I_{\mathcal{P}}(t_j)\big) \vee I_{\mathcal{P}}(t')$$

$$= \neg\Big(\big(\bigvee_{j=1}^{k} I_{\mathcal{P}}(t_j)\big) \wedge \neg I_{\mathcal{P}}(t')\Big) = \neg\Big(\big(\bigvee_{j=1}^{k} I_{\mathcal{P}}(t_j)\big) \star I_{\mathcal{P}}(t')\Big) = \neg I_{\mathcal{P}}(t). \qquad \square$$

Proof of Proposition 1. Suppose $t \equiv_{\mathcal{MS}} t'$. Thus, by definition we have $I_{\mathcal{MS}}(t) = I_{\mathcal{MS}}(t')$. Using the function f from Lemma 1, we have

$$I_{\mathcal{P}}(t) \ni f(I_{\mathcal{MS}}(t)) = f(I_{\mathcal{MS}}(t')) \in I_{\mathcal{P}}(t'), \text{ for } t, t' \in T^{\mathrm{p}}_{\Sigma},$$

$$I_{\mathcal{P}}(t) \ni \neg f(I_{\mathcal{MS}}(t)) = \neg f(I_{\mathcal{MS}}(t')) \in I_{\mathcal{P}}(t'), \text{ for } t, t' \in T^{\mathrm{o}}_{\Sigma},$$

Thus, $I_{\mathcal{P}}(t) = I_{\mathcal{P}}(t')$, i.e., $t \equiv_{\mathcal{P}} t'$, which concludes the proof. $\square$

Examples 2 and 3 show that the reciprocal of Proposition 1 does not hold. Thus, the propositional semantics and the multiset semantics are not equal.

3.3 Equational Semantics

The propositional and multiset semantics were obtained by mapping ADTerms to specific mathematical domains. An alternative way to define a semantics is to directly specify an equivalence relation on ADTerms through a set of equations.

This approach covers a concept from [3], which uses a specific set of rewrite rules to encode allowed tree transformations. Our framework is more general in that we allow any set of equations to define an equivalence relation on ADTerms. With the help of equations it is possible to implement tools that support iterative transformations and modifications of ADTrees.

Let VAR denote a set of typed variables ranged over by X, X_i, Y, Z. We extend the set T_Σ to the set T_Σ^{VAR} of typed ADTerms over the variables from VAR. An *equation* is a pair $(t, t') \in T_\Sigma^{VAR} \times T_\Sigma^{VAR}$, where t and t' have the same type. We use $t = t'$ to denote the equation (t, t'). An *algebraic specification* for ADTerms is a pair (Σ, E), where Σ is the AD–signature and E is a set of equations. Given an algebraic specification (Σ, E), we denote by $\widehat{E}$ the set of equations derivable from E, which is the smallest set satisfying the following

- if $t = t' \in E$, then $t = t' \in \widehat{E}$,
- if $\sigma\colon VAR \to T_\Sigma^{VAR}$ is a substitution, and $t = t' \in \widehat{E}$, then $\sigma(t) = \sigma(t') \in \widehat{E}$,
- if $t = t' \in \widehat{E}$, and $C[\,]$ is a context (i.e., a term with a hole of the same type as t), then $C[t] = C[t'] \in \widehat{E}$,
- $t = t \in \widehat{E}$, for every $t \in T_\Sigma^{VAR}$,
- if $t = t' \in \widehat{E}$, then $t' = t \in \widehat{E}$,
- if $t = t' \in \widehat{E}$ and $t' = t'' \in \widehat{E}$, then $t = t'' \in \widehat{E}$.

We now define a semantics for ADTerms induced by an algebraic specification.

Definition 8. *The equational semantics for ADTerms induced by an algebraic specification (Σ, E) is the equivalence relation $\equiv_E$ on T_Σ, defined by*

$$t \equiv_E t' \text{ iff } t = t' \in \widehat{E}.$$

Example 4. Consider the equational semantics induced by an algebraic specification (Σ, E), where

$$E = \{\vee^{\mathrm{P}}(X_1, \ldots, X_k) = \vee^{\mathrm{P}}(X_{\sigma(1)}, \ldots, X_{\sigma(k)}) \mid \forall \text{ permutation } \sigma \text{ of } \{1, \ldots, k\}\ \}.$$

The equations in E encode the commutativity of the disjunctive operator for the proponent. Thus, for $t_1 = \vee^{\mathrm{P}}(a, b)$ and $t_2 = \vee^{\mathrm{P}}(b, a)$, $a, b \in \mathbb{B}^{\mathrm{P}}$, we have $t_1 \equiv_E t_2$. In contrast, $t_1' = \wedge^{\mathrm{P}}(a, b) \not\equiv_E t_2' = \wedge^{\mathrm{P}}(b, a)$, because the commutativity of the conjunctive operator for the proponent is not modeled by E.

Consider two algebraic specifications (Σ, E) and (Σ, E'), such that $E \subseteq E'$. Since we have $\widehat{E} \subseteq \widehat{E'}$, the semantics $\equiv_E$ is finer then $\equiv_{E'}$.

4 Attributes

In order to analyze an attack–defense scenario represented by an ADTerm we use attributes. An attribute is a value assigned to an ADTerm, expressing a useful property, such as the minimal cost of an attack, expected impact, whether special equipment is required, or whether a considered scenario is feasible. In [2], Schneier introduces an intuitive, bottom-up algorithm for calculating the value of an attribute on an attack tree. This idea is formalized in [3]. In this section we extend it to attack–defense trees.

4.1 Bottom-up Evaluation

Let $\Sigma = (S, F)$ be the AD–signature. An *attribute domain* is a pair $A_\alpha = (D_\alpha, I_\alpha)$, where D_α is a set of values, and I_α is a function,which to every $f_k \in F$ with $k > 0$, associates a k-ary operation $I_\alpha(f_k)$ on D_α. An *attribute* for ADTerms is a pair $\alpha = (A_\alpha, \beta_\alpha)$ formed by an attribute domain A_α and a function $\beta_\alpha \colon \mathbb{B} \to D_\alpha$ called a *basic assignment* for α. The next definition formalizes the bottom-up procedure of calculating attribute values.

Definition 9. *Let $\alpha = ((D_\alpha, I_\alpha), \beta_\alpha)$ be an attribute. The function $\alpha \colon T_\Sigma \to D_\alpha$ which calculates the value of the attribute α for every ground ADTerm t, is defined recursively as follows*

$$\alpha(t) = \begin{cases} \beta_\alpha(t), & \text{if } t \in \mathbb{B}, \\ I_\alpha(f_k)(\alpha(t_1), \ldots, \alpha(t_k)), & \text{if } t = f_k(t_1, \ldots, t_k). \end{cases}$$

The following example illustrates the bottom-up evaluation of attribute values.

Example 5. Consider the ADTerm $t = \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(a, b), \mathrm{c}^{\mathrm{o}}(d, e))$, where $a, b, e \in \mathbb{B}^{\mathrm{p}}$, $d \in \mathbb{B}^{\mathrm{o}}$ are independent basic actions. We define the following operations on the interval $[0, 1]$: $\oplus(x, y) = x + y - xy$, $\odot(x, y) = xy$ and $\ominus(x, y) = x(1 - y)$. Using the attribute domain $A_{\mathrm{Pr}} = ([0, 1], I_{\mathrm{Pr}})$, where $I_{\mathrm{Pr}}(\vee^s) = \oplus$, $I_{\mathrm{Pr}}(\wedge^s) = \odot$ and $I_{\mathrm{Pr}}(\mathrm{c}^s) = \ominus$, for $s \in \{\mathrm{p}, \mathrm{o}\}$, we calculate the success probability of the attack–defense scenario represented by t. The success probabilities of basic actions are set as follows $\beta_{\mathrm{Pr}}(a) = 0.2$, $\beta_{\mathrm{Pr}}(b) = 0.7$, $\beta_{\mathrm{Pr}}(e) = 0.1$, and $\beta_{\mathrm{Pr}}(d) = 0.9$. According to Definition 9, we obtain $\mathrm{Pr}(t) = \mathrm{Pr}(\mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(a, b), \mathrm{c}^{\mathrm{o}}(d, e))) = \ominus(\odot(0.2, 0.7), \ominus(0.9, 0.1)) = 0.0266$.

4.2 Semantics Preserving Attribute Values

In our framework, we consider equivalent ADTrees as indistinguishable. Thus, the evaluation of attributes for equivalent ADTerms should be consistent, i.e., should yield the same values. This issue has already been discussed in case of attack trees, cf. [18, 3]. In [18], the evaluation of the attacker's expected outcome (based on cost, probabilities, expected penalties and gains) is considered. Using the propositional semantics, Jürgenson and Willemson propose a non-bottom-up procedure ensuring that the expected outcome attribute is calculated consistently. In their approach, optimization becomes necessary, because the corresponding computations are exponential with respect to the size of a tree. In [3], Mauw and Oostdijk consider a bottom-up way of calculating attributes, as defined in Definition 9. They show that when using the multiset semantics on attack trees, the attribute evaluation is consistent if the considered attribute domain is distributive, i.e., if it constitutes a semi-ring. In the current paper, we extend this result to any semantics for ADTrees. We introduce a notion of compatibility between a semantics and an attribute domain, which guarantees that the intuitive properties modeled by the semantics are preserved by the attribute domain. The compatibility is a necessary condition for consistent bottom-up evaluation of attributes on ADTrees.

Definition 10. *Let (Σ, E) be an algebraic specification, and let $\equiv$ be a semantics for ADTerms. The set E is called a complete set of axioms for $\equiv$, iff the relations $\equiv$ and $\widehat{E} \cap (T_\Sigma \times T_\Sigma)$ are equal.*

Example 6. Consider the equational semantics $\equiv_E$ induced by an algebraic specification (Σ, E). It follows directly from Definition 8 that the set E is a complete set of axioms for $\equiv_E$.

Let $A_\alpha = (D_\alpha, I_\alpha)$ be an attribute domain. Given $t \in T_\Sigma^{VAR}$, we denote by t_α an expression, composed of the elements from $\mathbb{B} \cup VAR$ (here considered as variables) and operators $I_\alpha(f_k)$, where $f_k \in F$, $k \geq 1$, defined as follows

$$t_\alpha = \begin{cases} t, & \text{if } t \in \mathbb{B} \cup VAR, \\ I_\alpha(f_k)\left(t_\alpha^1, \ldots, t_\alpha^k\right), & \text{if } t = f_k\left(t^1, \ldots, t^k\right). \end{cases}$$

Definition 11. *An equivalence relation $\equiv$ on T_Σ is compatible with an attribute domain $A_\alpha = (D_\alpha, I_\alpha)$ iff for all ADTerms t, t' such that $t \equiv t'$, the equality $t_\alpha = t'_\alpha$ holds in D_α.*

Consider a complete set of axioms E for a semantics $\equiv$. It follows from Definitions 11 and 10, that the semantics $\equiv$ is compatible with an attribute domain A_α iff for every equation $t = t'$ from E, the equality $t_\alpha = t'_\alpha$ holds in D_α.

In the following, we show that when considering a semantics that is compatible with a given attribute domain, the evaluation of attributes on equivalent ADTerms yields the same values.

Theorem 1. *Let $\alpha = ((D_\alpha, I_\alpha), \beta_\alpha)$ be an attribute, and t, t' be ADTerms. If $t_\alpha = t'_\alpha$ holds in D_α, then $\alpha(t) = \alpha(t')$.*

Proof. Since $t_\alpha = t'_\alpha$ holds in D_α, we have $\sigma(t_\alpha) = \sigma(t'_\alpha)$, for every substitution $\sigma \colon \mathbb{B} \cup VAR \to D_\alpha$. Thus, it suffices to show that for every ADTerm t, we have

$$\beta_\alpha(t_\alpha) = \alpha(t). \tag{2}$$

The proof of (2) is by induction on the structure of t. If $t \in \mathbb{B}$, then $t_\alpha = t$, thus $\beta_\alpha(t_\alpha) = \beta_\alpha(t) = \alpha(t)$. Suppose now, that for all ADTerms composing t, we have (2), and let $t = f_k(t^1, \ldots, t^k)$. We have

$$\begin{aligned} \beta_\alpha(t_\alpha) &= \beta_\alpha(I_\alpha(f_k)(t_\alpha^1, \ldots, t_\alpha^k)) = I_\alpha(f_k)(\beta_\alpha(t_\alpha^1), \ldots, \beta_\alpha(t_\alpha^k)) \\ &= I_\alpha(f_k)(\alpha(t^1), \ldots, \alpha(t^k)) = \alpha(t). \end{aligned}$$

□

Corollary 1. *Let $\alpha = (A_\alpha, \beta_\alpha)$ be an attribute and let $\equiv$ be a semantics for ADTerms compatible with A_α. If $t \equiv t'$, then $\alpha(t) = \alpha(t')$.*

Corollary 1 guarantees that, given an attribute domain and a compatible semantics, the attributes can be calculated in a consistent way, In Example 7 we show how the compatibility notion defined in Definition 11 covers the result obtained by Mauw and Oostdijk in [3].

Example 7. The multiset semantics for attack trees used in [3] can be axiomatized with the following set of rules, for $f_i \in \{\vee_i^{\mathrm{p}}, \wedge_i^{\mathrm{p}}\}$, $i \geq 1$,

$$f_k(X_1, \ldots, X_k) = f_k(X_{\sigma(1)}, \ldots, X_{\sigma(k)}), \text{ for every permutation } \sigma \text{ of } \{1, \ldots, k\},$$
$$f_{k+1}(X_1, \ldots, X_k, f_n(Y_1, \ldots, Y_n)) = f_{k+n}(X_1, \ldots, X_k, Y_1, \ldots, Y_n),$$
$$\wedge^{\mathrm{p}}(X, \vee^{\mathrm{p}}(X_1, \ldots, X_k)) = \vee^{\mathrm{p}}(\wedge^{\mathrm{p}}(X, X_1), \ldots, \wedge^{\mathrm{p}}(X, X_k)),$$
$$\vee^{\mathrm{p}}(X, X, X_1, \ldots, X_k) = \vee^{\mathrm{p}}(X, X_1, \ldots, X_k).$$

Note that the corresponding equalities always hold in every attribute domain $A_\alpha = (D_\alpha, I_\alpha)$, such that $(D_\alpha, I_\alpha(\vee^{\mathrm{p}}), I_\alpha(\wedge^{\mathrm{p}}))$ constitutes a semi-ring. Thus, the multiset semantics for attack trees is compatible with any attribute domain being a semi-ring.

5 Conclusion and Future Work

We introduce attack–defense trees as a new formal approach for security assessment. These ADTrees provide an intuitive and visual representation of interactions between an attacker and a defender of a system, as well as the evolution of the security mechanisms and vulnerabilities of a system.

The attack–defense language is based on ADTerms, i.e., the term algebra for ADTrees. We define semantics for ADTrees as equivalence relations on ADTerms. This general framework unifies different approaches [3–5] to attack trees that have been proposed in the literature, because they all rely upon an underlying equivalence relation.

Furthermore, analysis of ADTrees is supported through attributes and their bottom-up evaluation. This extends the approach proposed for attack trees in [3]. Finally, we formulate a necessary condition guaranteeing that equivalent ADTerms yield the same attribute value.

The purpose of this paper is to lay a formal foundation for attack–defense trees. To demonstrate the applicability of attack–defense trees on a real-world example is impossible without a tool, due to the large size of the resulting attack–defense trees. Examples for the applicability of a similar approach can, at present, be found in works on attack trees, e.g., [5, 12, 19, 20].

In order to allow for meaningful case-studies with attack–defense trees, a computer tool will be developed next. It will facilitate the construction of large ADTrees, support their graphical representation, and assist in the analysis of ADTrees by combining information assigned to the basic actions in an ADTree into a single value for the analyzed scenario. Furthermore, automated generation and analysis of ADTrees is planned for particular domains, such as network security. The feasibility of such a work has been demonstrated by Sheyner et al. [13], who have shown how to automatically generate and analyze attack graphs from the output of a model checker for an intrusion detection system.

We also plan to extend the attack–defense framework to attack–defense DAGs. Using DAGs one can model dependencies between the sub-goals. This issue is crucial when taking the execution order of sub-goals into account or when analyzing an attack–defense scenario from a probabilistic point of view.

References

1. Vesely, W.E., Goldberg, F.F., Roberts, N., Haasl, D.: Fault Tree Handbook. Technical Report NUREG-0492, U.S. Regulatory Commission (1981)
2. Schneier, B.: Attack Trees. Dr. Dobb's Journal of Software Tools **24**(12) (1999) 21–29
3. Mauw, S., Oostdijk, M.: Foundations of Attack Trees. In Won, D., Kim, S., eds.: ICISC. Volume 3935 of LNCS., Springer (2005) 186–198
4. Willemson, J., Jürgenson, A.: Serial Model for Attack Tree Computations. In Lee, D., Hong, S., eds.: Proc. ICISC 2009. Volume 5984 of LNCS., Springer (2010) 118–128
5. Edge, K.S., Dalton II, G.C., Raines, R.A., Mills, R.F.: Using Attack and Protection Trees to Analyze Threats and Defenses to Homeland Security. In: Military Communications Conference, 2006. MILCOM 2006. IEEE. (2006) 1–7
6. Saini, V., Duan, Q., Paruchuri, V.: Threat Modeling Using Attack Trees. Journal of Computing in Small Colleges **23**(4) (2008) 124–131
7. Bistarelli, S., Fioravanti, F., Peretti, P.: Defense Trees for Economic Evaluation of Security Investments. In: ARES, IEEE Computer Society (2006) 416–423
8. Bistarelli, S., Dall'Aglio, M., Peretti, P.: Strategic Games on Defense Trees. In Dimitrakos, T., Martinelli, F., Ryan, P.Y.A., Schneider, S.A., eds.: Formal Aspects in Security and Trust. Volume 4691 of LNCS., Springer (2006) 1–15
9. Moore, A.P., Ellison, R.J., Linger, R.C.: Attack Modeling for Information Security and Survivability. Technical Report CMU/ SEI-2001-TN-001, CMU Software Eng (2001)
10. Cervesato, I., Meadows, C.: One Picture Is Worth a Dozen Connectives: A Fault-Tree Representation of NPATRL Security Requirements. IEEE Transactions on Dependable and Secure Computing **4** (2007) 216–227
11. Amoroso, E.G.: Fundamentals of Computer Security Technology. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1994)
12. Morais, A.N.P., Martins, E., Cavalli, A.R., Jimenez, W.: Security Protocol Testing Using Attack Trees. In: CSE (2), IEEE Computer Society (2009) 690–697
13. Sheyner, O., Haines, J.W., Jha, S., Lippmann, R., Wing, J.M.: Automated Generation and Analysis of Attack Graphs. In: IEEE Symposium on Security and Privacy, Los Alamitos, CA, USA, IEEE Computer Society (2002) 273–284
14. Bistarelli, S., Peretti, P., Trubitsyna, I.: Analyzing Security Scenarios Using Defence Trees and Answer Set Programming. Electronic Notes in Theoretical Computer Science **197**(2) (2008) 121–129
15. Kordy, B., Mauw, S., Melissen, M., Schweitzer, P.: Attack–Defense Trees and Two-Player Binary Zero-Sum Extensive Form Games Are Equivalent. Proceedings of GameSec 2010. LNCS., Springer-Verlag. Avail. at http://arxiv.org/abs/1006.2732.
16. Rehák, M., Staab, E., Fusenig, V., Pěchouček, M., Grill, M., Stiborek, J., Bartoš, K., Engel, T.: Runtime Monitoring and Dynamic Reconfiguration for Intrusion Detection Systems. In Kirda, E., Jha, S., Balzarotti, D., eds.: Proceedings of RAID '09. Volume 5758 of LNCS., Springer-Verlag (2009) 61–80
17. Doets, K.: Basic Model Theory. Stanford: CSLI Publications (1996)
18. Jürgenson, A., Willemson, J.: Computing Exact Outcomes of Multi-parameter Attack Trees. In Meersman, R., Tari, Z., eds.: OTM Conferences (2). Volume 5332 of LNCS., Springer (2008) 1036–1051
19. Amenaza: SecurITree http://www.amenaza.com/.
20. Isograph: AttackTree+ http://www.isograph-software.com/atpover.htm.