Attack–defense trees[1]

BARBARA KORDY, *University of Luxembourg, Interdisciplinary Centre for Security, Reliability and Trust.*

SJOUKE MAUW, *University of Luxembourg, Interdisciplinary Centre for Security, Reliability and Trust.*

SAŠA RADOMIROVIĆ, *ETH Zürich, Institute of Information Security*

PATRICK SCHWEITZER, *University of Luxembourg, Interdisciplinary Centre for Security, Reliability and Trust.*
E-mail: {barbara.kordy,sjouke.mauw,patrick.schweitzer}@uni.lu
sasa.radomirovic@inf.ethz.ch

Abstract

Attack–defense trees are a novel methodology for graphical security modelling and assessment. They extend the well- known formalism of attack trees by allowing nodes that represent defensive measures to appear at any level of the tree. This enlarges the modelling capabilities of attack trees and makes the new formalism suitable for representing interactions between an attacker and a defender. Our formalization supports different semantical approaches for which we provide usage scenarios. We also formalize how to quantitatively analyse attack and defense scenarios using attributes.

Keywords: Attack–defense trees, attack trees, security assessment, semantics, countermeasures, attributes, complete set of axioms.

1 Introduction

It is a well-known fact that the security of any sufficiently valuable system is not static. 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?

In 1999, Schneier popularized *attack trees* as a tool to evaluate the security of complex systems [29]. An attack tree is a tree-like representation of an attack scenario. 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 actions to be executed by the attacker.

An obvious limitation of attack trees is that they cannot capture the *interaction* between attacks carried out on a system and the defenses that could be put in place to fend off the attacks. This consequently limits the precision with which the best defensive strategies can be analysed, since

[1] A preliminary version of this article has appeared in the proceedings of FAST 2010.

Vol. 24 No. 1,
Published online June 21, 2012 doi:10.1093/logcom/exs029

it does not take into account the effects of potential defensive measures which would need to be overcome by new attacks. Similarly, a regular attack tree does not allow for the visualization and consideration of the *evolution* of a system's security, because 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. To model the ongoing arms race between attacks and defenses, it is necessary to allow for alternation between these two types of actions. We therefore introduce attack–defense trees (ADTrees) as a graphical representation of possible measures an attacker might take to attack a system and the defenses that a defender can employ to protect the system.

The contributions of this article are as follows:

(1) We develop an extension of attack trees with defense nodes.
The new formalism is called *ADTrees*. It generalizes and unifies existing approaches to extend attack trees.

(2) We formalize the meaning of an ADTree.
We propose a framework in which a variety of semantics can be defined. This is motivated by the fact that different applications require different interpretations of ADTrees. We develop the following semantics:

- The class of *semantics induced by De Morgan lattices*.
This class contains the *propositional semantics* which is the most frequently used semantics for attack trees.
- *Multiset semantics*.
This class extends the semantics proposed for attack trees in [21] to ADTrees.
- The class of *equational semantics*.
Equational semantics are defined by sets of equations over ADTrees. They constitute, therefore, a very general class of semantics and aid in establishing relations between different semantics for ADTrees.

We provide a complete axiomatization for the propositional and the multiset semantics.

(3) We introduce the notion of an *attribute* for ADTrees.
The introduction of attributes enables a quantitative analysis of attack–defense scenarios. It requires the formalization of a *compatibility condition*, which guarantees that the evaluation of an attribute on two semantically equal ADTrees results in the same value for both trees.

The article is structured as follows. In Section 2 we formally introduce ADTrees, give an example, and define attack–defense terms (ADTerms) which are a formal representation of ADTrees. We present various semantics for ADTrees in Section 3. We show how to compare different semantics introduced in this article in Section 4 where we also provide complete axiomatizations for the propositional and the multiset semantics. In Section 5, we study how to quantitatively analyse ADTrees with the help of attributes. We review related work in Section 6 and conclude in Section 7.

2 Attack–defense trees

2.1 Terminology

An *attack–defense tree* (ADTree) is a node-labelled rooted tree describing the measures an attacker might take to attack a system and the defenses that a defender can employ to protect the system. ADTrees have nodes of two *opposite types*: *attack* nodes and *defense* nodes, which correspond to an attacker's and a defender's (sub-)goals, respectively.

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 so-called *basic actions*.

Every node may also have one child 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, i.e. an attack node and counters the defense.

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 model attack–defense scenarios. An attack–defense scenario can be seen as 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.

When drawing 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 an arc over all edges connecting the node and its children of equal type.

2.2 *Example*

To demonstrate the features of ADTrees, we consider the following fictitious scenario concerning data confidentiality in a data hosting center. The ADTree representing the scenario is shown in Figure 1. Its root node is a defense, thus the main goal expressed by the tree is the protection of data confidentiality.

To protect the confidentiality of costumer data, the hosting company needs to invest in network security as well as in physical security measures. These measures break up into several aspects that need to be taken care of. However, even if both of physical and network security were to be infallible, the company's employees would still be a weak point. Two common options to subvert a company through its employees are corruption and social engineering. These attacks can be mitigated through employee screenings and sensitivity training for social engineering techniques.

Network security is a very complex problem, and it is beyond the purpose of this introductory example to show all possible defenses. Some standard measures employed towards network security are firewalls, intrusion detection and access control systems. Of these, we are displaying the evolution of access control through the use of passwords. In many access controlled services, passwords used to be free of any restrictions regarding the type of characters they need to contain. Consequently, a significant number of passwords chosen consisted of a name or dictionary word, since these are much easier to remember than a random sequence of characters. This has led to access control breaches through so-called dictionary attacks. To prevent these attacks, computer systems nowadays require 'strong' passwords, which are to consist of letters, numbers and non-alphanumeric characters. This mechanism, however, induces people to write up their passwords on easily accessible sticky notes or to reuse the same strong password for different accounts and services. Thus, the strong password required for the data center may be recovered by attacking an unrelated and possibly weaker system the target user has an account on.

Regarding physical security, a building can be broken into through back doors, windows or fire escapes. It is, therefore, common to reinforce windows and to protect other entrances with locks. The locks can be circumvented by forcing them open or by obtaining a key. It is therefore increasingly

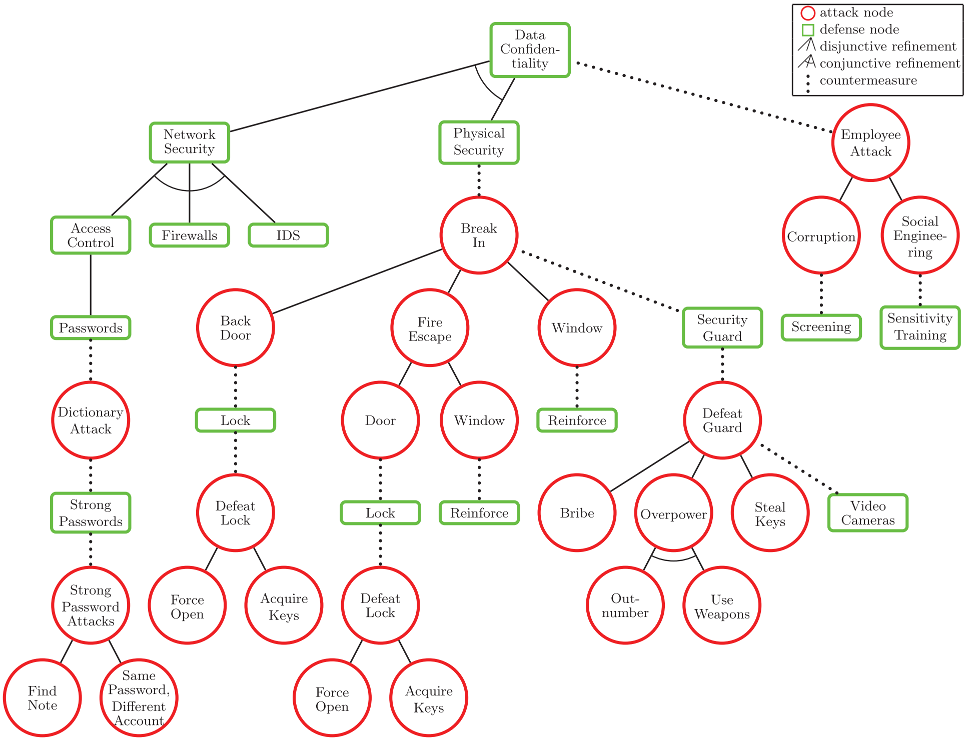


FIGURE 1. An ADTree for protecting data confidentiality.

common to employ security guards to monitor the building. To effectively monitor the building, a security guard will typically have the keys not only to the building itself, but also to all rooms in the building. This makes the security guard a possible attack vector. He could be bribed or overpowered, or his keys could be stolen in some manner. To overpower the guard, it would be necessary to outnumber him and threaten him with weapons. To prevent these three attacks, video cameras with remote surveillance could be employed.

The scenario as described thus far is obviously incomplete. It is clear, however, that for any addition to the scenario, it would be very simple to extend the ADTree shown in Figure 1 with new attacks and defenses.

2.3 *Formal representation*

To formally analyse ADTrees, we define an abstract syntax which we call ADTerms. ADTerms are typed terms over a particular signature called the AD-signature. To define the AD-signature, 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 $F_k : D^k \to R$, for $k > 0$. Given a set $\mathcal{S}$, we denote by $\mathcal{S}^*$ the set of all finite strings over $\mathcal{S}$ and by ε the empty string.

DEFINITION 2.1
The *AD-signature* is a pair $\Sigma = (\mathcal{S}, \mathcal{F})$, where

- $\mathcal{S} = \{\mathrm{p}, \mathrm{o}\}$ is a set of types, and

- $\mathcal{F} = \{(\vee^{\mathrm{p}}_k)_{k\in\mathbb{N}}, (\wedge^{\mathrm{p}}_k)_{k\in\mathbb{N}}, (\vee^{\mathrm{o}}_k)_{k\in\mathbb{N}}, (\wedge^{\mathrm{o}}_k)_{k\in\mathbb{N}}, \mathrm{c}^{\mathrm{p}}, \mathrm{c}^{\mathrm{o}}\} \cup \mathbb{B}^{\mathrm{p}} \cup \mathbb{B}^{\mathrm{o}}$ is a set of function symbols, such that $\{(\vee^{\mathrm{p}}_k)_{k\in\mathbb{N}}, (\wedge^{\mathrm{p}}_k)_{k\in\mathbb{N}}, (\vee^{\mathrm{o}}_k)_{k\in\mathbb{N}}, (\wedge^{\mathrm{o}}_k)_{k\in\mathbb{N}}, \mathrm{c}^{\mathrm{p}}, \mathrm{c}^{\mathrm{o}}\}$, $\mathbb{B}^{\mathrm{p}}$ and $\mathbb{B}^{\mathrm{o}}$ are pairwise disjoint.

Every function symbol $F \in \mathcal{F}$ is equipped with a mapping $\mathrm{rnk} : \mathcal{F} \to \mathcal{S}^* \times \mathcal{S}$, called rank. The rank of a function symbol F is a pair $\mathrm{rnk}(F) = (\mathrm{arity}(F), (F))$, where the first component describes the arity of F and the second specifies its type. For the function symbols in $\mathcal{F}$ and $k \in \mathbb{N}$, we define

$$\begin{array}{ll}
\mathrm{rnk}(b) = (\varepsilon, \mathrm{p}),\ \text{for } b \in \mathbb{B}^{\mathrm{p}}, & \mathrm{rnk}(b) = (\varepsilon, \mathrm{o}),\ \text{for } b \in \mathbb{B}^{\mathrm{o}}, \\
\mathrm{rnk}(\vee^{\mathrm{p}}_k) = (\mathrm{p}^k, \mathrm{p}), & \mathrm{rnk}(\vee^{\mathrm{o}}_k) = (\mathrm{o}^k, \mathrm{o}), \\
\mathrm{rnk}(\wedge^{\mathrm{p}}_k) = (\mathrm{p}^k, \mathrm{p}), & \mathrm{rnk}(\wedge^{\mathrm{o}}_k) = (\mathrm{o}^k, \mathrm{o}), \\
\mathrm{rnk}(\mathrm{c}^{\mathrm{p}}) = (\mathrm{po}, \mathrm{p}), & \mathrm{rnk}(\mathrm{c}^{\mathrm{o}}) = (\mathrm{op}, \mathrm{o}).
\end{array}$$

The elements of $\mathbb{B}^{\mathrm{p}}$ and $\mathbb{B}^{\mathrm{o}}$ are typed constants, which we call *basic actions of the proponent's type* and *basic actions of the opponent's type*, respectively. We denote the set of all basic actions by $\mathbb{B} = \mathbb{B}^{\mathrm{p}} \cup \mathbb{B}^{\mathrm{o}}$. 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 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 $s \in \mathcal{S}$, connect actions of type s with actions of the opposite type $\overline{s}$.

Definition 2.2
Typed ground terms over the AD-signature Σ are called ADTerms. The set of all ADTerms is denoted by $\mathbb{T}_\Sigma$.

For $s \in \{\mathrm{p}, \mathrm{o}\}$, we denote by $\mathbb{T}^s_\Sigma$ the set of all ADTerms with the head symbol of type s. We have $\mathbb{T}_\Sigma = \mathbb{T}^{\mathrm{p}}_\Sigma \cup \mathbb{T}^{\mathrm{o}}_\Sigma$. The elements of $\mathbb{T}^{\mathrm{p}}_\Sigma$ and $\mathbb{T}^{\mathrm{o}}_\Sigma$ are called *ADTerms of the proponent's* and *of the opponent's type*, respectively. The ADTerms of the proponent's type constitute a formal representation of ADTrees. Attack trees are formally represented by ADTerms of the proponent's type that are built exclusively from basic actions of the proponent's type and functions $\vee^{\mathrm{p}}$ and $\wedge^{\mathrm{p}}$.

In the remaining part of this section, we give a formal definition of ADTrees and we show how ADTrees correspond to ADTerms.

The definition of an ADTree is based on the notion of a finite ordered tree, as introduced in [9]. A *finite ordered tree T over a set of labels L* is a function $T : \mathrm{Pos}(T) \to L$, where $\mathrm{Pos}(T)$ is a prefix-closed subset of $(\mathbb{N} \setminus \{0\})^*$, called the set of positions of T. We depict T as a graph in the following manner. The positions in $\mathrm{Pos}(T)$ are drawn as nodes labelled with elements of L. The position ε is the root node of the graph, depicted as the topmost node. The positions pi, where $i \in \{1, \ldots, k\}$ for some $k > 0$, are the children of the node corresponding to the position p. Since T is ordered, the node corresponding to the position pi is drawn left of the node depicting position pj, for all $i < j$.

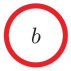
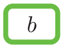
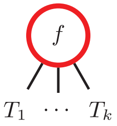
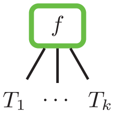
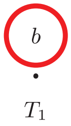
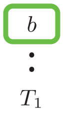
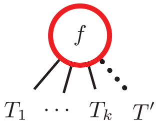
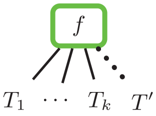
An ADTree is then formally defined as follows.

Definition 2.3
An ADTree (ADTree) is a finite ordered tree T over the set of labels $L_T = \mathbb{B}^{\mathrm{p}} \cup \mathbb{B}^{\mathrm{o}} \cup \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}, \vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$, together with a function $\lambda : \mathrm{Pos}(T) \to \{\bigcirc, \square\}$ which satisfies the following two conditions for every $p \in \mathrm{Pos}(T)$.

(1) If there exists $i \in \mathbb{N} \setminus \{0\}$, such that $pi \in \mathrm{Pos}(T)$ and $\lambda(pi) = \lambda(p)$, then

$$T(p) \in \begin{cases} \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}\} & \text{if } \lambda(p) = \lambda(\varepsilon), \\ \{\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\} & \text{else,} \end{cases}$$

TABLE 1. Transformation from ADTrees to ADTerms

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

otherwise

$$T(p) \in \begin{cases} \mathbb{B}^p & \text{if } \lambda(p) = \lambda(\varepsilon), \\ \mathbb{B}^o & \text{else.} \end{cases}$$

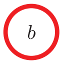
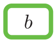
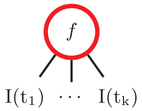
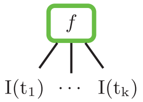
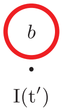
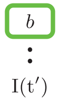
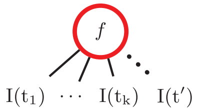
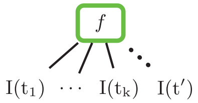
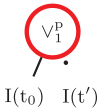
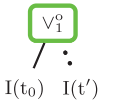
(2) For every $i \in \mathbb{N} \setminus \{0\}$, if $\lambda(pi) \neq \lambda(p)$, then $\forall j > i$ $pj \notin \mathrm{Pos}(T)$.

The function λ allows us to distinguish between attack nodes (○) and defense nodes (□). Value $\lambda(\varepsilon)$ determines for the considered tree which player (attacker or defender) is the proponent and which is the opponent. By comparing the values of λ applied to a parent node with the values of λ applied to its children we can decide which nodes are refined and which non-refined. A node p is refined if it has at least one child pi such that $\lambda(p) = \lambda(pi)$. A non-refined node can have at most one child $p1$, and this child needs to satisfy $\lambda(p) \neq \lambda(p1)$. Condition 1 of Definition 2.3 guarantees that each node p of an ADTree is either refined in a conjunctive or disjunctive way ($T(p) \in \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}, \vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$) or is a non-refined node ($T(p) \in \mathbb{B}^{\mathrm{p}} \cup \mathbb{B}^{\mathrm{o}}$). Condition 2 states that each node p may only have one child of the opposite type. Moreover, if such a child exists, it is always depicted as the rightmost child node of p.

In the formal definition of ADTrees, refined nodes are labelled with the associated refining symbols. In practice, such nodes are typically labelled with descriptive names of the (sub-)goals they represent, as shown in Figure 1.

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, circular nodes have to be replaced with rectangular 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 $s \in \{\mathrm{p}, \mathrm{o}\}$.

TABLE 2. Transformation from ADTerms to ADTrees

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 $f \in \{\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$, $k \geq 1$
I(t)				
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 $f \in \{\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$, $k \geq 1$
I(t)				
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 $t_0 = \mathrm{c}^{\mathrm{o}}(t_1, t_2)$
I(t)				

EXAMPLE 2.4
The ADTerm representing the sub-tree of the ADTree in Figure 1, rooted at the *Security Guard* node, is the following

$$\mathrm{c}^{\mathrm{p}}\Big(\text{SecGuard}, \mathrm{c}^{\mathrm{o}}\big(\vee^{\mathrm{o}}(\text{Bribe}, \wedge^{\mathrm{o}}(\text{Outnumb}, \text{Weapons}), \text{StealKeys}), \text{Cameras}\big)\Big).$$

Note that the names of refined nodes in the ADTree, such as 'Defeat Guard' and 'Overpower', do not appear in the ADTerm. Instead, these nodes are represented with the corresponding refining symbols $\vee^{\mathrm{o}}$ and $\wedge^{\mathrm{o}}$.

2.4 *Design choices*

When designing the ADTree formalism, we have deliberately made the following modelling choices to keep a balance between usability, complexity and representational impact.

(1) *Refinements and countermeasures.* An ADTree node is refined either conjunctively or disjunctively. Refinement operators are unranked. Each ADTree node may only have one child of opposite type. These choices were made in order for ADTrees to reflect as closely as possible a description of an attack–defense scenario in natural language.
These choices do not limit the expressiveness of the formalism. We would obtain an equally expressive formalism by restricting ADTrees to binary refinements, by allowing nodes with multiple countermeasures, or by allowing nodes that are conjunctively and disjunctively refined at the same time.

(2) *ADTrees versus parse trees of ADTerms.* The ADTree corresponding to an ADTerm of the form $t = \mathrm{c}^{\mathrm{p}}(t_1, t_2)$ differs from the parse tree of t. We depict the root of the tree corresponding to

t_2 as a child of the root node of the tree corresponding to t_1. In this manner we illustrate that t_2 represents a countermeasure for the scenario depicted by t_1. Such an illustration helps us to model interactions between the two players involved in an attack–defense scenario in an intuitive and understandable way.

(3) *Finite trees.* We consider only finite ADTrees in this article for the sake of simplicity. Infinite ADTrees are conceivable, for instance, to model recursive goals, such as obtaining keys to a locked box which contains the keys. Infinite ADTrees would also be a useful tool to study the limit case of evolving attack–defense scenarios, such as automated attacks and defenses.

(4) *Ordered trees.* We define ADTrees to be ordered trees. This choice makes ADTrees suitable for the analysis of scenarios in which the order between actions is relevant. This could, for instance, be the case when temporal relations are taken into account.

(5) *Trees versus DAG.* We use trees instead of directed acyclic graph (DAG) for simplicity of the formalism. DAGs are more expressive because they can be used to indicate dependencies between nodes. For instance, the two nodes labelled ‘Window’ in Figure 1 could be replaced by a single node to express that they concern the same physical window or that all attacks to one window also apply to the other window. Since such shared nodes give rise to different possible interpretations and to a more complicated semantical treatment, we leave the extension to DAGs for future research.

3 Semantics for ADTerms

3.1 Definition of semantics

ADTerms represent attack–defense scenarios. Depending on how ADTerms are interpreted, syntactically different terms may be considered equivalent. A semantics for ADTerms defines such equivalence classes. Terms that belong to the same equivalence class represent the same scenario.

DEFINITION 3.1
A semantics for ADTerms is an equivalence relation on $\mathbb{T}_\Sigma$ that preserves types.

Depending on the semantics, the most natural ADTerm for a scenario may not be the simplest possible.

EXAMPLE 3.2
Consider an attack scenario in which three different doors need to be opened with the same key. The scenario can be represented by the ADTerm $t = \wedge^{\mathrm{p}}(\mathrm{OpenDoor}, \mathrm{OpenDoor}, \mathrm{OpenDoor})$. If it is only the feasibility of the scenario which is of interest, the number of doors to be opened is irrelevant. In this case, t represents the same scenario as $t' = \mathrm{OpenDoor}$.

An essential feature of the ADTree methodology is that ADTerms can be equipped with multiple semantics. Different applications require the use of different semantics. The two terms t and t' in Example 3.2 are equivalent if the feasibility of the attack scenario is examined. However, t and t' are no longer equivalent if the attacker is interested in how much time is required to achieve his attack. The choice of an appropriate semantics becomes crucial when a quantitative analysis of an attack–defense scenario is to be performed. We will discuss this issue in Section 5.

3.2 Propositional semantics

Attack trees are often seen as representations of and– or formulæ. Thus, one of the most frequently used semantics for attack trees is the propositional semantics [16, 17, 26, 34]. In this section, we extend this semantics to ADTerms. When the propositional semantics is used, ADTerms are interpreted as propositional formulæ. The satisfiability of the formula interpreting an ADTerm t models the feasibility of the scenario represented by t. The propositional semantics is well-suited to evaluate whether a system is vulnerable to an attack, in how many different ways a system can be successfully attacked, or whether special equipment is needed to perform an attack.

We assign a propositional variable x_b to every basic action $b \in \mathbb{B}$. We assume that different basic actions give rise to different propositional variables. In particular, since the sets of basic actions of the proponent's and of the opponent's type are disjoint, we have

$$\{x_b \mid b \in \mathbb{B}^{\mathrm{p}}\} \cap \{x_b \mid b \in \mathbb{B}^{\mathrm{o}}\} = \emptyset.$$

A propositional formula $t_{\mathcal{P}}$, called a *propositional ADTerm*, is associated with every ADTerm t as follows. Let $t^1, t^2, \ldots, t^k \in \mathbb{T}_\Sigma$, $s \in \{\mathrm{p}, \mathrm{o}\}$ and $k > 0$. Then

$$\begin{aligned} &b_{\mathcal{P}} = x_b, \text{ for } b \in \mathbb{B}, && (\vee_k^s(t^1, \ldots, t^k))_{\mathcal{P}} = t^1_{\mathcal{P}} \vee \cdots \vee t^k_{\mathcal{P}}, \\ &(\mathrm{c}^s(t^1, t^2))_{\mathcal{P}} = t^1_{\mathcal{P}} \wedge \neg t^2_{\mathcal{P}}, && (\wedge_k^s(t^1, \ldots, t^k))_{\mathcal{P}} = t^1_{\mathcal{P}} \wedge \cdots \wedge t^k_{\mathcal{P}}. \end{aligned}$$

Every assignment of Boolean values (0 standing for *false* and 1 standing for *true*) to the propositional variables x_b, for $b \in \mathbb{B}$, which satisfies a propositional ADTerm $t_{\mathcal{P}}$, describes a way to achieve the proponent's goal represented by the ADTerm t.

EXAMPLE 3.3
Consider the ADTerm $t = \mathrm{c}^{\mathrm{p}}(b, \wedge^{\mathrm{o}}(d, e))$, where $b \in \mathbb{B}^{\mathrm{p}}$ and $d, e \in \mathbb{B}^{\mathrm{o}}$. The corresponding propositional ADTerm $t_{\mathcal{P}}$ is $x_b \wedge \neg(x_d \wedge x_e)$. The formula $t_{\mathcal{P}}$ is satisfied if and only if variable x_b is set to 1 and at least one of the variables x_d or x_e is set to 0. This models the fact that, to achieve his goal, the proponent needs to execute action b while at least one of the two actions d and e must not be executed by the opponent.

By $\approx$ we denote the canonical equivalence relation on propositional formulæ. Recall that two propositional formulæ ψ and ψ' are equivalent ($\psi \approx \psi'$) if and only if, for every assignment ν of Boolean values to propositional variables, we have $\nu(\psi) = \nu(\psi')$.

DEFINITION 3.4
The propositional semantics for ADTerms is the equivalence relation $\equiv_{\mathcal{P}}$ on $\mathbb{T}_\Sigma$ defined, for all $t, t' \in \mathbb{T}_\Sigma$, by

$$t \equiv_{\mathcal{P}} t' \text{ if and only if } t_{\mathcal{P}} \approx t'_{\mathcal{P}}.$$

The following example illustrates the use of the propositional semantics.

EXAMPLE 3.5
Consider the ADTerm $t = \mathrm{c}^{\mathrm{p}}(b, \wedge^{\mathrm{o}}(d, e))$, introduced in Example 3.3 and the ADTerm $t' = \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(b, b), \wedge^{\mathrm{o}}(d, e))$. Due to the idempotency of the propositional conjunction, the corresponding propositional ADTerms are equivalent formulæ, i.e. $t_{\mathcal{P}} = x_b \wedge \neg(x_d \wedge x_e) \approx (x_b \wedge x_b) \wedge \neg(x_d \wedge x_e) = t'_{\mathcal{P}}$. Therefore, we have $t \equiv_{\mathcal{P}} t'$.

3.3 Semantics induced by a De Morgan lattice

In the propositional semantics, ADTerms are interpreted as propositional formulæ. Such an interpretation limits the usefulness of the propositional semantics to those applications which take only binary properties into account. Examples of such properties are feasibility or presence of an attack. This implies that the propositional semantics is not well-suited to reason about properties, such as effectiveness or usefulness of an attack's components, which may have more than two states. To overcome this limitation of the propositional semantics, we propose the use of semantics induced by De Morgan lattices. In a semantics induced by a De Morgan lattice, ADTerms are interpreted as functions whose range is the De Morgan lattice.

Let $\langle A,+,\times\rangle$ be an algebraic structure defined over a non-empty set A with two binary operations $+$ and $\times$. The structure $\langle A,+,\times\rangle$ is called a *distributive lattice* if the operators $+$ and $\times$ are associative and commutative and if the following laws hold: $a\times(a+b)=a$, $a+(a\times b)=a$ (absorption) and $a\times(b+c)=(a\times b)+(a\times c)$ (distributivity of $\times$ over $+$). It is a basic fact in lattice theory that the last condition is equivalent to its dual, i.e. $a+(b\times c)=(a+b)\times(a+c)$. Furthermore, it is well-known that if $\langle A,+,\times\rangle$ is a lattice it can always be equipped with a canonical partial order, defined for all $a,b\in A$, by

$$a\preceq b \text{ if and only if } a+b=b. \tag{$\preceq$}$$

This order is monotonic with respect to the operations $+$ and $\times$, see [11].

To introduce the notion of a De Morgan lattice, we extend a distributive lattice $\langle A,+,\times\rangle$ with a unary operation, denoted by $\neg$, satisfying De Morgan's laws and double negation.

DEFINITION 3.6
An algebraic structure $\langle A,+,\times,\neg\rangle$ is called a De Morgan lattice if the sub-structure $\langle A,+,\times\rangle$ is a distributive lattice and, for all $a,b\in A$, we have

$$\neg(a+b)=(\neg a)\times(\neg b), \qquad \neg(a\times b)=(\neg a)+(\neg b), \qquad \neg(\neg a)=a.$$

We assume that every De Morgan lattice $\langle A,+,\times,\neg\rangle$ contains the neutral elements $\mathbf{0}$ for $+$ and $\mathbf{1}$ for $\times$. This is not a significant restriction, since using a result of Pouly [25], it can be easily shown that $\langle A,+,\times,\neg\rangle$ can always be adjoined with such elements.

De Morgan lattices are well-suited to represent outcomes requiring more than just the two Boolean values 0 and 1.

EXAMPLE 3.7
The tuple $\langle\{\mathtt{F},\mathtt{M},\mathtt{T}\},\max,\min,\neg\rangle$, where $\mathtt{F}\le\mathtt{M}\le\mathtt{T}$ and $\neg\mathtt{F}=\mathtt{T}$, $\neg\mathtt{M}=\mathtt{M}$ and $\neg\mathtt{T}=\mathtt{F}$ is a De Morgan lattice[1]. The values T, M and F, for instance, could allow us to distinguish between fully effective, partially effective and ineffective actions, respectively.

Furthermore, De Morgan lattices are more general than Boolean algebras, as shown in the following example.

EXAMPLE 3.8
Consider the De Morgan lattice $\langle\{\mathtt{T},\mathtt{M},\mathtt{F}\},\max,\min,\neg\rangle$ introduced in Example 3.7. Note that F and T are neutral elements for max and min, respectively. However, this De Morgan lattice is not a Boolean algebra. It does not satisfy the laws of complements, because $\max\{\mathtt{M},\neg\mathtt{M}\}=\mathtt{M}\neq\mathtt{T}$ and $\min\{\mathtt{M},\neg\mathtt{M}\}=\mathtt{M}\neq\mathtt{F}$.

We now introduce *De Morgan valuations* which are functions represented by ADTerms when a semantics induced by a De Morgan lattice is used. As in the case of the propositional semantics, we

[1] One can show that order $\le$ coincides with the canonical order given by ($\preceq$) for $+=\max$.

assign a propositional variable x_b to every action $b \in \mathbb{B}$. Given a set $V \subseteq \{x_b \mid b \in \mathbb{B}\}$, we denote by $\mathbf{x} \in \{0,1\}^V$ a function that associates a value $\mathbf{x}(x_b) \in \{0,1\}$ with every variable $x_b \in V$. In other words, every such function $\mathbf{x} \in \{0,1\}^V$ represents an assignment of Boolean values to the variables in V.

Definition 3.9
Let $\langle A, +, \times, \neg \rangle$ be a De Morgan lattice and let $V \subseteq \{x_b \mid b \in \mathbb{B}\}$ be a set of propositional variables. A De Morgan valuation f with domain V is a function $f\colon \{0,1\}^V \to A$ assigning a value $f(\mathbf{x}) \in A$ to each $\mathbf{x} \in \{0,1\}^V$.

Example 3.10
The propositional Boolean algebra $\langle \{0,1\}, \vee, \wedge, \neg \rangle$ is an example of a De Morgan lattice. In this case, the De Morgan valuations are simply Boolean functions, i.e. functions of the form $f\colon \{0,1\}^V \to \{0,1\}$.

Given a function $\mathbf{x} \in \{0,1\}^V$, we denote by $\mathbf{x}^{\downarrow W}$ the projection of $\mathbf{x}$ to a subset $W \subseteq V$. This notation allows us to define the sum and the product of De Morgan valuations. Let $\langle A, +, \times, \neg \rangle$ be a De Morgan lattice and let f and g be two De Morgan valuations with domains V and U, respectively. The *sum* of f and g (denoted by $f+g$) and the *product* of f and g (denoted by $f \times g$) are De Morgan valuations with domain $V \cup U$, defined for every $\mathbf{x} \in \{0,1\}^{V \cup U}$ by

$$(f+g)(\mathbf{x}) = f(\mathbf{x}^{\downarrow V}) + g(\mathbf{x}^{\downarrow U}) \quad \text{and} \quad (f \times g)(\mathbf{x}) = f(\mathbf{x}^{\downarrow V}) \times g(\mathbf{x}^{\downarrow U}).$$

The *negation* of the De Morgan valuation f (denoted by $\neg f$) is the De Morgan valuation with domain V, defined for every $\mathbf{x} \in \{0,1\}^V$ by $(\neg f)(\mathbf{x}) = \neg(f(\mathbf{x}))$.

Example 3.11
Consider the De Morgan lattice $\langle \{\mathtt{T}, \mathtt{M}, \mathtt{F}\}, \max, \min, \neg \rangle$ introduced in Example 3.7. Let $f\colon \{0,1\}^{\{y\}} \to \{\mathtt{T}, \mathtt{M}, \mathtt{F}\}$ and $g\colon \{0,1\}^{\{z\}} \to \{\mathtt{T}, \mathtt{M}, \mathtt{F}\}$ be two De Morgan valuations, given by

$$f(y=0) = \mathtt{F}, \qquad g(z=0) = \mathtt{F},$$
$$f(y=1) = \mathtt{M}, \qquad g(z=1) = \mathtt{T}.$$

Negations of f and g are defined as

$$\neg f(y=0) = \mathtt{T}, \qquad \neg g(z=0) = \mathtt{T},$$
$$\neg f(y=1) = \mathtt{M}, \qquad \neg g(z=1) = \mathtt{F}.$$

Table 3 illustrates the sum of f and g as well as their product. Note that, in this case $+ = \max$ and $\times = \min$.

To define a semantics induced by a De Morgan lattice $\langle A, +, \times, \neg \rangle$, we first associate, with every ADTerm t, a De Morgan valuation f_t. If $t = b$ and b is a basic action, then f_t is a De Morgan valuation with domain $\{x_b\}$, i.e. a function of the form $f_b\colon \{0,1\}^{\{x_b\}} \to A$. With the help of f_b, we express how the value assigned to action b changes, depending on whether this action is present ($x_b = 1$) or absent ($x_b = 0$). De Morgan valuations associated with composed ADTerms are then defined recursively, as

TABLE 3. Sum and product of two De Morgan valuations

y	z	$f+g=\max\{f,g\}$	$f\times g=\min\{f,g\}$
0	0	F	F
0	1	T	F
1	0	M	F
1	1	T	M

follows. For $s\in\{\mathrm{p},\mathrm{o}\}$, $k>0$, we set [2]

$$f_{\vee^s(t_1,\ldots,t_k)}=\sum_{i=1}^{k} f_{t_i}, \qquad f_{\wedge^s(t_1,\ldots,t_k)}=\prod_{i=1}^{k} f_{t_i}, \qquad f_{\mathrm{c}^s(t_1,t_2)}=f_{t_1}\times\neg f_{t_2}.$$

Note that the same De Morgan lattice may induce several semantics. In fact, each semantics induced by a De Morgan lattice is fully determined by a De Morgan lattice $\langle A,+,\times,\neg\rangle$ and a given set of De Morgan valuations $\{f_b\colon \{0,1\}^{\{x_b\}}\to A \mid b\in\mathbb{B}\}$. Modification of at least one De Morgan valuation f_b results in a different semantics induced by the lattice $\langle A,+,\times,\neg\rangle$.

The purpose of a semantics for ADTerms is to define which ADTerms are equivalent. This is achieved with the help of equivalent De Morgan valuations. Consider a De Morgan lattice $\langle A,+,\times,\neg\rangle$ and two subsets of propositional variables $V,U\subseteq\{x_b \mid b\in\mathbb{B}\}$. Two De Morgan valuations f and g, with respective domains V and U, are said to be *equivalent* (denoted by $f\equiv g$) if and only if, for every $\mathbf{x}\in\{0,1\}^{V\cup U}$, we have $f(\mathbf{x}^{\downarrow V})=g(\mathbf{x}^{\downarrow U})$.

DEFINITION 3.12
The semantics for ADTerms induced by a De Morgan lattice $\langle A,+,\times,\neg\rangle$ and a set of De Morgan valuations $\{f_b\colon \{0,1\}^{\{x_b\}}\to A \mid b\in\mathbb{B}\}$ is the equivalence relation $\equiv_{\mathcal{DM}}$ on $\mathbb{T}_\Sigma$ defined, for all $t,t'\in\mathbb{T}_\Sigma$, by

$$t\equiv_{\mathcal{DM}} t' \text{ if and only if } f_t\equiv f_{t'}.$$

Since every Boolean algebra satisfies the properties of a De Morgan lattice, the propositional semantics introduced in Section 3.2 is a semantics induced by a De Morgan lattice.

REMARK 3.13
The propositional semantics for ADTerms is the semantics induced by the Boolean algebra $\langle\{0,1\},\vee,\wedge,\neg\rangle$, where a basic action $b\in\mathbb{B}$ represents the Boolean function $f_b\colon \{0,1\}^{\{x_b\}}\to\{0,1\}$, given by $f_b(x_b=v)=v$, for $v\in\{0,1\}$.

We end this section with a discussion showing how a semantics induced by a De Morgan lattice different from the Boolean algebra $\langle\{0,1\},\vee,\wedge,\neg\rangle$ extends the expressive capabilities of the propositional semantics. Boolean functions interpreting basic actions in the propositional semantics are of the form $f_b(x_b=v)=v$. Such an interpretation does not allow us to differentiate between the execution of an action and its effectiveness. In other words, the propositional semantics assumes that actions which are executed are always fully effective. However, this is rarely the case in a real life scenario. For instance, the execution of a dictionary attack to guess a password does not guarantee that the password will be found. The following example illustrates how to more accurately model such an attack by using a semantics induced by the De Morgan lattice $\langle\{\mathtt{F},\mathtt{M},\mathtt{T}\},\max,\min,\neg\rangle$.

[2] $\sum$ and $\prod$ stand for extensions of sum and product of two valuations to any finite number of valuations. They are well-defined by associativity of $+$ and $\times$.

Example 3.14
Let us consider the De Morgan lattice introduced in Example 3.7 and let $t = c^{p}(b, \wedge^{o}(d, e))$ be the ADTerm in Example 3.3. We use De Morgan valuations to describe efficiency levels of actions b, d and e. We assume that when actions b, d and e are not executed ($x_i = 0$, $i \in \{b, d, e\}$), they are ineffective ($\mathtt{F}$)

$$f_b(x_b = 0) = \mathtt{F}, \qquad f_d(x_d = 0) = \mathtt{F}, \qquad f_e(x_e = 0) = \mathtt{F}.$$

Moreover, executing actions b and e ($x_i = 1, i \in \{b, e\}$) ensures their full effectiveness ($\mathtt{T}$), but executing action d guarantees its partial effectiveness only ($\mathtt{M}$)

$$f_b(x_b = 1) = \mathtt{T}, \qquad f_d(x_d = 1) = \mathtt{M}, \qquad f_e(x_e = 1) = \mathtt{T}.$$

Analysing the De Morgan valuation associated with t, given by $f_t(x_b, x_d, x_e) = \min \{f_b(x_b), \neg(\min\{f_d(x_d), f_e(x_e)\})\}$, allows us to reason about effectiveness of the scenario represented by t. We have

$$\begin{array}{llll} f_t(0,0,0) = \mathtt{F} & f_t(0,1,0) = \mathtt{F} & f_t(1,0,0) = \mathtt{T} & f_t(1,1,0) = \mathtt{T} \\ f_t(0,0,1) = \mathtt{F} & f_t(0,1,1) = \mathtt{F} & f_t(1,0,1) = \mathtt{T} & f_t(1,1,1) = \mathtt{M}. \end{array}$$

From $f_t^{-1}(\{\mathtt{M}, \mathtt{T}\})$, we deduce that the scenario is at least partially effective for the proponent if action b is executed, independent of actions d and e.

3.4 Multiset semantics

In every semantics considered so far, the refining symbols $\vee^{s}$ and $\wedge^{s}$, for $s \in \{\mathrm{p}, \mathrm{o}\}$, have been interpreted with idempotent operators. Therefore, all these semantics assume that the multiplicity of a sub-goal is irrelevant. This assumption, however, might not be intended in all applications of ADTrees. It might, for instance, depend on whether the components can be reused or not.

Example 3.15
Consider the scenario illustrated in Figure 1. To deprive the attacker of the possibilities to break in through the back door and to break in through the main door, the defender has to install locks on both doors. Since the two doors are in two physically distinct locations, a reuse of locks is not possible in this case.

The multiset semantics, introduced in this section, allows us to distinguish between multiple occurrences of the same actions. Thus, it is suitable for analysing scenarios in which such multiple occurrences of the same sub-goal are significant, as in Example 3.15. The multiset semantics has initially been defined for attack trees in [21]. Our construction extends this framework to ADTrees.

Given a set H, we use 2^{H} to denote the power set of H, and $\mathfrak{M}(H)$ to denote the set of all multisets of elements in H. We use $\{\!|a_1, \ldots, a_n|\!\}$ to denote a multiset composed of (not necessarily distinct) elements $a_1, \ldots, a_n$. The symbol $\uplus$ stands for the multiset union.

In the multiset semantics, ADTerms are interpreted as a set of pairs of the form $(P, O) \in \mathfrak{M}(\mathbb{B}^{\mathrm{p}}) \times \mathfrak{M}(\mathbb{B}^{\mathrm{o}})$, called *bundles*. A bundle (P, O) encodes how the proponent can achieve his goal: the proponent must perform all actions present in P whereas the opponent must not perform any of the actions in O. The set of bundles corresponding to an ADTerm t is an element of $2^{\mathfrak{M}(\mathbb{B}^{\mathrm{p}}) \times \mathfrak{M}(\mathbb{B}^{\mathrm{o}})}$, denoted by $t_{\mathcal{M}}$. It represents alternative possibilities for the proponent to achieve his goal. A basic action b of the proponent's type is interpreted as a singleton $b_{\mathcal{M}} = \{(\{\!|b|\!\}, \emptyset)\}$, because to achieve his goal it is sufficient for the proponent to execute action b. A basic action b of the opponent's type is interpreted as $b_{\mathcal{M}} = \{(\emptyset, \{\!|b|\!\})\}$, because in order for the proponent to be successful, action b must not be executed by the opponent. To obtain the multiset interpretation of the composed ADTerms, we use the union of sets of bundles ($\cup$) and the *distributive product* of sets of bundles ($\otimes$). The distributive product of two sets of bundles S and Z is defined as the 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 multiset interpretation $t_{\mathcal{M}}$ of a composed ADTerm t is then given by

$$\begin{aligned}
(\vee_k^{\mathrm{p}}(t^1, \ldots, t^k))_{\mathcal{M}} &= t^1_{\mathcal{M}} \cup \cdots \cup t^k_{\mathcal{M}}, & (\vee_k^{\mathrm{o}}(t^1, \ldots, t^k))_{\mathcal{M}} &= t^1_{\mathcal{M}} \otimes \cdots \otimes t^k_{\mathcal{M}},\\
(\wedge_k^{\mathrm{p}}(t^1, \ldots, t^k))_{\mathcal{M}} &= t^1_{\mathcal{M}} \otimes \cdots \otimes t^k_{\mathcal{M}}, & (\wedge_k^{\mathrm{o}}(t^1, \ldots, t^k))_{\mathcal{M}} &= t^1_{\mathcal{M}} \cup \cdots \cup t^k_{\mathcal{M}},\\
(\mathrm{c}^{\mathrm{p}}(t^1, t^2))_{\mathcal{M}} &= t^1_{\mathcal{M}} \otimes t^2_{\mathcal{M}}, & (\mathrm{c}^{\mathrm{o}}(t^1, t^2))_{\mathcal{M}} &= t^1_{\mathcal{M}} \cup t^2_{\mathcal{M}}.
\end{aligned}$$

Let t be an ADTerm and let t' be one of its sub-terms. Note that the set of bundles $t'_{\mathcal{M}}$ encodes how the *proponent of term t* can be successful in the situation described by sub-term t', regardless of the type of t'. In particular, to achieve a disjunctive goal, the proponent has to achieve *at least one* of the corresponding sub-goals. Similarly, to successfully prevent a conjunctive countermeasure of the opponent, it is sufficient for the proponent to prevent *at least one* of the corresponding sub-countermeasures. An analogous reasoning holds for a goal of the proponent which is conjunctively refined and a disjunctively refined countermeasure of the opponent. This is the reason why the operator used to define the multiset interpretation of a disjunctively refined goal for one player is the same as the operator used to define the multiset interpretation of a conjunctively refined goal for the other player.

Definition 3.16
The multiset semantics for ADTerms is an equivalence relation on $\mathbb{T}_\Sigma$, denoted by $\equiv_{\mathcal{M}}$ and defined for all $t, t' \in \mathbb{T}_\Sigma$ by

$$t \equiv_{\mathcal{M}} t' \text{ if and only if } t_{\mathcal{M}} = t'_{\mathcal{M}}.$$

Example 3.17 shows that the multiset semantics takes into account multiple occurrences of the same actions.

Example 3.17
The ADTerms $t = \mathrm{c}^{\mathrm{p}}(b, \wedge^{\mathrm{o}}(d,e))$ and $t' = \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(b,b), \wedge^{\mathrm{o}}(d,e))$ from Example 3.5 have been shown to be equivalent with respect to the propositional semantics. The multiset interpretation of t is $t_{\mathcal{M}} = \{(\{\!|b|\!\}, \{\!|d|\!\}), (\{\!|b|\!\}, \{\!|e|\!\})\}$ and the multiset interpretation of t' is $t'_{\mathcal{M}} = \{(\{\!|b,b|\!\}, \{\!|d|\!\}), (\{\!|b,b|\!\}, \{\!|e|\!\})\}$. Since $t_{\mathcal{M}} \neq t'_{\mathcal{M}}$, the ADTerms t and t' are not equivalent with respect to the multiset semantics.

By comparing Examples 3.5 and 3.17, we deduce that the partition of $\mathbb{T}_\Sigma$ defined by the multiset semantics does not coincide with the partition defined by the propositional semantics. A more detailed comparison of these two semantics is presented in Section 4.1.

3.5 Equational semantics

As discussed in previous sections, the choice of an appropriate semantics depends on the requirements imposed by the domain the ADTrees are applied to. Such requirements can frequently be modelled as mathematical properties. For example, if the order in which sub-goals of conjunctively refined goals are executed is irrelevant, we should model the conjunctive refinement using an operator which is commutative. Similarly, if executing the same action twice is in practice the same as executing it only once, the corresponding operator should be idempotent. In this section, we show how to construct a semantics for ADTerms which takes a given set of properties into account. The idea is to specify an equivalence relation on ADTerms through a set of equations expressing the desired properties. This approach covers a concept described by Mauw and Oostdijk in [21], 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.

Let $\mathrm{VAR} = \mathrm{VAR}^{\mathrm{p}} \cup \mathrm{VAR}^{\mathrm{o}}$ be a set of typed variables. We use capital letters such as X, X_i, Y, Y_i, to denote elements of VAR. We extend the set $\mathbb{T}_\Sigma$ to the set $\mathbb{T}_\Sigma^{\mathrm{VAR}}$ of typed ADTerms over the variables in VAR. An *equation* is a pair $(t,t') \in \mathbb{T}_\Sigma^{\mathrm{VAR}} \times \mathbb{T}_\Sigma^{\mathrm{VAR}}$, where t and t' have the same type. In the remainder of this article, equation (t,t') is denoted by $t = t'$. 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 define the set of syntactic consequences of E as the smallest subset of $\mathbb{T}_\Sigma^{\mathrm{VAR}} \times \mathbb{T}_\Sigma^{\mathrm{VAR}}$ containing E and being closed under reflexivity, symmetry, transitivity, substitutions and contexts. In other words, the equation $t = t'$ is a syntactic consequence of E (denoted by $E \vdash t = t'$) if it can be derived from E by using the following rules

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

In the following definition we introduce the notion of equational semantics for ADTerms.

Definition 3.18

The equational semantics for ADTerms induced by an algebraic specification (Σ, E) is the equivalence relation $\equiv_E$ on $\mathbb{T}_\Sigma$, defined by

$$t \equiv_E t' \text{ if and only if } E \vdash t = t'.$$

Example 3.19 illustrates the use of equational semantics.

Example 3.19

Let Sym_k denote the set of all bijections from $\{1,\dots,k\}$ to itself. Consider the equational semantics induced by an algebraic specification (Σ, E), where

$$E = \{\vee^{\mathrm{p}}(X_1,\dots,X_k) = \vee^{\mathrm{p}}(X_{\sigma(1)},\dots,X_{\sigma(k)}) \mid \sigma \in \mathrm{Sym}_k\}.$$

The equations in E encode the commutativity of the disjunctive operator for the proponent. Thus, for the ADTerms $t_1 = \vee^{\mathrm{p}}(a,b)$ and $t_2 = \vee^{\mathrm{p}}(b,a)$, we have $t_1 \equiv_E t_2$, i.e. t_1 and t_2 model the same situation when the semantics $\equiv_E$ is used. 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 modelled by E.

The importance of defining a semantics, given a set of equations, is twofold. First, equations allow us to encode many of the mathematical properties desired for analysis of ADTrees. Second, the equations in E model all possible transformations of ADTerms, which preserve the semantics $\equiv_E$. In the next section, we use the notion of equational semantics to axiomatize the propositional and the multiset semantics for ADTerms.

4 Axiomatization of semantics for ADTerms

4.1 Complete set of axioms

We start by providing a definition of a complete set of axioms for a semantics for ADTerms.

Definition 4.1
Let (Σ, E) be an algebraic specification and let $\equiv$ be a semantics for ADTerms. Set E is a complete set of axioms for the semantics $\equiv$ if and only if $\equiv$ is equal to the equational semantics induced by the algebraic specification (Σ, E).

Remark 4.2
It follows directly from Definition 4.1 that E is a complete set of axioms for the equational semantics induced by an algebraic specification (Σ, E).

The importance of a complete set of axioms for a semantics of ADTerms is manifold. First, having complete sets of axioms unifies the treatment of different semantics for ADTrees. Instead of having to argue within different domains, such as sets of multisets or propositional logics, we can reason with ADTerms over the AD-signature. Second, the equations of a complete set of axioms state important properties modelled by a semantics, as shown in Example 3.19. We see in Section 5 that this helps us to formally define how to quantitatively analyse attack–defense scenarios using attributes. Third, knowing a complete set of axioms is a crucial step in developing algorithms which assign unique representatives to every equivalence class arising from a semantics. This simplifies the development of a computer tool supporting the ADTree methodology. Finally, we can use complete sets of axioms to facilitate a comparison between different semantics. In the remainder of this section we take a closer look at this issue.

To decide whether properties of ADTerms interpreted using one semantics can be exported to reason about ADTerms within a different semantics, we need to compare the corresponding partitions of the set of ADTerms. To this end, we define the notions of finer and coarser semantics. Intuitively, given two semantics, we say that one is finer than the other if it partitions the set of ADTerms in a finer way.

Definition 4.3
Let $\equiv_1$ and $\equiv_2$ be two semantics for ADTerms. The semantics $\equiv_1$ is finer than the semantics $\equiv_2$ if and only if $\equiv_1 \subseteq \equiv_2$, i.e., for $t, t' \in \mathbb{T}_\Sigma$, $t \equiv_1 t' \Rightarrow t \equiv_2 t'$. If $\equiv_1$ is finer than $\equiv_2$, we say that $\equiv_2$ is coarser than $\equiv_1$.

The fact that ADTerms which are equivalent according to a semantics which is finer are also equivalent according to any semantics which is coarser, allows us to import properties of a finer semantics into any coarser semantics.

In general, given two semantics for ADTerms, it is not easy to decide whether they are comparable, and if so, which one is finer. However, this task may become trivial, if we are able to appropriately axiomatize both semantics using complete sets of axioms.

THEOREM 4.4
Let $\equiv_1$ and $\equiv_2$ be two semantics for ADTerms with complete sets of axioms E_1 and E_2, respectively. If $E_1 \subseteq E_2$, then $\equiv_1$ is finer than $\equiv_2$.

PROOF. An immediate consequence of $E_1 \subseteq E_2$ is that every equation derivable from E_1 is also derivable from E_2, which proves the theorem. ■

In Sections 4.2 and 4.3, we construct complete sets of axioms for the propositional and the multiset semantics, respectively. These sets help us to compare the two semantics. For instance, the idempotency laws hold in the propositional but not in the multiset semantics. The relationship between the propositional and the multiset semantics is captured by the following theorem.

THEOREM 4.5
The multiset semantics for ADTerms is finer than the propositional semantics for ADTerms.

PROOF. It is sufficient to consider the complete sets of axioms $E_{\mathcal{P}}$ for the propositional semantics and $E_{\mathcal{M}}$ for the multiset semantics, that we introduce in Theorems 4.6 and 4.9, respectively. We observe that $E_{\mathcal{M}} \subseteq E_{\mathcal{P}}$, which according to Theorem 4.4 finishes the proof.

■

To conclude this section, we remark that the propositional semantics is not finer than the multiset semantics, as shown by Examples 3.5 and 3.17. Thus, these two semantics are not equal.

4.2 *Complete set of axioms for $\equiv_{\mathcal{P}}$*

We give a complete set of axioms for the propositional semantics in Theorem 4.6. This set of axioms is then used to compare the propositional semantics to other semantics induced by De Morgan lattices, as shown in Theorem 4.8.

THEOREM 4.6
Let $s \in \{\mathrm{p}, \mathrm{o}\}$ and $X, Y, X_i, Y_j \in \mathrm{VAR}$, for $i, j \geq 1$ and $k, n \in \mathbb{N} \setminus \{0\}$. Moreover, let Sym_k denote the set of all bijections from $\{1, \ldots, k\}$ to itself. The following set of equations, denoted by $E_{\mathcal{P}}$, is a complete set of axioms for the propositional semantics.[3]

$$\vee^s(X_1,\ldots,X_k) = \vee^s(X_{\sigma(1)},\ldots,X_{\sigma(k)}),\ \forall\sigma \in \mathrm{Sym}_k \tag{E_1^s}$$

$$\wedge^s(X_1,\ldots,X_k) = \wedge^s(X_{\sigma(1)},\ldots,X_{\sigma(k)}),\ \forall\sigma \in \mathrm{Sym}_k \tag{E_2^s}$$

$$\vee^s(X_1,\ldots,X_k,\vee^s(Y_1,\ldots,Y_n)) = \vee^s(X_1,\ldots,X_k,Y_1,\ldots,Y_n) \tag{E_3^s}$$

$$\wedge^s(X_1,\ldots,X_k,\wedge^s(Y_1,\ldots,Y_n)) = \wedge^s(X_1,\ldots,X_k,Y_1,\ldots,Y_n) \tag{E_4^s}$$

$$\vee^s(X) = X \tag{E_5^s}$$

$$\wedge^s(X) = X \tag{E_6^s}$$

[3] Note that the set of axioms given in Theorem 4.6 is in fact an axiom scheme. This is unavoidable, because the AD-signature contains infinitely many function symbols modelled using unranked functions.

$$\vee^s(X,\wedge^s(X,X_1,\dots,X_k))=X \tag{E^s_7}$$

$$\wedge^s(X,\vee^s(X,X_1,\dots,X_k))=X \tag{E^s_8}$$

$$\vee^s(X,\wedge^s(X_1,\dots,X_k))=\wedge^s(\vee^s(X,X_1),\dots,\vee^s(X,X_k)) \tag{E^s_9}$$

$$\wedge^s(X,\vee^s(X_1,\dots,X_k))=\vee^s(\wedge^s(X,X_1),\dots,\wedge^s(X,X_k)) \tag{E^s_{10}}$$

$$\vee^s(X,X,X_1,\dots,X_k)=\vee^s(X,X_1,\dots,X_k) \tag{E^s_{11}}$$

$$\wedge^s(X,X,X_1,\dots,X_k)=\wedge^s(X,X_1,\dots,X_k) \tag{E^s_{12}}$$

$$\mathrm{c}^s(\vee^s(X_1,\dots,X_k),X)=\vee^s(\mathrm{c}^s(X_1,X),\dots,\mathrm{c}^s(X_k,X)) \tag{E^s_{13}}$$

$$\mathrm{c}^s(\wedge^s(X_1,\dots,X_k),X)=\wedge^s(\mathrm{c}^s(X_1,X),\dots,\mathrm{c}^s(X_k,X)) \tag{E^s_{14}}$$

$$\mathrm{c}^s(X,\vee^{\bar{s}}(X_1,\dots X_k))=\wedge^s(\mathrm{c}^s(X,X_1),\dots,\mathrm{c}^s(X,X_k)) \tag{E^s_{15}}$$

$$\mathrm{c}^s(X,\wedge^{\bar{s}}(X_1,\dots X_k))=\vee^s(\mathrm{c}^s(X,X_1),\dots,\mathrm{c}^s(X,X_k)) \tag{E^s_{16}}$$

$$\mathrm{c}^s(\mathrm{c}^s(X,X_1),X_2)=\mathrm{c}^s(X,\vee^{\bar{s}}(X_1,X_2)) \tag{E^s_{17}}$$

$$\mathrm{c}^s(X,\mathrm{c}^{\bar{s}}(X_1,X_2))=\vee^s(\mathrm{c}^s(X,X_1),\wedge^s(X,X_2)) \tag{E^s_{18}}$$

$$\vee^s(\mathrm{c}^s(X_1,Y),X_2,\dots,X_k)=\mathrm{c}^s(\vee^s(X_1,\dots,X_k),\mathrm{c}^{\bar{s}}(Y,\vee^s(X_2,\dots,X_k))) \tag{E^s_{19}}$$

$$\wedge^s(\mathrm{c}^s(X_1,Y),X_2,\dots,X_k)=\mathrm{c}^s(\wedge^s(X_1,\dots,X_k),Y) \tag{E^s_{20}}$$

$$\vee^s(\mathrm{c}^s(X,Y),X)=X \tag{E^s_{21}}$$

$$\wedge^s(\mathrm{c}^s(X,Y),X)=\mathrm{c}^s(X,Y). \tag{E^s_{22}}$$

PROOF. To prove Theorem 4.6, we define the notion of a complete set of axioms for a set of propositional formulæ. We transform the problem of finding a complete set of axioms for the propositional semantics into the problem of finding a complete set of axioms for the set of all propositional ADTerms. The outline of the remaining part of the proof runs as follows.

(1) By reformulating equations in $E_{\mathcal{P}}$, we define a complete set G of axioms for the set of propositional ADTerms.
(2) We show using axioms in G that every propositional ADTerm can be transformed into a disjunctive form.
(3) We transform the obtained disjunctive forms further into minimal disjunctive forms.
(4) We prove that these minimal disjunctive forms are unique modulo associativity and commutativity.

The above considerations help us to conclude that two ADTerms are equivalent with respect to the propositional semantics if and only if the minimal disjunctive forms for the corresponding propositional ADTerms are equal modulo associativity and commutativity. This finishes the proof of Theorem 4.6, due to the fact that each axiom in G constitutes a propositional interpretation of an axiom scheme in $E_{\mathcal{P}}$. ■

In the remainder of this section, we give details for Steps 1–4.

1. We first define a grammar that generates all propositional ADTerms, which are defined in Section 3.2. Let $X^G=\{x_b\,|\,b\in\mathbb{B}^{\mathrm{p}}\}$ and $Y^G=\{x_b\,|\,b\in\mathbb{B}^{\mathrm{o}}\}$ be two sets of propositional variables that correspond to basic actions in the propositional semantics. We have $X^G\cap Y^G=\emptyset$. Consider the propositional formulæ over $X^G\cup Y^G$ generated by the following grammar, denoted by $\mathcal{G}$, where

$x_{bi} \in X^G$, $y_{bj} \in Y^G$ and $\psi \star \phi = \psi \wedge \neg\phi$:

$$\begin{array}{rlcccccc} P: & x_{bi} & \mid & P \vee P & \mid & P \wedge P & \mid & P \star N \\ N: & y_{bj} & \mid & N \vee N & \mid & N \wedge N & \mid & N \star P. \end{array}$$

Often we write that $\psi \in \mathcal{G}$ if ψ is generated by $\mathcal{G}$. Thus, we abuse notation and let $\mathcal{G}$ denote both the grammar and the set of formulæ generated by the grammar. It is easy to see that $t \in \mathbb{T}^{\mathrm{p}}_{\Sigma}$ (resp. $\mathbb{T}^{\mathrm{o}}_{\Sigma}$) if and only if there exists a formula $P \in \mathcal{G}$ (resp. $N \in \mathcal{G}$), such that $t_{\mathcal{P}} = P$ (resp. $t_{\mathcal{P}} = N$) modulo associativity.

Let A be a set of equations of the form $\xi = \zeta$, where ξ and ζ are propositional formulæ and let $\mathcal{A}$ be a set of propositional formulæ. We say that A is a *complete set of axioms for* $\mathcal{A}$ if and only if two propositionally equivalent formulæ in $\mathcal{A}$ can be transformed into each other by applying substitutions and context to equations in A. The problem of finding a complete set of axioms for the propositional semantics can be reduced to finding a complete set of axioms for the set of propositional formulæ generated by $\mathcal{G}$.

Lemma 4.7 (Complete set of axioms for $\mathcal{G}$)
Let X, Y, Z be propositional variables. The following set G is a complete set of axioms for $\mathcal{G}$.[4]

$$\begin{array}{ll}
X \vee Y = Y \vee X & X \wedge Y = Y \wedge X \\
X \vee (Y \vee Z) = (X \vee Y) \vee Z & X \wedge (Y \wedge Z) = (X \wedge Y) \wedge Z \\
\vee(X) = X & \wedge(X) = X \\
X \vee (X \wedge Y) = X & X \wedge (X \vee Y) = X \\
X \vee (Y \wedge Z) = (X \vee Y) \wedge (X \vee Z) & X \wedge (Y \vee Z) = (X \wedge Y) \vee (X \wedge Z) \\
X \vee X = X & X \wedge X = X \\
(X \vee Y) \star Z = (X \star Z) \vee (Y \star Z) & (X \wedge Y) \star Z = (X \star Z) \wedge (Y \star Z) \\
X \star (Y \vee Z) = (X \star Y) \wedge (X \star Z) & X \star (Y \wedge Z) = (X \star Y) \vee (X \star Z) \\
(X \star Y) \star Z = X \star (Y \vee Z) & X \star (Y \star Z) = (X \star Y) \vee (X \wedge Z) \\
(X \star Y) \vee Z = (X \vee Z) \star (Y \star Z) & (X \star Y) \wedge Z = (X \wedge Z) \star Y \\
(X \star Y) \vee X = X & (X \star Y) \wedge X = (X \star Y).
\end{array}$$

Proof. For every axiom $\xi = \zeta$ in G, we have $\xi \approx \zeta$. Therefore, if a formula ψ' is obtained from a formula ψ by using axioms in G, we have $\psi \approx \psi'$. This proves soundness.

Proving completeness is done by showing that, using axioms in G, every formula $\psi \in \mathcal{G}$ can be transformed into a minimal disjunctive form, denoted by $\mathrm{mdf}(\psi)$. In Steps 2–4 below, we prove that this minimal disjunctive form is unique up to commutativity and associativity of $\vee$ and $\wedge$, denoted by $=_{AC}$. In other words, we show that, for $\psi, \psi' \in \mathcal{G}$,

$$\psi \approx \psi' \text{ if and only if } \mathrm{mdf}(\psi) =_{AC} \mathrm{mdf}(\psi') \quad (1)$$

holds. ■

[4] Note that contrary to the set $E_{\mathcal{P}}$, the set G is finite. The reduction of the number of equations is made possible, because the unranked function symbols $\vee^s$ and $\wedge^s$, for $s \in \{\mathrm{p}, \mathrm{o}\}$, are interpreted with the associative operators $\vee$ and $\wedge$.

2. Note that for all $P, N \in \mathcal{G}$, we have that $P \not\approx N$, because $X^G \cap Y^G = \emptyset$. To define minimal disjunctive forms for the formulæ in $\mathcal{G}$, we first introduce a grammar $\mathcal{B}$ generating propositional formulæ in disjunctive form, and we show that every formula generated by $\mathcal{G}$ can be transformed, using axioms in G, into an equivalent formula in disjunctive form, generated by $\mathcal{B}$. We later use these disjunctive forms to obtain minimal forms.

Let the following grammar be denoted by $\mathcal{B}$:

$$\begin{array}{llllll}
K^P & : x_{bi} & \mid & K^P \wedge K^P & & \\
D^N & : y_{bj} & \mid & D^N \vee D^N & & \\
B^P & : K^P & \mid & K^P \star D^N & \mid & B^P \vee B^P \\
 & & & & & \\
K^N & : y_{bj} & \mid & K^N \wedge K^N & & \\
D^P & : x_{bi} & \mid & D^P \vee D^P & & \\
B^N & : K^N & \mid & K^N \star D^P & \mid & B^N \vee B^N.
\end{array}$$

It is clear that every formula generated by $\mathcal{B}$ is also generated by $\mathcal{G}$. To prove the converse, we show that, for every $P \in \mathcal{G}$ (resp. $N \in \mathcal{G}$), there exists an equivalent disjunctive formula, denoted by $\mathrm{df}(P)$ (resp. $\mathrm{df}(N)$) of the form $B^P \in \mathcal{B}$ (resp. $B^N \in \mathcal{B}$). The formula $\mathrm{df}(P)$ (resp. $\mathrm{df}(N)$) is obtained from P (resp. N) by using axioms in G. This is proven by induction on the structure of P (resp. N), by using the following two statements, for $S \in \{P, N\}$.

- If $B_1^S, B_2^S \in \mathcal{B}$, then $B_1^S \wedge B_2^S$ can be transformed, using axioms in G, into a formula of the form $B^S \in \mathcal{B}$.
- If $B_1^S, B_2^S \in \mathcal{B}$, then $B_1^S \star B_2^S$ can be transformed, using axioms in G, into a formula of the form $B^S \in \mathcal{B}$.

The statements themselves can easily be proven by structural induction. The technical details are omitted.

Let $I \subseteq \mathbb{N}$ be a non-empty, finite index set. From $\mathcal{B}$ we see that every formula $\mathrm{df}(P)$ is in the following disjunctive form

$$\mathrm{df}(P) = B^P = \bigvee_{k \in I} \alpha_k \star \beta_k,$$

where $\alpha_k = (x_{bk_1} \wedge \cdots \wedge x_{bk_u})$ and $\beta_k = (y_{bk_1} \vee \cdots \vee y_{bk_l})$, for some $x_{bk_1}, \ldots, x_{bk_u} \in X^G$, $y_{bk_1}, \ldots, y_{bk_l} \in Y^G$, $k_u \geq 1$ and $k_l \geq 0$. (A similar disjunctive form exists for N^P.)

3. Our goal is now to minimize the obtained disjunctive forms. We show that, using axioms in G, we can transform every $P \in \mathcal{G}$ into an equivalent formula, denoted by $\mathrm{mdf}(P)$, which is of the form

$$\mathrm{mdf}(P) = \bigvee_{k \in I} \alpha_k \star \beta_k \tag{2}$$

where $\alpha_k = (x_{bk_1} \wedge \cdots \wedge x_{bk_u}), \beta_k = (y_{bk_1} \vee \cdots \vee y_{bk_l})$, and for all $k, k' \in I$, $k \neq k'$ we have $\alpha_{k'} \star \beta_{k'}$ does not imply $\alpha_k \star \beta_k$ and for $i \neq j$ we have $x_{bk_i} \neq x_{bk_j}$ and $y_{bk_i} \neq y_{bk_j}$. We proceed by contraposition. We already know that, by using axioms in G, every $P \in \mathcal{G}$ can be transformed into the formula $\mathrm{df}(P) = B^P \in \mathcal{B}$. Assume that $\mathrm{df}(P) = \bigvee_{k \in I} \alpha_k \star \beta_k$ is not minimal. This means that either there exist $k, k' \in I$, such that $\alpha_{k'} \star \beta_{k'}$ implies $\alpha_k \star \beta_k$ or there exists $k \in I$ and $i \neq j$, such that $x_{bk_i} = x_{bk_j}$ or $y_{bk_i} = y_{bk_j}$. In the latter case, we minimize the formula with the help of the idempotency axiom. From [18], we know that every P represents a monotone Boolean function, hence the former case may only happen if, for $\alpha_k=(x_{bk_1}\wedge\cdots\wedge x_{bk_u})$, $\beta_k=(y_{bk_1}\vee\cdots\vee y_{bk_l})$, $\alpha_{k'}=(x_{bk'_1}\wedge\cdots\wedge x_{bk'_u})$ and $\beta_{k'}=(y_{bk'_1}\vee\cdots\vee y_{bk'_l})$, it holds that $\{x_{bk_1},\ldots,x_{bk_u}\}\subseteq\{x_{bk'_1},\ldots,x_{bk'_u}\}$ and $\{y_{bk_1},\ldots,y_{bk_l}\}\subseteq\{y_{bk'_1},\ldots,y_{bk'_l}\}$. We now sketch how to minimize $\mathrm{df}(P)$, using axioms in G, in all possible cases. For ease of notation, we assume that $\alpha_k=x_b$, $\alpha_{k'}=x_b\wedge x_{b'}$, $\beta_k=y_b$ and $\beta_{k'}=y_b\vee y_{b'}$, unless otherwise stated.

(a) If $\alpha_k\star\beta_k=\alpha_{k'}\star\beta_{k'}$, then $(\alpha_k\star\beta_k)\vee(\alpha_{k'}\star\beta_{k'})$ can be reduced to $\alpha_{k'}\star\beta_{k'}$ by using idempotency of $\vee$.

(b) If $\{y_{bk_1},\ldots,y_{bk_l}\}\neq\emptyset$, the following scheme can be used:

$$\begin{aligned}
&(\alpha_k\star\beta_k)\vee(\alpha_{k'}\star\beta_{k'})=\\
&=(x_b\star y_b)\vee((x_b\wedge x_{b'})\star(y_b\vee y_{b'}))\\
&=(x_b\star y_b)\vee((x_b\star(y_b\vee y_{b'}))\wedge(x_{b'}\star(y_b\vee y_{b'})))\\
&=(x_b\star y_b)\vee((x_b\star y_b)\wedge((x_b\star y_{b'})\wedge(x_{b'}\star y_b)\wedge(x_{b'}\star y_{b'})))\\
&=(x_b\star y_b)=\alpha_k\star\beta_k.
\end{aligned}$$

(c) If $\{y_{bk_1},\ldots,y_{bk_l}\}=\emptyset$ and $\{y_{bk'_1},\ldots,y_{bk'_l}\}\neq\emptyset$, the following scheme can be used:

$$\begin{aligned}
\alpha_k\vee(\alpha_{k'}\star\beta_{k'})&=x_b\vee((x_b\wedge x_{b'})\star y_b)=x_b\vee((x_b\star y_b)\wedge(x_{b'}\star y_b))\\
&=(x_b\vee(x_b\star y_b))\wedge(x_b\vee(x_{b'}\star y_b))=x_b\wedge(x_b\vee(x_{b'}\star y_b))\\
&=x_b=\alpha_k.
\end{aligned}$$

(d) If $\{y_{bk_1},\ldots,y_{bk_l}\}=\emptyset$ and $\{y_{bk'_1},\ldots,y_{bk'_l}\}=\emptyset$, the following scheme can be used:

$$\alpha_k\vee\alpha_{k'}=x_b\vee(x_b\wedge x_{b'})=x_b=\alpha_k.$$

4. It remains to be shown that two minimal disjunctive forms are propositionally equivalent if and only if they are equal modulo associativity and commutativity. This follows from the fact that formulæ generated by grammar $\mathcal{G}$ represent monotone Boolean functions, which have a unique minimal disjunctive normal form (DNF) representation modulo associativity and commutativity (see [10]). Hence formulæ in minimal disjunctive forms are in fact unique modulo associativity and commutativity. This ends the proof of (1) and hence the proof of Lemma 4.7.

The complete set of axioms $E_{\mathcal{P}}$ introduced in Theorem 4.6 allows us to compare the propositional semantics with other semantics induced by De Morgan lattices.

THEOREM 4.8
Let $\equiv_{\mathcal{P}}$ be the propositional semantics and let $\equiv_{\mathcal{DM}}$ be a semantics induced by a De Morgan lattice. The propositional semantics is finer than $\equiv_{\mathcal{DM}}$.

PROOF. It is sufficient to notice that every equation in the complete set of axioms $E_{\mathcal{P}}$ for the propositional semantics is also valid for $\equiv_{\mathcal{DM}}$. According to Theorem 4.4, this proves that $\equiv_{\mathcal{P}}$ is finer than $\equiv_{\mathcal{DM}}$. ■

In other words, Theorem 4.8 states that the propositional semantics is the finest among all semantics induced by De Morgan lattices.

4.3 Complete sets of axioms for $\equiv_{\mathcal{M}}$

In Theorem 4.9, we give a complete set of axioms for the multiset semantics. We employ a standard proof strategy by transforming the equations into a rewriting system and showing its strong termination as well as confluence. The proof mainly discusses ADTerms before linking ADTerms to the multiset semantics. The proof is constructive in that it can easily be turned into an algorithm that assigns a unique representative to every equivalence class defined by the multiset semantics.

THEOREM 4.9
Let the prerequisites of Theorem 4.6 hold. The following set, denoted by $E_{\mathcal{M}}$, is a complete set of axioms for the multiset semantics

$$\{(E_1^s),(E_2^s),(E_3^s),(E_4^s),(E_5^s),(E_6^s),(E_9^{\mathrm{o}}),(E_{10}^{\mathrm{p}}),(E_{11}^{\mathrm{p}}),(E_{12}^{\mathrm{o}}),$$
$$(E_{13}^{\mathrm{p}}),(E_{16}^{\mathrm{p}}),(E_{17}^{\mathrm{p}}),(E_{18}^{\mathrm{p}}),(E_{19}^{\mathrm{o}}),(E_{20}^{\mathrm{p}})\}.$$

Note that contrary to the equations in the set $E_{\mathcal{P}}$, some of the equations in $E_{\mathcal{M}}$ only hold for either the proponent, e.g. (E_{10}^{p}), or the opponent, e.g. (E_9^{o}).

PROOF. We make use of class rewriting by setting up an equational rewriting system. Then, we show that the system is strongly terminating and class confluent, which guarantees that the system has unique normal forms modulo the given equations, see [24]. Finally we show how the normal forms can be used to prove completeness of the axioms. The proof is structured into the following steps:

(1) We transform $E_{\mathcal{M}}$ into an equational term rewriting system (ETRS) R.
(2) We provide expressions which describe the normal forms of R.
(3) We show strong termination of R.
(4) We show confluence of R.
(5) We prove that the expressions given in Step 2 describe the normal forms of R.
(6) Using the normal forms, we show that the multiset semantics ($\equiv_{\mathcal{M}}$) is equal to the equational semantics induced by $E_{\mathcal{M}}$ ($\equiv_{E_{\mathcal{M}}}$).

(1) To define the ETRS, see [14], we divide the equations in $E_{\mathcal{M}}$ into two parts. Equations (E_1^s)–(E_6^s) express commutativity and associativity of the operators $\vee^s$ and $\wedge^s$ and serve in our system as equations. The remaining ten equations we turn into rewrite rules by directing them from left to right. By R we denote the ETRS composed of equations (E_1^s) – (E_6^s) and directed rewrite rules corresponding to the equations $(E_9^{\mathrm{o}}),(E_{10}^{\mathrm{p}}),(E_{11}^{\mathrm{p}}),(E_{12}^{\mathrm{o}}),(E_{13}^{\mathrm{p}}),(E_{16}^{\mathrm{p}}),(E_{17}^{\mathrm{p}}),(E_{18}^{\mathrm{p}}),(E_{19}^{\mathrm{o}}),(E_{20}^{\mathrm{p}})$.
(2) We introduce the operator C^{p} to ease notation. Let $M=\{\!|t_1,\ldots,t_m|\!\}$, $t_i\in\mathbb{T}_\Sigma^{\mathrm{p}}$, for $i\in\{1,\ldots,m\}$ and $m\in\mathbb{N}\setminus\{0\}$, be a multiset of ADTerms of proponent type and $M'=\{\!|t'_1,\ldots,t'_l|\!\}$, $t'_j\in\mathbb{T}_\Sigma^{\mathrm{o}}$, for $j\in\{1,\ldots,l\}$ and $l\in\mathbb{N}$, be a multiset of ADTerms of opponent type. The operator C^{p} is defined by

$$\begin{aligned}\mathrm{C}^{\mathrm{p}}\colon \mathfrak{M}(\mathbb{T}_\Sigma^{\mathrm{p}})\times\mathfrak{M}(\mathbb{T}_\Sigma^{\mathrm{o}}) &\to \mathbb{T}_\Sigma^{\mathrm{p}},\\ (M,M') &\mapsto \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(t_1,\ldots,t_m),\vee^{\mathrm{o}}(t'_1,\ldots,t'_l)).\end{aligned}$$

With the help of this operator, we define expressions which serve as normal forms for R. Let $I\subseteq\mathbb{N}$ be a nonempty index set. For every $k\in I$, let B_k be a finite multiset of basic actions of the proponent, such that $|B_k|\geq 1$, and let C_k be a finite multiset of basic actions of the opponent, such that $|C_k|\geq 0$. Then, the following expressions represent ADTerms which are in normal form with respect to R

$$\bigvee_{k\in I}^{\mathrm{p}} \mathrm{C}^{\mathrm{p}}(B_k,C_k), \tag{3}$$

where $\bigvee^{\mathrm{p}}$ represents the unranked function symbol $(\vee_k^{\mathrm{p}})_{k\in I}$. Moreover, we require that, for $k\neq k'$, we have $(B_k,C_k)\neq(B_{k'},C_{k'})$.

```
(VAR x y z)
(THEORY (AC a b c d))
(RULES

g(v(x, y)) → a(g(x), g(y))                 h(v(x, y)) → b(h(x), h(y))
k(v(x, y)) → c(k(x), k(y))                 l(v(x, y)) → d(l(x), l(y))
g(u(x)) → x                                h(u(x)) → x
k(u(x)) → x                                l(u(x)) → x

c(x, a(y, z)) → a(c(x, y), c(x, z))        b(x, d(y, z)) → d(b(x, y), b(x, z))
a(x, x) → x                                d(x, x) → x

e(e(x, y), z) → e(x, b(y, z))              c(x, e(y, z)) → e(c(x, y), z)
e(a(x, y), z) → a(e(x, z), e(y, z))        e(x, d(y, z)) → a(e(x, y), e(x, z))
e(x, f(y, z)) → a(e(x, y), c(x, z))        b(x, f(y, z)) → f(b(x, y), e(z, x))
)
```

FIGURE 2. ETRS in WST syntax.

(3) We prove strong termination of R with the help of the AProVE tool [1]. AProVE can handle equational TRS, but it cannot handle unranked functions. To overcome this problem, we use currying (see [30]). We create curried versions of $\vee^s$ and $\wedge^s$, which are unary functions, denoted as $\vee^s_{\mathrm{cu}}$ and $\vee^s_{\mathrm{cu}}$, respectively. A specific list of arguments of an unranked function would, e.g., be encoded in the following way: for $\vee^s(a,b,c)$, we write $\vee^s_{\mathrm{cu}}(v(v(u(a),u(b)),u(c)))$. Therefore, due to currying, we add the following rewrite rules $\vee^s_{\mathrm{cu}}(v(x,y)) \rightarrow \vee^s(\vee^s_{\mathrm{cu}}(x), \vee^s_{\mathrm{cu}}(y))$ and $\vee^s_{\mathrm{cu}}(u(x)) \rightarrow u$ and similar rules for $\wedge^s$.

We input the equational TRS in AProVE using the Workshop on Termination (WST) syntax (see [20]). Due to input restrictions in AProVE, the syntax, given in Figure 2, uses the transformations $\mathtt{a}=\vee^{\mathrm{p}}, \mathtt{b}=\vee^{\mathrm{o}}, \mathtt{c}=\wedge^{\mathrm{p}}, \mathtt{d}=\wedge^{\mathrm{o}}, \mathtt{e}=\mathrm{c}^{\mathrm{p}}, \mathtt{f}=\mathrm{c}^{\mathrm{o}}, \mathtt{g}=\vee^{\mathrm{p}}_{\mathrm{cu}}, \mathtt{h}=\vee^{\mathrm{o}}_{\mathrm{cu}}, \mathtt{k}=\wedge^{\mathrm{p}}_{\mathrm{cu}}, \mathtt{l}=\wedge^{\mathrm{o}}_{\mathrm{cu}}$.

The line 'THEORY' together with the first four lines of rewrite rules correspond to equations (E^s_1)–(E^s_6) and represent associativity, commutativity and currying of the operators $\vee^s$ and $\wedge^s$. The other rules correspond to the remaining ten equations. Strong termination of R is shown by multiple application of polynomial interpretation and removal of redundant rewrite rules.

(4) To prove confluence of R, it suffices to show that all critical pairs are joinable. We prove the joinability with the help of the tool TTT2, see [19]. Unfortunately TTT2 cannot handle equational rewriting. We circumvent this problem by adding one rewrite rule for commutativity and two for associativity for each of the binary operators, as shown in Figure 3. As output we obtain that all critical pairs are joinable.

```
a(x, y) → a(y, x)   a(x, a(y, z)) → a(a(x, y), z)    a(a(x, y), z) → a(x, a(y, z))
b(x, y) → b(y, x)   b(x, b(y, z)) → b(b(x, y), z)    b(b(x, y), z) → b(x, b(y, z))
c(x, y) → c(y, x)   c(x, c(y, z)) → c(c(x, y), z)    c(c(x, y), z) → c(x, c(y, z))
d(x, y) → d(y, x)   d(x, d(y, z)) → d(d(x, y), z)    d(d(x, y), z) → d(x, d(y, z))
```

FIGURE 3. Additional rewrite rules, due to conversion of equation into rewrite rules.

(5) We show that all ADTerms represented by (3) are irreducible and that all other ADTerms are reducible.

If the symbol $\vee^{\mathrm{p}}$ exists in an ADTerm represented by (3), it is always the head symbol. For these ADTerms, it is easily seen that, the only rewrite rule corresponding to equation (E_{11}^{p}) has $\vee^{\mathrm{p}}$ as the head symbol on the left-hand side. However, an ADTerm can only be rewritten using this rewrite rule if the arguments of $\vee^{\mathrm{p}}$ are not distinct, which is specifically excluded in the ADTerms represented by (3).

The ADTerms represented by (3) that do not contain $\vee^{\mathrm{p}}$, have either $\wedge^{\mathrm{p}}$ or c^{p} as the head symbol. In the former case, the expressions are in normal form. In the latter case, the rewrite rules corresponding to equations (E_{13}^{p}), (E_{16}^{p}), (E_{17}^{p}) and (E_{18}^{p}) may be applicable, because they contain $\vee^{\mathrm{p}}$. In these rules the other occurring operators are $\mathrm{c}^{\mathrm{p}}, \vee^{\mathrm{p}}, \wedge^{\mathrm{o}}$ and c^{o}, respectively. None of these, however, appear in the ADTerms represented by (3). Hence, we conclude that none of the ADTerms represented by (3) can be rewritten.

Now we show that every ADTerm, which is not represented by (3), can be rewritten. First, we remark that the ADTerms represented by (3) are of proponent type only. Hence, if an ADTerm is of opponent type, we know that it is a sub-term of an ADTerm of proponent type. Since the c^{p} is the only operator that takes an ADTerm of opponent type and outputs an ADTerm of proponent type, we know that if we discover a sub-term of opponent type, the complete ADTerm must contain c^{p}. We classify the ADTerms with respect to the number of (not necessarily distinct) constants they contain.

ADTerms with one constant can be divided into two classes: ADTerms of proponent type and ADTerms of opponent type. The former are in normal form, the latter do not represent any ADTree, due to our remark above.

ADTerms with two constants we also subdivide into ADTerms of opponent type and ADTerms of proponent type. The ADTerms in the first class do not represent any ADTree. In the second class, an ADTerm is either in normal form or it is of the form $\vee^{\mathrm{p}}(b,b)$. Then, the rewrite rule corresponding to equation (E_{11}^{p}) can be used.

ADTerms with three or more constants are classified as follows:

(a) ADTerms that contain c^{o} or $\wedge^{\mathrm{o}}$ are either of opponent type or can be rewritten.

(b) If an ADTerm contains a nested $\vee^{\mathrm{p}}$, it is either of opponent type or it can be rewritten or its head symbol is $\vee^{\mathrm{p}}$.

(c) If an ADTerm contains a c^{p} which is not preceded by a $\vee^{\mathrm{p}}$ operator, it is either of opponent type or can be rewritten.

(d) All remaining ADTerms are in normal form or contain only the same functional symbol $\circ$ in $\{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}, \vee^{\mathrm{o}}\}$. In the case were $\circ = \vee^{\mathrm{o}}$ the ADTerm is of opponent type, in case $\circ = \wedge^{\mathrm{p}}$ it is in normal form and in case $\circ = \vee^{\mathrm{p}}$ it can be rewritten if and only if the rewrite rule corresponding to equation (E_{11}^{p}) can be applied.

(6) Together Steps 1–5 show that R is a convergent ETRS with the unique normal forms represented by (3). It remains to be shown that the equations $E_{\mathcal{M}}$, are sound and complete with respect to the multiset semantics. We can easily verify soundness by proving that every equation in $E_{\mathcal{M}}$ holds in the multiset semantics. In other words, for all $t, t' \in \mathbb{T}_{\Sigma}^{\mathrm{p}}$, it holds that from $t \equiv_{E_{\mathcal{M}}} t'$ it follows that $t_{\mathcal{M}} = t'_{\mathcal{M}}$. This is essentially due to the fact $\langle \mathfrak{M}, \cup, \otimes \rangle$ forms a semi-ring. For example, equation (E_{20}^{p}) concretely yields

$$\wedge^{\mathrm{p}}(x, \mathrm{c}^{\mathrm{p}}(y,z))_{\mathcal{M}} = x_{\mathcal{M}} \otimes (y_{\mathcal{M}} \otimes z_{\mathcal{M}}) = (x_{\mathcal{M}} \otimes y_{\mathcal{M}}) \otimes z_{\mathcal{M}} = \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(x,y),z)_{\mathcal{M}}$$

All other cases are similar.

Finally, we prove completeness by showing that, for $t,t' \in \mathbb{T}^{\mathrm{p}}_{\Sigma}$, from $t_{\mathcal{M}} = t'_{\mathcal{M}}$ it follows that $t \equiv_{E_{\mathcal{M}}} t'$. To facilitate reasoning, let NF(t) and NF(t') denote the normal forms obtained by R, given as expression in (3), in other words, NF(t) $\equiv_{E_{\mathcal{M}}} t$ and NF(t') $\equiv_{E_{\mathcal{M}}} t'$. Since the elements considered for the multiset semantics are sets of pairs of multisets, there exists a 1–1 correspondence between such sets of pairs of multisets and the normal forms given by the ADTerm represented by the expression given in (3): each pair of multisets is mapped to a pair (B_k, C_k) which corresponds to ADTerms in normal form, as shown in Steps 2–5. The pairs (B_k, C_k) are mutually different for different indexes k, because we map sets which do not have multiple occurrences of the same element. We conclude that given $t_{\mathcal{M}} = t'_{\mathcal{M}}$, it follows that NF($t$) = NF($t'$). Consequently, from $t_{\mathcal{M}} = t'_{\mathcal{M}}$, it follows that $t \equiv_{E_{\mathcal{M}}} \mathrm{NF}(t) = \mathrm{NF}(t') \equiv_{E_{\mathcal{M}}} t'$, which concludes the proof of Theorem 4.9. ■

5 Attributes

5.1 Bottom-up evaluation

Attributes are used to quantitatively analyse attack–defense scenarios represented by ADTerms. An attribute expresses a particular property of a scenario, such as the minimal cost of an attack or the expected impact of a defensive measure. Schneier [29] sketched an intuitive bottom-up algorithm for calculating the value of an attribute on an attack tree. His procedure was formalized by Mauw and Oostdijk [21]. In this section, we extend the bottom-up approach for evaluation fo attributes to ADTerms. We start by introducing the notion of an attribute domain which formally specifies an attribute.

Definition 5.1
An attribute domain for ADTerms is a tuple

$$A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha, \vee^{\mathrm{o}}_\alpha, \wedge^{\mathrm{o}}_\alpha, \mathrm{c}^{\mathrm{p}}_\alpha, \mathrm{c}^{\mathrm{o}}_\alpha),$$

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

- $\vee^s_\alpha, \wedge^s_\alpha$ are unranked functions on D_α,
- c^s are binary functions on D_α.

Example 5.2
Attribute domain $A_{\mathrm{sat}} = (\{0,1\}, \vee, \wedge, \vee, \wedge, \star, \star)$, where $x \star y = x \wedge \neg y$, for all $x, y \in \{0,1\}$, can be used to decide whether the proponent's goal modelled by the root of an ADTerm t is satisfied or not.

Since attack trees only have one type of nodes—the proponent's nodes—an attribute domain in case of attack trees is a triple $A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha)$. In this case, the bottom-up evaluation of an attribute works as follows: first the values in D_α are assigned to the leaf nodes of an attack tree and then the values for the remaining nodes are deduced in a bottom-up way, with the help of the operations $\vee^{\mathrm{p}}_\alpha$ and $\wedge^{\mathrm{p}}_\alpha$. To extend this procedure to ADTrees, we first assign values in D_α to all non-refined nodes of an ADTree, then we compute the values corresponding to all its sub-trees by using the operations $\vee^s_\alpha, \wedge^s_\alpha$ and c^s_α for $s \in \{\mathrm{p}, \mathrm{o}\}$.

Let $A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha, \vee^{\mathrm{o}}_\alpha, \wedge^{\mathrm{o}}_\alpha, \mathrm{c}^{\mathrm{p}}_\alpha, \mathrm{c}^{\mathrm{o}}_\alpha)$ be an attribute domain for ADTerms. We now formalize the bottom-up computation of attribute values on ADTerms. A function $\beta_\alpha : \mathbb{B} \to D_\alpha$, which assigns to every basic action a value in the set D_α, is called a *basic assignment*. The function $\alpha\colon \mathbb{T}_\Sigma \to D_\alpha$, which assigns to every ADTerm t the value of an attribute, is defined recursively as follows

$$\alpha(t) = \begin{cases} \beta_\alpha(t), & \text{if } t \in \mathbb{B}, \\ \vee^s_\alpha(\alpha(t_1), \ldots, \alpha(t_k)), & \text{if } t = \vee^s(t_1, \ldots, t_k), \\ \wedge^s_\alpha(\alpha(t_1), \ldots, \alpha(t_k)), & \text{if } t = \wedge^s(t_1, \ldots, t_k), \\ \mathrm{c}^s_\alpha(\alpha(t_1), \alpha(t_2)), & \text{if } t = \mathrm{c}^s(t_1, t_2), \end{cases} \tag{4}$$

where $s \in \{\mathrm{p}, \mathrm{o}\}$ and $k > 0$. The following example illustrates the bottom-up evaluation of attribute values.

EXAMPLE 5.3
Consider the ADTerm $t = \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(a,b), \mathrm{c}^{\mathrm{o}}(d,e))$, where $a,b,e \in \mathbb{B}^{\mathrm{p}}$ and $d \in \mathbb{B}^{\mathrm{o}}$. The objective is to calculate the proponent's minimal cost necessary to achieve his goal. We use the attribute domain $A_{\mathrm{cost}} = (\mathbb{R}^+ \cup \{+\infty\}, \min, +, +, \min, +, \min)$ and the following basic assignment: $\beta_{\mathrm{cost}}(a) = 5$, $\beta_{\mathrm{cost}}(b) = 7$, $\beta_{\mathrm{cost}}(e) = 6$ and $\beta_{\mathrm{cost}}(d) = +\infty$. These values express the minimal investment required of the proponent to execute the corresponding action. Since the opponent's basic action d is not under control of the proponent, we set $\beta_{\mathrm{cost}}(d) = +\infty$. By countering the opponent's action d with the proponent's action e, and by using appropriate operators $\mathrm{c}^{\mathrm{p}}_{\mathrm{cost}} = +$ and $\mathrm{c}^{\mathrm{o}}_{\mathrm{cost}} = \min$, we can compute the actual minimal cost for the proponent to succeed in the scenario. Using function $\alpha = \mathrm{cost}$, as defined by (4), we calculate the proponent's minimal cost in the scenario as follows: $\mathrm{cost}(t) = \mathrm{cost}(\mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(a,b), \mathrm{c}^{\mathrm{o}}(d,e))) = +(+(5,7), \min(6, +\infty)) = +(12,6) = 18$.

Example 5.3 demonstrates how to calculate the proponent's minimal cost to achieve his goal in a scenario. Since the opponent's cost has no influence on the proponent's cost, the values associated with the opponent's basic actions express the cost from the proponent's point of view rather than the actual cost for the opponent. To reflect this fact, every basic action of the opponent is assigned $+\infty$, which is the absorbing element for the operation $\mathrm{c}^{\mathrm{p}}_\alpha = +$ and the neutral element for the operation $\mathrm{c}^{\mathrm{o}}_\alpha = \min$.

5.2 *Semantics preserving attribute domains*

In our framework, we consider equivalent ADTrees to be indistinguishable. Thus, the evaluation of attributes on equivalent ADTerms should be consistent, i.e. should yield the same values. However, as shown in the example below, this is not always the case.

EXAMPLE 5.4
The ADTerm $t = \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(a,b), \mathrm{c}^{\mathrm{o}}(d,e))$, considered in Example 5.3, is equivalent to the ADTerm $t' = \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(a,a,b), \mathrm{c}^{\mathrm{o}}(d,e))$, if the propositional semantics is used. However, the evaluation of the proponent's minimal cost in t', gives $\mathrm{cost}(t') = +(+(5,5,7), \min(6,+\infty)) = 23 \neq 18 = \mathrm{cost}(t)$.

The problem of consistent bottom-up evaluation of attribute values has already been discussed in the case of attack trees (see [15, 21]). The authors of [21] identify a sufficient condition guaranteeing that, when the multiset semantics is used, the bottom-up evaluation of attributes on attack trees is performed in a consistent way. In the current article, we generalize this result to any semantics for ADTerms, by introducing the notion of compatibility of an attribute domain with a semantics for ADTerms. Compatibility constitutes a sufficient condition for consistent bottom-up evaluation of attributes on equivalent ADTrees.

Consider an attribute domain $A_\alpha = (D_\alpha, \vee_\alpha^{\mathrm{p}}, \wedge_\alpha^{\mathrm{p}}, \vee_\alpha^{\mathrm{o}}, \wedge_\alpha^{\mathrm{o}}, \mathrm{c}_\alpha^{\mathrm{p}}, \mathrm{c}_\alpha^{\mathrm{o}})$ and the set $\mathbb{T}_\Sigma^{\mathrm{VAR}}$ of typed ADTerms over the variables in VAR, as introduced in Section 3.5. Given an ADTerm $t \in \mathbb{T}_\Sigma^{\mathrm{VAR}}$, we denote by t_α an expression built from the elements of $\mathbb{B} \cup \mathrm{VAR}$ and operators $\vee_\alpha^s, \wedge_\alpha^s, \mathrm{c}_\alpha^s$, for $s \in \{\mathrm{p}, \mathrm{o}\}$, recursively defined as follows. Let $t^1, \dots, t^k \in \mathbb{T}_\Sigma^{\mathrm{VAR}}$ and $k > 0$. Then

$$\begin{aligned} &t_\alpha = t, \text{ if } t \in \mathbb{B} \cup \mathrm{VAR}, && (\vee^s(t^1, \dots, t^k))_\alpha = \vee_\alpha^s(t_\alpha^1, \dots, t_\alpha^k), \\ &(\mathrm{c}^s(t^1, t^2))_\alpha = \mathrm{c}_\alpha^s(t_\alpha^1, t_\alpha^2), && (\wedge^s(t^1, \dots, t^k))_\alpha = \wedge_\alpha^s(t_\alpha^1, \dots, t_\alpha^k). \end{aligned} \tag{5}$$

Note that in the expressions t_α the elements of $\mathbb{B} \cup \mathrm{VAR}$ are variables ranging over D_α.

Definition 5.5

An attribute domain $A_\alpha = (D_\alpha, \vee_\alpha^{\mathrm{p}}, \wedge_\alpha^{\mathrm{p}}, \vee_\alpha^{\mathrm{o}}, \wedge_\alpha^{\mathrm{o}}, \mathrm{c}_\alpha^{\mathrm{p}}, \mathrm{c}_\alpha^{\mathrm{o}})$ is compatible with a semantics $\equiv$ for ADTerms if and only if, for all $t, t' \in \mathbb{T}_\Sigma$, the semantical equivalence $t \equiv t'$ implies that the equality $t_\alpha = t'_\alpha$ holds in D_α.

Example 5.6

Consider two terms $t = \mathrm{c}^{\mathrm{p}}(b, \wedge^{\mathrm{o}}(d, e))$ and $t' = \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(b, b) \wedge^{\mathrm{o}} (d, e))$. In Example 3.5, we have shown that $t \equiv_{\mathcal{P}} t'$. By using the attribute domain $A_{\mathrm{sat}} = (\{0, 1\}, \vee, \wedge, \vee, \wedge, \star, \star)$, where $x \star y = x \wedge \neg y$, for all $x, y \in \{0, 1\}$, introduced in Example 5.2, and the procedure described by (5), we define the expressions t_{sat} and t'_{sat} as follows

$$t_{\mathrm{sat}} = b \wedge \neg(d \wedge e) \quad \text{and} \quad t'_{\mathrm{sat}} = (b \wedge b) \wedge \neg(d \wedge e).$$

Due to the idempotency of $\wedge$, we obtain that the equality $t_{\mathrm{sat}} = t'_{\mathrm{sat}}$ holds in $D_{\mathrm{sat}} = \{0, 1\}$.

From Definitions 4.3 and 5.5 we can easily deduce that if an attribute domain is compatible with a semantics for ADTerms it is also compatible with every semantics which is finer.

In most cases, due to the infinite number of equivalent ADTerms, employing Definition 5.5 is impractical. The next proposition overcomes this obstacle. It follows directly from (5) and Definitions 5.5 and 4.1.

Proposition 5.7

Let E be a complete set of axioms for a semantics $\equiv$ for ADTerms. An attribute domain $A_\alpha = (D_\alpha, \vee_\alpha^{\mathrm{p}}, \wedge_\alpha^{\mathrm{p}}, \vee_\alpha^{\mathrm{o}}, \wedge_\alpha^{\mathrm{o}}, \mathrm{c}_\alpha^{\mathrm{p}}, \mathrm{c}_\alpha^{\mathrm{o}})$ is compatible with the semantics $\equiv$ if and only if, for every equation $t = t'$ in E, the equality $t_\alpha = t'_\alpha$ holds in D_α.

Proposition 5.7 shows that a complete set of axioms is a powerful tool to ensure the practical usability of semantics for ADTerms. By making use of a complete set of axioms, Proposition 5.7 gives us a simple and efficient procedure for checking compatibility of a given attribute domain with a considered semantics.

Example 5.8

Using Proposition 5.7, we can easily prove that the attribute domain A_{cost}, used in Examples 5.3 and 5.4 to compute the proponent's minimal cost, is not compatible with the propositional semantics. Indeed, according to Theorem 4.6, the axiom

$$\wedge^{\mathrm{p}}(X, X, X_1, \dots, X_k) = \wedge^{\mathrm{p}}(X, X_1, \dots, X_k)$$

holds for the propositional semantics, but in $\mathbb{R}\cup\{+\infty\}$ we have

$$(\wedge^{\mathrm{p}}(X,X,X_1,\dots,X_k))_{\mathrm{cost}} = +(X,X,X_1,\dots,X_k) \neq \\ +(X,X_1,\dots,X_k) = (\wedge^{\mathrm{p}}(X,X_1,\dots,X_k))_{\mathrm{cost}},$$

because $+$ is not idempotent. This explains why the evaluation of the proponent's minimal cost on two equivalent ADTerms in the propositional semantics, presented in Example 5.4, gives two different results.

We now prove that semantically equivalent ADTerms always yield equal attribute values over compatible attribute domains.

LEMMA 5.9
Consider an attribute domain $A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha, \vee^{\mathrm{o}}_\alpha, \wedge^{\mathrm{o}}_\alpha, \mathrm{c}^{\mathrm{p}}_\alpha, \mathrm{c}^{\mathrm{o}}_\alpha)$, a basic assignment $\beta_\alpha : \mathbb{B} \to D_\alpha$ and two ADTerms t and t'. 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 $\sigma : \mathbb{B}\cup\mathrm{VAR} \to D_\alpha$. Thus, it suffices to show that for every ADTerm t, we have

$$\beta_\alpha(t_\alpha) = \alpha(t). \tag{6}$$

The proof of (6) 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 (6) holds for all ADTerms composing t, and let $t = \vee^{\mathrm{p}}(t^1,\dots,t^k)$. We have

$$\beta_\alpha(t_\alpha) = \beta_\alpha(\vee^{\mathrm{p}}_\alpha(t^1_\alpha,\dots,t^k_\alpha)) = \vee^{\mathrm{p}}_\alpha(\beta_\alpha(t^1_\alpha),\dots,\beta_\alpha(t^k_\alpha)) \\ = \vee^{\mathrm{p}}_\alpha(\alpha(t^1),\dots,\alpha(t^k)) = \alpha(t).$$

The proof for the remaining composed ADTerms is similar.

Using (6), we obtain that, if $t_\alpha = t'_\alpha$ holds in D_α, then $\alpha(t) = \beta_\alpha(t_\alpha) = \beta_\alpha(t'_\alpha) = \alpha(t')$, which finishes the proof. ■

From (4), Definition 5.5 and Lemma 5.9, we obtain the following result.

THEOREM 5.10
Let $A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha, \vee^{\mathrm{o}}_\alpha, \wedge^{\mathrm{o}}_\alpha, \mathrm{c}^{\mathrm{p}}_\alpha, \mathrm{c}^{\mathrm{o}}_\alpha)$ be an attribute domain compatible with a semantics $\equiv$ for ADTerms. If $t \equiv t'$, then, given any basic assignment $\beta_\alpha : \mathbb{B} \to D_\alpha$, we have $\alpha(t) = \alpha(t')$.

5.3 Attribute domains compatible with the multiset semantics

Although performing various case studies using the ADTree methodology, we have noticed that most of the useful attribute domains for ADTrees admit a structure $A_\alpha = (D_\alpha, \boxplus, \boxtimes, \boxtimes, \boxplus, \boxtimes, \boxplus)$, where $(D_\alpha, \boxplus, \boxtimes)$ forms an idempotent semi-ring[5], i.e. the operations $\boxplus$ and $\boxtimes$ are both commutative and associative, $\boxtimes$ distributes over $\boxplus$, operator $\boxplus$ is idempotent and D_α contains a neutral element w.r.t. $\boxplus$ which is absorbing w.r.t. $\boxtimes$. This is, for instance, the case for the attribute domain $A_\alpha = (\mathbb{R}^+\cup\{+\infty\}, \min, +, +, \min, +, \min)$ which is used to calculate the minimal cost or the minimal time necessary for the proponent to achieve his main goal. This also holds

[5] Due to the commutativity of $\boxtimes$, the structure $(D_\alpha, \boxplus, \boxtimes)$ is usually called a commutative semi-ring.

for $A_\alpha = (\mathbb{R}_{\geq 0} \cup \{-\infty, +\infty\}, \min, \max, \max, \min, \ \max, \min)$ which can be used to calculate the minimal skill level necessary for the proponent to perform an attack. A third example is $A_\alpha = (\{0, \ldots, k\}, \min, \star, \star, \min, \star, \min)$, where $\star(a,b) = \min\{a+b, k\}$, which helps us to model which goals of the proponent are executable in $< k$ units of time. Theorem 5.11 shows that the evaluation of this type of attributes is consistent with the multiset semantics. It therefore implies that the multiset semantics is one of the most important and useful semantics for ADTerms.

THEOREM 5.11
Every attribute domain of the form $A_\alpha = (D_\alpha, \boxplus, \boxtimes, \boxtimes, \boxplus, \boxtimes, \boxplus)$, where $(D_\alpha, \boxplus, \boxtimes)$ forms an idempotent semi-ring, is compatible with the multiset semantics for ADTerms.

PROOF. Let us consider the complete set of axioms $E_{\mathcal{M}}$ for the multiset semantics, given in Theorem 4.9. According to Proposition 5.7, it is sufficient to show that for every $l = r \in E_{\mathcal{M}}$, the equality $l_\alpha = r_\alpha$ holds in D_α. The equalities corresponding to axioms (E^{s}_{1}), (E^{s}_{2}) result from the commutativity of $\boxplus$ and $\boxtimes$. The equalities corresponding to (E^{s}_{3}), (E^{s}_{4}), (E^{p}_{17}) and (E^{p}_{20}) hold due to associativity of both operations. Distributivity of $\boxtimes$ over $\boxplus$ guarantees that the equalities corresponding to (E^{o}_{9}), (E^{p}_{10}), (E^{p}_{13}), (E^{p}_{16}), (E^{p}_{18}) and (E^{o}_{19}) are satisfied in D_α. Finally, the equalities corresponding to (E^{p}_{11}) and (E^{o}_{12}) result from the idempotency of $\boxplus$. Note that axioms (E^{s}_{5}), (E^{s}_{6}) are purely syntactical and are needed for technical reasons, only. ■

The authors of [21] show that, in the case of attack trees, every attribute domain which is a semi-ring is compatible with the multiset semantics. Theorem 5.11 extends this result from attack trees to ADTrees. Note that the result proven in [21] only holds for idempotent semi-rings. Indeed, equations $(E^{\mathrm{p}}_{1}), (E^{\mathrm{p}}_{2}), (E^{\mathrm{p}}_{3}), (E^{\mathrm{p}}_{4}), (E^{\mathrm{p}}_{5}), (E^{\mathrm{p}}_{6}), (E^{\mathrm{p}}_{10})$ and (E^{p}_{11}) axiomatize the multiset semantics for attack trees. Thus, if the attribute domain forms a semi-ring which is not idempotent (as for instance in the case of the algebraic semi-ring, i.e. $(\mathbb{R}, +, \times)$), the computation of attribute values on two equivalent attack trees, such as $t = \vee^{\mathrm{p}}(a, a)$ and $t' = a$, does not yield the same result.

6 Related work

The idea of using AND–OR trees for security assessment takes its origins from the field of safety analysis. As early as in the 60s, Vesely *et al.* [32] proposed *fault trees* to evaluate safety of critical infrastructures and analyse associated risks. In early 90s, inspired by fault trees, Weiss [33] and Amoroso [3] developed *threat trees* to model vulnerabilities that complex systems, such as hospital information systems, are subject to. The notion of *attack trees* is due to Schneier who introduced them in 1999 as a visual and systematic methodology for security assessment [29]. In 2005, Mauw and Oostdijk [21] augmented attack trees with semantics, providing a solid, formal and methodological framework for security analysis. Since then, the attack tree methodology has been taken up by numerous researchers. An excellent summary about the history of formal graphical security models, including attack trees, is given by Piètre-Cambacédès and Bouissou [23]. Attack trees constitute a very popular method for security modelling in both both in industrial and academic environment. They have been adopted as a support tool in a number of international research projects, for instance SHIELDS [31], EVITA [13] and ANIKETOS [4]).

Several authors have proposed to augment attack trees with a notion of defense, in the past [2, 5, 8, 23, 27, 36]. Different approaches, ranging from adding defenses to the leaf nodes of the attack tree, over extending attack trees with various types of defensive measures, such as mitigation, response or detection nodes, to also considering a separate tree that describes possible protection scenarios and relates to the root of the attack tree, have been considered.

Edge *et al.* [12] have shown how to compute the cost or the probability of an attack from an attacker's as well as from a defender's point of view. Modelling the defender's point of view was made possible by creating a protection component, for every leaf of an attack tree, and then constructing protection trees by using these components as leaf nodes.

It is also possible to unite the attacker's and the defender's points of view and create a single framework, instead of keeping two separated models for the attacker and the defender. Bistarelli *et al.* [7, 8] have proposed so-called defense trees, where defensive measures are added to the leaves of attack trees. Furthermore, they use methods from game theory and answer set programming, to deduce which defensive measures should be selected.

Roy *et al.* [27] have introduced attack countermeasure trees, where countermeasures, such as detection and mitigation, are allowed at any level of the tree. They have studied consequences of adding countermeasures in a border gateway protocol attack, an attack on a supervisory control and data acquisition system and a malicious insider attack. The practical feasibility of their approach was illustrated by computing the impact and the cost of a successful attack as well as the system's risk to a particular attack scenario.

Piètre-Cambacédès and Bouissou [23] have used Boolean logic Driven Markov Processes (BDMPs) to assign a new semantics to attack trees. The general idea of BDMPs is to associate a Markov process to each leaf of an attack tree. Since BDMPs are dynamic, their use allows for the modelling of attack sequences. Moreover, BDMPs can also model dynamic defensive aspects, such as detections or mitigations.

Another extension of attack trees is described by Baca and Petersen [5]. Instead of focusing on the identification of attacks, they propose to prioritize and evaluate countermeasures, which are again only assigned to the leaves of the attack trees. Since they allow countermeasures to counter several attacks, their formalism is based on DAGs. Cost and effectiveness of countermeasures are evaluated and then depicted in a two-dimensional graph. The approach has been applied on an open source system, called Code 43. The case study showed that the described method identifies the most effective and cost-efficient countermeasures.

Extending attack trees with defenses or countermeasures is not the only way of enriching the attack tree formalism. In [35], Yager introduces ordered weighted averaging trees (OWA trees) by allowing attack trees with additional refinement of operators. By ordering the children of nodes it is possible to model how many children of a node must be satisfied, to satisfy the parent node. The introduced OWA nodes even allow probabilistic uncertainty of the number of children that need to be satisfied, so that the parent node is satisfied.

In [34], Willemson and Jürgenson also consider ordered attack trees. Their novelty is to introduce an order on the set of leaves of an attack tree, which allows them to select the best attack option represented by the tree. Moreover, the authors generalize their framework from tree structures to DAGs.

Abdulla *et al.* [2] take generalizing the tree structure even further by defining attack jungles, allowing for multiple roots, cycles and nodes representing reusable assets. The authors have implemented a prototype tool and used it to evaluate the security in the GSM radio network using attack jungles.

The ADTree methodology extends attack trees as formalized in [21], in two ways. It introduces defenses and it generalizes the notions of semantics and attributes. Consequently, our formalism provides a single framework covering concepts developed in [12, 22, 26, 28, 34].

The ADTrees formalism has proven to be useful in theoretical considerations. In [16], the relation between ADTrees and game theory has been studied. This work shows that ADTrees interpreted with the propositional semantics and binary zero-sum two-player extensive form games can be converted into each other. Both formalisms have their advantages. On the one hand, ADTrees provide easily understandable and intuitive representation of attack–defense scenarios. On the other hand, the game theoretic approach benefits from the well-studied methodology used in games such as solution concepts, which can be used to find optimal strategies for involved players.

Computational aspects of ADTrees have been studied in [18]. The paper describes semantics for which ADTrees extend attack trees to a richer formalism without increasing the computational complexity of the model. This is the case for the propositional semantics and more generally for every semantics induced by a De Morgan lattice. In other words, the authors of [18] show that when ADTrees are interpreted with De Morgan valuations, the analysis of ADTrees does not require more computational power than the analysis of regular attack trees, as ADTrees can be processed by algorithms developed for attack trees. This, in particular, implies that all queries which can be efficiently solved on attack trees can also be efficiently solved on ADTrees.

Finally, the applicability of the ADTree methodology for quantitative analysis of vulnerability scenarios has been tested in a case study described in [6]. In this work, a denial of service attack for an RFID-based goods management system has been analysed. An extensive ADTree modeling a considered DoS attack have been created and a number of useful attributes have been evaluated. The case study resulted in a definition of precise guidelines specifying how to use ADTrees in practice.

7 Conclusion and future work

We have introduced ADTrees as a new formal approach for security assessment. The ADTrees provide an intuitive and visually appealing representation of interactions between an attacker and a defender of a system. Furthermore, due to the countermeasure operators which connect the opponent's actions to the proponent's actions, ADTrees can be used to represent 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 have introduced several semantics for ADTerms, demonstrating their versatility. Our semantics are defined through equivalence relations on the set of ADTerms. This unifies different approaches [12, 21, 34] to attack trees that have been proposed in the literature, because they all rely upon an underlying equivalence relation.

We have introduced attributes for ADTerms and an evaluation algorithm for ADTerms allowing us to analyse attack–defense scenarios modelled with ADTrees. This extends the approach proposed for attack trees in [21]. Moreover, we have formulated and proved sufficient conditions under which the evaluation of an attribute on equivalent ADTerms results in the same value.

To be able to demonstrate the applicability of ADTrees on real-world examples, we are currently developing a computer tool. The tool will facilitate the construction of large ADTrees, support their graphical representation, and assist in the quantitative analysis of attack–defense scenarios. Furthermore, the complete axiomatization of semantics, as introduced in this article, constitutes a first step towards automated equivalence checking for ADTrees.

In the future, we plan to extend our framework from ADTrees to attack–defense DAGs. Using DAGs we can model dependencies between the sub-goals. This issue is crucial when taking the execution order of sub-goals into account or when analysing an attack–defense scenario from a probabilistic point of view.

Acknowledgments

We would like to thank Marc Pouly and the anonymous reviewers for their invaluable and detailed comments which helped us to improve this article. Barbara Kordy and Patrick Schweitzer were supported by grants from the National Research Fund, Luxembourg, with No. C08/IS/26 and PHD-09-167, respectively.

References

[1] RWTH Aachen. Automated Program Verification Environment (AProVE). `http://aprove.informatik.rwth-aachen.de/`.

[2] P.A. Abdulla, J. Cederberg and L. Kaati. Analyzing the security in the GSM radio network using attack jungles. In *ISoLA (1)*, T. Margaria and B. Steffen, eds, vol. 6415 of *LNCS*, pp. 60–74. Springer, 2010.

[3] E.G. Amoroso. *Fundamentals of Computer Security Technology*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[4] ANIKETOS project. `http://www.aniketos.eu/`.

[5] D. Baca and K. Petersen. Prioritizing countermeasures through the countermeasure method for software security (CM-Sec). In *PROFES*, M. Babar, M. Vierimaa and M. Oivo, eds, vol. 6156 of *LNIBP*, pp. 176–190. Springer, 2010.

[6] A. Bagnato, B. Kordy, P. H. Meland and P. Schweitzer. Attribute decoration of attack–defense trees. *International Journal of Secure Software Engineering (IJSSE)*, **3**, 1–35, 2012.

[7] S. Bistarelli, M. Dall'Aglio and P. Peretti. Strategic games on defense trees. In *FAST*, T. Dimitrakos, F. Martinelli, P. Y. A. Ryan and S. A. Schneider, eds, vol. 4691 of *LNCS*, pp. 1–15. Springer, 2006.

[8] S. Bistarelli, P. Peretti and I. Trubitsyna. Analyzing security scenarios using defence trees and answer set programming. *ENTCS*, **197**, 121–129, 2008.

[9] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[10] Y. Crama and P. Hammer. *Boolean Functions: Theory, Algorithms and Applications*. Cambridge University Press, 2011.

[11] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[12] K. S. Edge, G. C. Dalton II, R. A. Raines and R. F. Mills. Using attack and protection trees to analyze threats and defenses to homeland security. In IEEE, *MILCOM*, pp. 1–7, 2006.

[13] EVITA project. `http://www.evita-project.org/`.

[14] R. Freese, J. Ježek and J. B. Nation. Term rewrite systems for lattice theory. *Journal of Symbolic Computation*, **16**, 279–288, 1993.

[15] A. Jürgenson and J. Willemson. Computing exact outcomes of multi-parameter attack trees. In *OTM Conferences (2)*, R. Meersman and Z. Tari, eds, vol. 5332 of *LNCS*, pp. 1036–1051. Springer, 2008.

[16] B. Kordy, S. Mauw, M. Melissen and P. Schweitzer. Attack–defense trees and two-player binary zero-sum extensive form games are equivalent. In *GameSec*, T. Alpcan, L. Buttyán, and J. S. Baras, eds, vol. 6442 of *LNCS*, pp. 245–256. Springer, 2010.

[17] B. Kordy, S. Mauw, S. Radomirovic and P. Schweitzer. Foundations of attack–defense trees. In *FAST*, P. Degano, S. Etalle, and J. D. Guttman, eds, vol. 6561 of *LNCS*, pp. 80–95. Springer, 2010.

[18] B. Kordy, M. Pouly and P. Schweitzer. Computational aspects of attack–defense trees. In *Security & Intelligent Information Systems*, vol. 7053 of *LNCS*, pp. 103–116. Springer, 2011.

[19] M. Korp, C. Sternagel, H. Zankl and A. Middeldorp. Tyrolean Termination Tool 2. `http://cl-informatik.uibk.ac.at/software/ttt2/index.php`.

[20] C. Marché, A. Rubio and H. Zantema. Termination Problem Data Base: format of input files. `http://www.lri.fr/~marche/tpdb/format.html`.

[21] S. Mauw and M. Oostdijk. Foundations of attack trees. In D. Won and S. Kim, eds, *ICISC*, vol. 3935 of *LNCS*, pp. 186–198. Springer, 2005.

[22] A. N. P. Morais, E. Martins, A. R. Cavalli and W. Jimenez. Security protocol testing using attack trees. In *CSE (2)*, pp. 690–697. IEEE Computer Society, 2009.

[23] L. Piètre-Cambacédès and M. Bouissou. Beyond attack trees: dynamic security modeling with boolean logic driven markov processes (BDMP). In *European Dependable Computing Conference*, pp. 199–208, Los Alamitos, CA, USA, IEEE Computer Society, 2010.

[24] D. A. Plaisted. *Equational Reasoning and Term Rewriting Systems*, pp. 274–364. Oxford University Press, Inc., New York, NY, USA, 1993.

[25] M. Pouly. *A Generic Framework for Local Computation*. PhD Thesis, Department of Informatics, University of Fribourg, 2008.

[26] M. Rehák, E. Staab, V. Fusenig, M. Pěchouček, M. Grill, J. Stiborek, K. Bartoš and T. Engel. Runtime monitoring and dynamic reconfiguration for intrusion detection systems. In *RAID*, E. Kirda, S. Jha, and D. Balzarotti, eds, vol. 5758 of *LNCS*, pp. 61–80. Springer, 2009.

[27] A. Roy, D. S. Kim and K. S. Trivedi. Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. *Journal of Security and Communication Networks, SI: Insider Threats*, 2011. `http://dx.doi.org/10.1002/sec.299`.

[28] V. Saini, Q. Duan and V. Paruchuri. Threat modeling using attack trees. *Journal of Computing Small Colleges*, **23**, 124–131, 2008.

[29] B. Schneier. Attack trees. *Dr. Dobb's Journal of Software Tools*, **24**, 21–29, 1999.

[30] M. Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, **92**, 305–316, 1924.

[31] SHIELDS project. `http://www.shields-project.eu/`.

[32] W. E. Vesely, F. F. Goldberg, N.H. Roberts and D.F. Haasl. Fault tree handbook. *Technical Report NUREG-0492*, U.S. Regulatory Commission, 1981.

[33] J. D. Weiss. A system security engineering process. In *14th National Computer Security Conference*, pp. 572–581, 1991.

[34] J. Willemson and A. Jürgenson. Serial model for attack tree computations. In *ICISC*, D. Lee and S. Hong, eds, vol. 5984 of *LNCS*, pp. 118–128. Springer, 2010.

[35] R. R. Yager. OWA trees and their role in security modeling using attack trees. *Information Sciences*, **176**, 2933–2959, 2006.

[36] S. A. Zonouz, H. Khurana, W. H. Sanders and T. M. Yardley. RRE: a game-theoretic intrusion response and recovery engine. In *IEEE/IFIP International Conference on Dependable Systems Networks, 2009. DSN '09.* pp. 439–448. 2009.

Received 23 May 2012