# Exploiting attack–defense trees to find an optimal set of countermeasures

Barbara Fila
Univ Rennes, INSA Rennes,
CNRS, IRISA, Rennes, France
barbara.fila@irisa.fr

Wojciech Wideł
Univ Rennes, INSA Rennes,
CNRS, IRISA, Rennes, France
&
Division of Network and Systems Engineering,
KTH Royal Institute of Technology,
Stockholm, Sweden
widel@kth.se

***Abstract*—Selecting the most pertinent countermeasures to secure a system is one of the ultimate goals of risk assessment. In this context, it is important to rely on modeling methods that the security experts are already familiar with, so that the solution can be smoothly adopted within industry.**

**We propose a full-fledged framework, relying on attack–defense trees and integer linear programming, to find an optimal set of countermeasures. We use attack–defense trees formalized with directed acyclic graphs. This enables us to conveniently reason about attacker's actions that can contribute to several distinct attacks, and countermeasures that can block different ways of attacking. We provide a constructive way of extracting all reasonable behaviors of the two actors from such models. We then exploit this extracted information to formulate a generic solution, based on integer linear programing, to address a wide class of optimization problems. We show how to instantiate it for specific security-relevant optimization criteria. We cover deterministic and probabilistic cases. The framework has been implemented in a prototype tool, and validated in a real-life case study.**

## I. Introduction

By estimating the cost of an optimal set of countermeasures, the security expert can provide to the system owner an impartial argument about the minimal budget that should be devoted for securing the system. However, defining what an optimal way of protecting a system means might differ depending on the perspective. A security expert might be mostly interested in identifying countermeasures that cover the largest possible part of the attack surface. The system owner, in turn, could rather take an economic point of view, and aim at spending on security only as much as it is really necessary. While considering an attacker having limited resources, it could be wise, from the defender's perspective, to select the countermeasures in such a way that the remaining uncountered attacks are as expensive for the attacker as possible.

Before any of the aforementioned problems can be addressed, one first needs to describe the scenarios of interest in a rigorous way, to be able to reason about possible attacks and countermeasures against them. This can be achieved with the help of *attack–defense trees* (ADTrees). ADTrees extend a well-known and industrially recognized model of *attack trees* with countermeasures. An ADTree involves two actors: an *attacker* who aims at attacking a system, and a *defender* who tries to protect it. ADTrees thus provide a natural way of modeling an interplay between an attacker and a defender.

*Contribution.* The ultimate objective of this work is to devise a sound, mature, and automated solution for finding an optimal set of countermeasures in security scenarios modeled with ADTrees. We address two major scientific challenges underlying this problem.

1) We design a constructive way for extracting all strategies that allow a reasonable attacker to achieve the objective described by an ADTree, and all reasonable strategies of the defender allowing to counter the potential attacks.
2) We then exploit this information on attack and defense strategies to develop a generic optimization framework expressed in terms of integer linear programming (ILP). Instantiating this framework with suitable optimization functions allows for selecting an optimal set of countermeasures according to various practical criteria.

A first attempt to address the above challenges has been proposed in [1], where only ADTrees without repeated nodes were considered. In real-life situations, however, the same action of the attacker can contribute to several attacks. For instance, obtaining admin privileges on a machine grants access to the confidential data stored on the machine but also allows for further penetration of the network that the machine is connected to via horizontal privilege escalation. Similarly, a countermeasure is rarely installed to protect against one specific attack. For instance, proper network segmentation will limit the range of computer worms propagating over resource sharing protocols, but it will also make accessing sensitive data more difficult for an attacker who manages to compromise Internet-facing assets of the network. Generally speaking, an attacker's action having multiple consequences will appear in several attacks, and a security measure bringing numerous benefits will appear in many defense strategies. Thus, there will be multiple nodes corresponding to such an action in an ADTree. This richer model is however much more challenging to analyze, as demonstrated in the Ph.D. thesis [2].


DOI 10.1109/CSF49147.2020.00035

In order to ease the reasoning about scenarios where a single action may have several consequences, we introduce a formalization of ADTrees based on *directed acyclic graphs* (DAGs), rather than on classically used trees or terms. We then design an algorithm to extract from an ADTree its *defense semantics* describing how the two actors may interact. The defense semantics assumes that the attacker and the defender are reasonable, i.e., they do not execute actions that are useless. Although the idea behind the defense semantics stays the same as in [1], its construction in the case of ADTrees with repeated nodes presents new non-trivial, computational challenges.

To address the second challenge, we transform the defense semantics of an ADTree into an ILP problem. By instantiating it with specific optimization functions, we can express numerous optimization problems, including coverage and investment tackled by [1] in a simplified setting without repeated nodes. Our framework is suitable for solving problems involving a single parameter, e.g., cost of the attack, as well as for addressing those relying on simultaneous analysis of multiple parameters. We cover a deterministic case, where the countermeasures are fully successful, and a probabilistic case, where countermeasures may fail with some probability.

Finally, to make our results applicable in practice, we implement a tool that automates the computation of the defense semantics and solves some of the discussed optimization problems. The implementation has been tested on a number of ADTrees having various size and structure. Our framework and tool have been validated on a real-life case study.

*Paper structure.* We present the ADTree model and introduce its formalization based on DAGs in Section II. The definition and construction of the defense semantics for ADTrees with repeated nodes is presented in Section III. Section IV is devoted to the specification of various optimization problems in terms of ILP. In Section V, we briefly describe the tool that we implemented and discuss its performance. We also give a pointer to an empirical validation of our framework. We discuss the related work in Section VI, and conclude in Section VII.

## II. Attack–defense trees

To make this article self-contained, we start by recalling background information about ADTrees. In Section II-A, we present a toy scenario that we use to illustrate the ADTree model, and we introduce a formalization of ADTrees based on DAGs. Then, in Section II-B, we adapt the standard bottom-up procedure for analysis of ADTrees to our DAG-based formalization, and we discuss some results on satisfiability of ADTrees, that will be helpful to reason about reasonable behavior of the attacker and the defender.

### A. Attack–defense tree formalism

Classically, ADTrees [3] are rooted, labeled trees representing an interplay between an *attacker* and a *defender*. The nodes' labels depict the goals of the two actors. The root goal corresponds to the main objective of the modeled scenario. The actor trying to achieve this goal is called *proponent*. It can be either the attacker or the defender. The other actor, called *opponent*, wants to prevent the proponent from succeeding. The nodes representing complex goals are *refined* in a disjunctive (`OR`) or a conjunctive (`AND`) way. The goal of an `OR` node is achieved when a goal of at least one of its refining children is achieved. The goal of an `AND` node is achieved when the corresponding actor achieves the goals of all of its refining children. Labels of the *non-refined nodes* represent atomic steps performed by the actors in order to achieve more complex goals. Finally, each goal of an actor can be *countered* by a goal of the other one.

As visible, e.g., in [4], [5], ADTrees (and more generally, all attack tree-based models) used in practice often contain several nodes bearing the same label. In order to devise correct analysis procedures, the meaning of such nodes needs to be clear. In this work, nodes having the same label represent exactly the same instance of a goal. Following the approach introduced in [6], we call such nodes *clones*.

From the efficiency perspective, it is wise to use a DAG representation, where clones are merged into a single node. Using DAGs instead of trees reduces storage requirements and might be exploited for speeding up computations. It also improves readability of the models as it corresponds better to our interpretation of nodes with the same label.

Example 1 describes a toy scenario of getting confidential data. It illustrates the ADTree syntax and provides a setting on which the notions we define can be illustrated.

**Example 1.** *The attacker's goal is to get the password protecting a computer. To do so, they could get physical access to the computer, extract the password's hash from it, and crack it. They could also find a post-it note with the password. A phishing attack could also be used. The first two approaches require the attacker to find the victim's office and enter it.*

*The attacker will not be able to enter the office if an access control measure is in place. If the user (defender) sets up a strong password, cracking it will become impossible. Finally, a well-educated user will not write their password on a post-it note, and will not be vulnerable to a phishing attack.*

*Each of the actions* find office*,* enter office *and* security training *is a clone.*

The scenario from Example 1 is modeled in Figure 1 with an ADTree depicted as a DAG. We use standard graphical notation for ADTrees: the nodes of the attacker are represented with red ellipses, and those of the defender with green rectangles; the refining children of an `AND` node are connected with an arc, and the countermeasures are attached to the countered nodes using dotted edges.

Before introducing a formal definition of ADTrees based on DAGs, we provide the necessary terminology.

A *directed graph* is an ordered pair $(V, E)$ consisting of a set $V$ of *nodes* and a set $E \subseteq V \times V$ of *directed edges*. Given an edge $(v, w) \in E$, we say that $v$ is a *child* of $w$ and that $w$ is a *parent* of $v$. A *path* in a directed graph $(V, E)$ is a sequence of nodes from $V$, in which each node is a child of its successor. If none of the paths in a directed graph contains some node

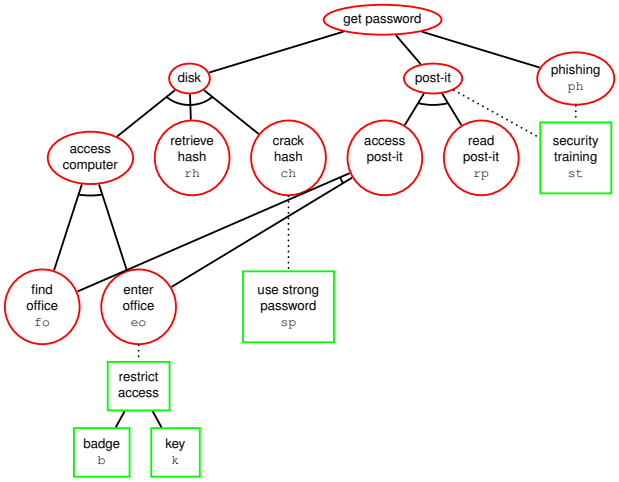


Figure 1. ADTree for getting data, in a form of a DAG

more than once, then the graph is *acyclic*. A *directed acyclic graph* (DAG) is *rooted* if it has exactly one node with no parents. This node is called *root*. Given two DAGs $(V, E)$ and $(V', E')$, where $V' \subseteq V$ and $E'$ consists of all the edges of $E$ whose both endpoints belong to $V'$, we say that $(V', E')$ is a *subdag of* $(V, E)$ *induced by* $V'$.

Our definition of ADTrees based on DAGs follows.

**Definition 1** (ADTree)**.** *An attack–defense tree is a tuple* $T = (V, E, \mathrm{L}, \lambda, \mathrm{actor}, \tau)$*, where*

- $(V, E)$ *is a rooted DAG,*
- $\mathrm{L}$ *is a set of labels representing the attacker's and the defender's goals,*
- $\lambda\colon V \to \mathrm{L}$ *is an injective function assigning labels to the nodes,*
- $\mathrm{actor}\colon V \to \{\mathtt{a}, \mathtt{d}\}$ *is a function assigning actors to the nodes, in such a way that every node has at most one child assigned to the other actor,*
- $\tau\colon V \to \{\mathtt{OR}, \mathtt{AND}, \mathtt{N}\}$ *describes the refinement type of a node. We use* OR *for disjunctively and* AND *for conjunctively refined nodes,* N *stands for the non-refined nodes, i.e., nodes labeled with basic actions,*
- *for every node* $v \in V$*,* $\tau(v) = \mathtt{N}$ *if and only if* $v$ *has no child assigned to the same actor as* $v$*.*

Let $T = (V, E, \mathrm{L}, \lambda, \mathrm{actor}, \tau)$ be an ADTree. The *root* of $T$, denoted $\mathrm{root}(T)$, is the root of its underlying DAG. We set

$$\mathrm{ref}_T(v) = \{w \in V\colon (w, v) \in E,\ \mathrm{actor}(w) = \mathrm{actor}(v)\},$$

to denote the (possibly empty) set of all nodes refining $v \in V$, i.e., its children belonging to the same actor as $v$. The (possibly non-existing) child of $v$ belonging to the other actor is denoted $\bar{v}$.

The actor assigned to the root of an ADTree is the proponent, and the other one is the opponent. Given a tree $T$, its proponent is denoted with $\mathrm{p}_T$, i.e., $\mathrm{p}_T = \mathrm{actor}(\mathrm{root}(T))$, and its opponent with $\mathrm{o}_T$. For an actor $\mathrm{s} \in \{\mathtt{a}, \mathtt{d}\}$, we use $\bar{\mathrm{s}}$ to denote the other actor, i.e., $\bar{\mathtt{a}} = \mathtt{d}$, $\bar{\mathtt{d}} = \mathtt{a}$.

The labels of the non-refined nodes are *basic actions*. For $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$, we denote by $\mathbb{B}^{\mathrm{s}_T}$ the set of basic actions of the corresponding actor in $T$, and we set $\mathbb{B}_T := \mathbb{B}^{\mathrm{p}_T} \cup \mathbb{B}^{\mathrm{o}_T}$. Note that, since the labeling function $\lambda$ is injective, the sets $\mathbb{B}^{\mathrm{p}_T}$ and $\mathbb{B}^{\mathrm{o}_T}$ are disjoint, for every ADTree $T$. The universe of all basic actions is denoted by $\mathbb{B}$.

Finally, for $v \in V$, we use $T(v)$ to denote *the maximal subdag of* $T$ *rooted at* $v$ i.e., the subdag induced by all nodes $v'$ such that there exists a path from $v'$ to $\mathrm{root}(T)$ containing $v$.

### B. Bottom-up analysis of attributes on attack–defense trees

*Attributes* (also called *parameters* or *metrics*) allow to reason about quantitative aspects of security scenarios [7], [8], [9]. In this work, we are mostly interested in the *cost* and the *satisfiability* attributes. The first one is used to identify cost-efficient behavior of the two actors, while the second one is helpful in finding potential ways of attacking or protecting the analyzed system. Bottom-up evaluation of attributes is a standard method for analyzing ADTrees. The notions employed in this section are well-established [10], [3], [8], but we adapt them to our new formalization of ADTrees based on DAGs.

**Definition 2** (Attribute domain)**.** *Let* $\alpha$ *be an attribute. The attribute domain for* $\alpha$ *is a tuple* $A_\alpha = (D_\alpha, \mathtt{OR}^{\mathrm{p}}_\alpha, \mathtt{AND}^{\mathrm{p}}_\alpha, \mathtt{OR}^{\mathrm{o}}_\alpha, \mathtt{AND}^{\mathrm{o}}_\alpha, \mathtt{C}^{\mathrm{p}}_\alpha, \mathtt{C}^{\mathrm{o}}_\alpha)$*, where for* $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$

- $D_\alpha$ *is a set of values that the attribute can attain,*
- $\mathtt{OR}^{\mathrm{s}}_\alpha$ *and* $\mathtt{AND}^{\mathrm{s}}_\alpha$ *are unranked operations*[1] *on* $D_\alpha$*,*
- $\mathtt{C}^{\mathrm{s}}_\alpha$ *are binary operations on* $D_\alpha$*.*

Attribute domain $A_\alpha$ can be used to analyze an ADTree w.r.t. attribute $\alpha$ in a bottom-up way. To do so, one first assigns values to the basic actions using a function $\beta\colon \mathbb{B} \to D_\alpha$, called *basic assignment*, and then combines them using the attribute domain's operations according to the ADTree structure.

**Definition 3** (Bottom-up evaluation of attributes)**.** *Let* $\alpha$ *be an attribute with the domain* $A_\alpha = (D_\alpha, \mathtt{OR}^{\mathrm{p}}_\alpha, \mathtt{AND}^{\mathrm{p}}_\alpha, \mathtt{OR}^{\mathrm{o}}_\alpha, \mathtt{AND}^{\mathrm{o}}_\alpha, \mathtt{C}^{\mathrm{p}}_\alpha, \mathtt{C}^{\mathrm{o}}_\alpha)$*, and let* $\beta\colon \mathbb{B} \to D_\alpha$ *be a basic assignment. Given an ADTree* $T = (V, E, \mathrm{L}, \lambda, \mathrm{actor}, \tau)$ *and a node* $v \in V$ *such that* $\mathrm{actor}(v) = \mathrm{s}_T$*,* $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$*, and* $\tau(v) = \mathtt{OP}$*, the* value of $\alpha$ at $v$ under $\beta$*, denoted by* $\alpha(T, \beta, v)$*, is defined recursively as follows*

$$\alpha(T, \beta, v) := \begin{cases} \beta(\lambda(v)), & \text{if } \mathtt{OP} = \mathtt{N} \text{ and } \bar{v} \text{ does not exist,} \\ \mathtt{C}^{\mathrm{s}}_\alpha\left(\beta(\lambda(v)), \alpha(T, \beta, \bar{v})\right), & \text{if } \mathtt{OP} = \mathtt{N} \text{ and } \bar{v} \text{ exists,} \\ (\mathtt{OP}^{\mathrm{s}}_\alpha)_{v' \in \mathrm{ref}_T(v)} \alpha(T, \beta, v'), & \text{if } \mathtt{OP} \neq \mathtt{N} \text{ and } \bar{v} \text{ does not exist,} \\ \mathtt{C}^{\mathrm{s}}_\alpha((\mathtt{OP}^{\mathrm{s}}_\alpha)_{v' \in \mathrm{ref}_T(v)} \alpha(T, \beta, v'), \alpha(T, \beta, \bar{v})), & \text{otherwise.} \end{cases}$$

*The* value of attribute $\alpha$ for $T$ under $\beta$ obtained via the bottom–up procedure*, denoted by* $\alpha(T, \beta)$*, is then defined as* $\alpha(T, \beta, \mathrm{root}(T))$.

[1]Formally, an unranked operation $f$ on a set $D$ is a family of $k$-ary operations $\{f_k\colon D^k \to D\}_{k>0}$.

Of special interest for our work is the *satisfiability* attribute, denoted $\texttt{sat}$, and modeled with the domain $A_{\texttt{sat}} = (\{0,1\}, \vee, \wedge, \vee, \wedge, \star, \star)$, where $x \star y = x \wedge \neg y$. In Section III-B, we also define some nonstandard attributes that will be helpful in extracting the behavior of the two actors from an ADTree.

We now discuss some interesting properties of the *satisfiability* attribute that are relevant for our framework.

First, remark that the intuitive notions of refinement and goal achievement can be formalized using the attribute domain $A_{\texttt{sat}}$. Throughout the paper, for an ADTree $T$, one of its nodes $v$, and a set $B \subseteq \mathbb{B}_T$ of basic actions, we use $\text{achieved}_{\text{T}}(v, B)$ as a shorthand for $\texttt{sat}(T, \mathbb{1}_B, v)$, where $\mathbb{1}_B$ is the indicator function of $B$. We say that *the goal of $v$ is achieved by $B$ in $T$* if $\text{achieved}_{\text{T}}(v, B) = 1$; otherwise, *the goal of $v$ is not achieved by $B$ in $T$*. The following example illustrates our formalization of a goal achievement in ADTrees.

**Example 2.** *Let $T$ be the ADTree from Figure 2. Assuming that the attacker executes only the actions* $\texttt{a}$ *and* $\texttt{c}$ *and the defender executes only* $\texttt{d}_1, \texttt{d}_2$, *and* $\texttt{d}_3$, *which is modeled by assigning* 1 *to each of these actions and* 0 *to the remaining basic actions, the defender fails to counter the action* $\texttt{a}$ *(the value computed at the* AND *node countering* $\texttt{a}$ *is* 0*), and so the attacker achieves the goal of the root node (the value computed at the root node is* 1*). In other words, for the set $X = \{\texttt{a}, \texttt{c}, \texttt{d}_1, \texttt{d}_2, \texttt{d}_3\}$ the equality $\text{achieved}_{\text{T}}(\text{root}(T), X) = 1$ holds.*

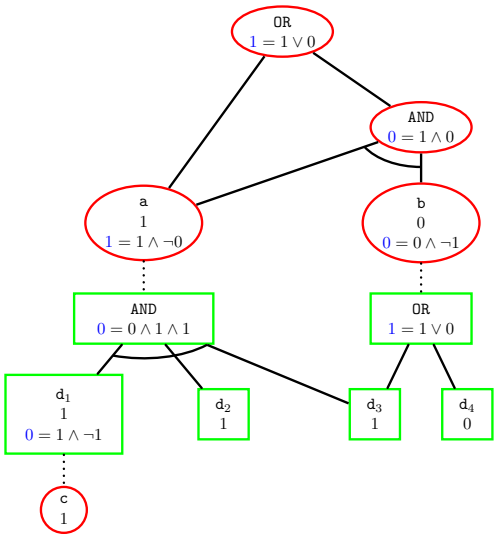


Figure 2. Bottom-up evaluation of the *satisfiability* attribute. Values assigned to the basic actions are given in black, and values computed at the intermediate nodes using the attribute domain's operations in dark blue.

For a node $v$ of a tree $T$ and a set $B \subseteq \mathbb{B}_T$, the value of $\text{achieved}_{\text{T}}(v, B)$ is obtained by evaluating a specific Boolean function (called *propositional semantics* in [11]). This function has been proven to be positive[2] in the variables corresponding to the basic actions of $\text{actor}(v)$, and negative in the remaining variables [11]. This fact has multiple direct consequences, some of them intuitively obvious, of which the following two corollaries will be of use for us.

[2]Boolean function $f$ is positive in variable $x$, iff $f(x = 0) \leq f(x = 1)$; otherwise, $f$ is negative in $x$.

**Corollary 1.** *Let $T = (V, E, \text{L}, \lambda, \text{actor}, \tau)$ be an ADTree and let $B \subseteq \mathbb{B}^{\text{s}_T}$, with $\text{s} \in \{\text{p}, \text{o}\}$, be a set of actions of one of the actors. If the equality $\text{achieved}_{\text{T}}(v, B) = 0$ holds for a node $v \in V$, then $\text{achieved}_{\text{T}}(v, B \setminus \{\texttt{b}\}) = 0$, for every $\texttt{b} \in \mathbb{B}_T$.*

*Proof.* We may assume that $\texttt{b} \in B$, since otherwise the statement is obviously true.

Suppose first that $\text{actor}(v) = \text{s}_T$. The value of $\text{achieved}_{\text{T}}(v, B)$ is computed by replacing the 0 assigned to the variable corresponding to the basic action $\texttt{b}$ in $\text{achieved}_{\text{T}}(v, B \setminus \{\texttt{b}\})$ into 1. Since $\texttt{b} \in B$ and $B \subseteq \mathbb{B}^{\text{s}_T}$, the function $\text{achieved}_{\text{T}}(v, \cdot)$ is positive in that variable. Together with the equality $\text{achieved}_{\text{T}}(v, B) = 0$, this implies that $\text{achieved}_{\text{T}}(v, B \setminus \{\texttt{b}\}) = 0$.

Suppose now that $\text{actor}(v) = \bar{\text{s}}_T$. Observe that Definition 1 and 3 together with the definition of the *satisfiability* attribute domain imply that $\text{achieved}_{\text{T}}(v, \emptyset) = 0$. Together with the fact, the function $\text{achieved}_{\text{T}}(v, \cdot)$ is negative in the variables corresponding to the actions belonging to the set $B$, this implies that $\text{achieved}_{\text{T}}(v, B \setminus \{\texttt{b}\}) = 0$. □

**Corollary 2.** *Let $T = (V, E, \text{L}, \lambda, \text{actor}, \tau)$ be an ADTree, $B \subseteq \mathbb{B}_T$ be a set of basic actions of the actors, and $v \in V$ be a node with $\text{actor}(v) = \text{s}_T$, for $\text{s} \in \{\text{p}, \text{o}\}$. Then,*

- *if $\text{achieved}_{\text{T}}(v, B) = 0$, then $\text{achieved}_{\text{T}}(v, B \cup B') = 0$, for every $B' \subseteq \mathbb{B}^{\bar{\text{s}}_T}$, and*
- *if $\text{achieved}_{\text{T}}(v, B) = 1$, then $\text{achieved}_{\text{T}}(v, B \cup B') = 1$, for every $B' \subseteq \mathbb{B}^{\text{s}_T}$.*

We are especially interested in the contraposition of the first statement from Corollary 2, for $\text{s}_T = \text{p}_T$.

**Corollary 3.** *Let $T = (V, E, \text{L}, \lambda, \text{actor}, \tau)$ be an ADTree, let $v \in V$ be a node satisfying $\text{actor}(v) = \text{p}_T$ and let $P \subseteq \mathbb{B}^{\text{p}_T}$ be a set of basic actions of the proponent. If there exists a set $O \subseteq \mathbb{B}^{\text{o}_T}$ of basic actions of the opponent such that $\text{achieved}_{\text{T}}(v, P \cup O) = 1$, then $\text{achieved}_{\text{T}}(v, P) = 1$.*

Corollary 2 implies also the following.

**Corollary 4.** *Let $T = (V, E, \text{L}, \lambda, \text{actor}, \tau)$ be an ADTree, and let $P \subseteq \mathbb{B}^{\text{p}_T}$ and $O \subseteq \mathbb{B}^{\text{o}_T}$ be sets of basic actions of the actors. If the equalities $\text{achieved}_{\text{T}}(v, P) = 0$ and $\text{achieved}_{\text{T}}(v, O) = 0$ hold for a node $v \in V$, then $\text{achieved}_{\text{T}}(v, P \cup O) = 0$.*

We rely on Corollary 4 to prove the intuitively obvious statement: if a root goal of an ADTree is achieved by a set of basic actions and an action from this set does not contribute to the goal being achieved, then the goal is still achieved after the action is removed from the set. This statement is formalized in the following lemma.

**Lemma 1.** *Let $T = (V, E, \mathrm{L}, \lambda, \mathrm{actor}, \tau)$ be an ADTree, and let $P \subseteq \mathbb{B}^{\mathrm{p}_T}$ and $O \subseteq \mathbb{B}^{\mathrm{o}_T}$ be sets of basic actions of the actors. If $v \in V$ is a node such that*

- $\tau(v) = \mathtt{N}$,
- $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) = 1$, *and*
- *on every path from $v$ to $\mathrm{root}(T)$, there exists a node $v'$ s.t. $\mathrm{achieved}_{\mathrm{T}}(v', P) = 0$ and $\mathrm{achieved}_{\mathrm{T}}(v', O) = 0$,*

*then* $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O \setminus \{\lambda(v)\}) = 1$.

*Proof.* Let $v'$ be one of the nodes satisfying the last condition of the lemma. Corollary 4 implies that $\mathrm{achieved}_{\mathrm{T}}(v', P \cup O) = 0$. Therefore, when the value of $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O)$ is computed using the bottom-up procedure, the value propagated up to the root from $v'$ is zero. Furthermore, it follows from Corollary 1 that $\mathrm{achieved}_{\mathrm{T}}(v', O \setminus \{\lambda(v)\}) = 0$ and $\mathrm{achieved}_{\mathrm{T}}(v', P \setminus \{\lambda(v)\}) = 0$. Thus, by Corollary 4, $\mathrm{achieved}_{\mathrm{T}}(v', P \cup O \setminus \{\lambda(v)\}) = 0$, i.e., the value propagated from $v'$ remains unchanged after the removal of the basic action $\lambda(v)$ from $P \cup O$. Hence,

$$\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O \setminus \{\lambda(v)\}) = \mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) = 1.$$

□

The above results will be useful in investigating relations between sets of actions of the two actors, and, eventually, in determining optimal sets of countermeasures in security scenarios modeled with ADTrees.

## III. Defense semantics for attack–defense trees

The ultimate goal of this work is to extract possible behaviors of rational actors from an ADTree modeling a security scenario, and to exploit this information for optimal selection of countermeasures to be implemented by the opponent. We express actors' behavior in terms of sets of their basic actions, that we call *strategies*. While some works, e.g., [12], [13], consider any subset of basic actions of an actor to be a possible strategy, such an approach is not only computationally ineffective, but also unnecessary, in the sense that among all the subsets there are inefficient ones that do not correspond to a reasonable behavior.

In this section, we present a semantics for ADTrees called *defense semantics*. Defense semantics of an ADTree $T$ pairs each of the rational ways for the proponent to achieve the root goal of $T$ together with the minimal sets of the opponent's actions preventing the proponent from succeeding. It has been introduced in [1], for trees containing no clones. We adapt the notions introduced in [1] to our DAG-based formalization of ADTrees in Section III-A. A novel algorithm for the construction of the defense semantics, suitable for trees containing clones, is given in Section III-B. The complexity of the defense semantics computation is discussed in Section III-C.

### A. Defense semantics definition

When looking at a graphically depicted ADTree, one can easily distinguish its structural components, namely, the maximal rooted subdags where all nodes belong to the same actor. We call them *homogeneous subdags*. They play a crucial role in defining the defense semantics.

**Definition 4** (Homogeneous subdag)**.** *Let $T = (V, E, \mathrm{L}, \lambda, \mathrm{actor}, \tau)$ be an ADTree and let $H = (V_H, E_H)$ be a rooted DAG such that*

- $V_H \subseteq V$, $E_H = E \cap (V_H \times V_H)$,
- *either at least one of the parents of $\mathrm{root}(H)$ in $T$ belongs to the actor other than $\mathrm{actor}(\mathrm{root}(H))$, or $\mathrm{root}(H) = \mathrm{root}(T)$,*
- $\mathrm{ref}_T(v) \subseteq V_H$, *for every $v \in V_H$,*
- $\bar{v} \notin V_H$, *for every $v \in V_H$.*

*Moreover, let $\lambda_H$, $\mathrm{actor}_H$, and $\tau_H$ be restrictions of $\lambda$, actor, and $\tau$ to $V_H$. If the actor assigned to all the nodes of $V_H$ is $\mathrm{p}_T$ (resp. $\mathrm{o}_T$), then the ADTree $(V_H, E_H, \mathrm{L}, \lambda_H, \mathrm{actor}_H, \tau_H)$ is called a homogeneous subdag of the proponent (resp. of the opponent) in $T$. If the only homogeneous subdag of $T$ is $T$ itself, then $T$ is called a homogeneous ADTree.*

ADTree from Figure 1 has four homogenous subdags. They are illustrated in Figure 3.

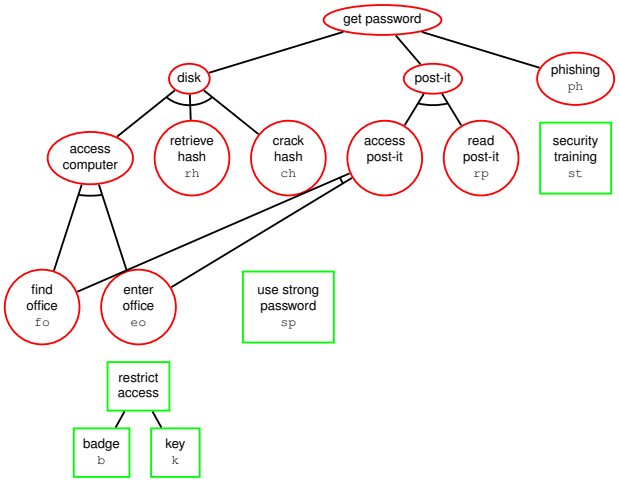


Figure 3. Homogenous subdags of the ADTree from Figure 1

Note that the second condition of Definition 4 implies that the goal of the root node of a homogeneous subdag either counters some goal of the other actor, or achieving it means success for the proponent. Therefore, in order to succeed, the actors need to achieve the root goals of (some of) their corresponding homogeneous subdags. If a set of actions achieves none of the root goals of the homogeneous subdags, its execution has no impact on the realization of the modeled scenario. Therefore, we use minimal sets of actions that achieve root goals of homogeneous subdags as the building blocks for our formalization of the behavior of rational actors. We call them *proponent's* and *opponent's vectors*.

**Definition 5** (Proponent's/opponent's vector)**.** *Let $T$ be an ADTree and let $H$ be a homogenous subdag of the proponent (resp. opponent) in $T$. A minimal, w.r.t. the inclusion, set*

*of basic actions of the proponent (resp. opponent) achieving the root goal of $H$ is called a* proponent's vector *(resp. an* opponent's vector*)* in $T$.

For example, there are three attacker's vectors in the homogeneous subtree from Figure 3 rooted in the `get password` node: {`fo`, `eo`, `rh`, `ch`}, {`fo`, `eo`, `rp`} and {`ph`}.

In order to counter the proponent in the best way possible, the opponent might be interested in executing a number of opponent's vectors from some homogeneous subdags of an ADTree. Given such a specific behavior of the opponent, we assume that a rational proponent executes only those actions that are *necessary* for achieving the root goal. Such behavior of rational actors is captured with the notion of their strategy.

**Definition 6** (Proponent's/opponent's strategy)**.** *Let $T$ be an ADTree.*

- *A set $O \subseteq \mathbb{B}^{\mathrm{o}_T}$ is called an opponent's strategy in $T$ if it is a union of some of the opponent's vectors in $T$. Remark that the empty set is a possible opponent's strategy.*
- *A set $P \subseteq \mathbb{B}^{\mathrm{p}_T}$ is called a proponent's strategy in $T$ if there exists an opponent's strategy $O$ in $T$ for which $P$ is a minimal set satisfying* $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) = 1$. *Such a set $O$ is called a* witness *for the proponent's strategy $P$.*

Note that every proponent's strategy can be witnessed by many opponent's strategies, and each of the opponent's strategies can be a witness for a number of proponent's strategies.

Let $P$ and $O$ be sets of basic actions of the proponent and the opponent in $T$, respectively. We say that $O$ *counters* $P$, if $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) = 0$; otherwise $P$ counters $O$. With the actors' strategies defined by Definition 6, our objective of determining possible behavior of a rational proponent and ways of countering it is accomplished with the notion of *defense semantics*.

**Definition 7** (Defense semantics)**.** *The defense semantics of an ADTree $T$, denoted $\mathcal{D}(T)$, is the set of all pairs $(P, O)$, where $P$ is a proponent's strategy in $T$ and $O$ is a minimal (w.r.t. the inclusion) opponent's strategy in $T$ that counters $P$.*

We would like to stress that the proponent's strategies in an ADTree that cannot be countered *do not* appear in its defense semantics. The proponent's strategies in $T$ that *do* appear in the defense semantics of $T$, i.e., those that can be countered by an opponent's strategy in $T$, are called *counterable*.

**Example 3.** *The defense semantics of the ADTree from Figure 1 is composed of the following five pairs of strategies:*

$(\{fo, eo, rh, ch\}, \{b\})$, $(\{fo, eo, rh, ch\}, \{k\})$,
$(\{fo, eo, rh, ch\}, \{sp\})$, $(\{fo, eo, rp\}, \{st\})$,
$(\{ph\}, \{st\})$.

While the concept of the defense semantics is intuitively simple, constructing this semantics is a complex task, especially in the case of ADTrees with repeated nodes. We discuss it in the next section.

### *B. Defense semantics construction*

To construct the defense semantics of an ADTree $T$, one could consider the following naive approach, which we are going to build upon.

1) Create all the opponent's strategies in $T$.
2) For every opponent's strategy, determine the proponent's strategies witnessed by it.
3) For every proponent's strategy, identify the minimal opponent's strategies countering it.

The first of the three steps is already very expensive, since, in the worst case, every subset of basic actions of the opponent might constitute an opponent's strategy. Luckily, we can show (see Proposition 1 and 2) that this step's complexity can be reduced by creating (if possible) only a subset of all possible opponent's strategies, while ensuring that every proponent's strategy is witnessed by at least one element of this subset. Once this subset has been obtained, one can proceed with the remaining two steps. The construction of the defense semantics is summarized in Algorithm 1. The rest of this section is devoted to explaining it and to proving its correctness and completeness.

We start by introducing four operations on sets of sets that we use to define attribute domains employed by Algorithm 1. For $n$ sets $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of sets, let

$$\bigotimes_{i=1}^{n} \mathcal{A}_i := \{\bigcup_{i=1}^{n} A_i \mid A_i \in \mathcal{A}_i\}, \tag{1}$$

$$\bigoplus_{i=1}^{n} \mathcal{A}_i := \bigcup_{I \subseteq \{1,\ldots,n\}} \bigotimes_{i \in I} \mathcal{A}_i, \tag{2}$$

$$\mathcal{A}_1 \oslash \mathcal{A}_2 := \begin{cases} \{\emptyset\}, \text{ if } \mathcal{A}_1 = \{\emptyset\} \text{ or } \mathcal{A}_2 = \{\emptyset\}, \\ \mathcal{A}_1 \cup \mathcal{A}_2, \text{ otherwise,} \end{cases} \tag{3}$$

$$\mathcal{A}_1 \odot \mathcal{A}_2 := \mathcal{A}_1 \cup (\mathcal{A}_1 \otimes \mathcal{A}_2). \tag{4}$$

To construct a set of witnesses sufficient for determining all proponent's strategies, we employ the bottom-up evaluation of the *sufficient witnesses* attribute, abbreviated as `SuffWit`, formalized with the attribute domain $A_{\texttt{SuffWit}} := (2^{2^{\mathbb{B}}}, \oplus, \oplus, \oplus, \otimes, \oplus, \odot)$. We begin, in Proposition 1, with specifying a basic assignment under which each of the elements belonging to the result of this bottom-up evaluation is an opponent's strategy in the tree.

**Proposition 1.** *Let $T = (V, E, \mathrm{L}, \lambda, \mathrm{actor}, \tau)$ be an ADTree and let $\beta_T^{\texttt{SW}}$ be the basic assignment for the* `SuffWit` *attribute, defined as*

$$\beta_T^{\texttt{SW}}(\mathtt{b}) := \begin{cases} \emptyset, \text{ if } \mathtt{b} \in \mathbb{B}^{\mathrm{p}_T}, \\ \{\{\mathtt{b}\}\}, \text{ otherwise.} \end{cases} \tag{5}$$

*If $O \in \texttt{SuffWit}(T, \beta_T^{\texttt{SW}})$, then $O$ is an opponent's strategy in $T$.*

*Proof.* We shall prove that, for every $v \in V$, every element $O \in \texttt{SuffWit}(T, \beta_T^{\texttt{SW}}, v)$ is a union of opponent's vectors from some of the homogeneous subdags of $T(v)$. The validity of this statement for $v = \mathrm{root}(T)$ completes the proof of Proposition 1.

**Algorithm 1** Defense semantics for ADTrees

**Input:** ADTree $T$
**Output:** Defense semantics $\mathcal{D}(T)$

$\mathcal{O} \leftarrow \mathtt{SuffWit}(T, \beta_T^{\mathtt{SW}}) \cup \{\emptyset\}$
$\mathcal{P} \leftarrow \emptyset$
**for** $O \in \mathcal{O}$ **do**
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{P\colon P \text{ is a minimal set in } \mathtt{CounterOpp}(T, \beta_T^O)\}$
**end for**
$\mathcal{D}(T) \leftarrow \emptyset$
**for** $P \in \mathcal{P}$ **do**
    $\mathcal{D}(T) \leftarrow \mathcal{D}(T) \cup \{(P, O)\colon O \text{ is a minimal set in } \mathtt{CounterPro}(T, \beta_T^P)\}$
**end for**
**return** $\mathcal{D}(T)$

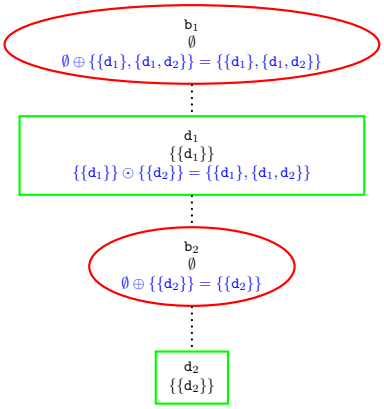


Figure 4. Bottom-up evaluation of the SuffWit attribute. Basic assignment given in black, values computed given in blue.

The proof is by induction on the structure of $T(v)$. If $v$ is a non-refined node such that $\bar{v}$ does not exist, then the statement is obviously true by the definition of the basic assignment $\beta_T^{\mathtt{SW}}$.

If $v$ is refined or $\bar{v}$ exists, then, since every element of $\mathtt{SuffWit}(T, \beta_T^{\mathtt{SW}}, v)$ is a union of some sets belonging to

$$\bigcup_{v' \in \mathrm{ref}_T(v) \cup \{\bar{v}\}} \mathtt{SuffWit}(T, \beta_T^{\mathtt{SW}}, v'),$$

by formulæ (1), (2) and (4), the statement follows from the induction hypothesis. □

Throughout the rest of this work, whenever we say "bottom-up evaluation of the SuffWit attribute", we mean the evaluation under the basic assignment $\beta_T^{\mathtt{SW}}$ given by formula (5).

The proof of Proposition 1 provides some insight into our motivation for the choice of the operations of the SuffWit attribute domain: they are defined in a way that ensures that the result of the bottom-up evaluation of the SuffWit attribute consists of opponent's strategies. Nevertheless, being aware that the definition of the attribute domain $A_{\mathtt{SuffWit}}$ is non-intuitive, we illustrate its usage with Example 4.

**Example 4.** *Figure 4 exemplifies the bottom-up evaluation of the* SuffWit *attribute. It is easy to verify that the opponent's strategy* $\{\mathtt{d}_1\}$ *is the unique minimal witness for the proponent's strategy* $\{\mathtt{b}_1, \mathtt{b}_2\}$*. The set* $\{\mathtt{d}_1, \mathtt{d}_2\}$ *is an opponent's strategy in the tree, but it is not a witness for any of the proponent's strategies.*

*Should a node of the attacker labeled* $\mathtt{b}_3$ *be attached as a countermeasure to the node labeled* $\mathtt{d}_2$*, the set obtained with the bottom-up evaluation of* SuffWit *in the resulting tree would be the same as in the tree from Figure 4. In this case, however, the opponent's strategy* $\{\mathtt{d}_1, \mathtt{d}_2\}$ *would be the unique minimal witness for the proponent's strategy* $\{\mathtt{b}_1, \mathtt{b}_2, \mathtt{b}_3\}$.

There are ADTrees for which the result of the bottom-up evaluation of the SuffWit attribute consists of exactly the non-empty witnesses necessary for determining all proponent's strategies. As illustrated by Example 4, this is the case, for instance, for an ADTree being a path of alternating non-refined nodes of the proponent and the opponent, where the first node on the path belongs to the proponent. We discuss such trees further in the following example.

**Example 5.** *Let* $T$ *be an ADTree being a path of* $2n+1$ *alternating non-refined nodes of the proponent and the opponent, with the first node on the path belonging to the proponent, as schematized in Figure 5. The total number of non-empty opponent's strategies in* $T$ *is* $2^n - 1$*, whereas there are only* $n - 1$ *strategies in the result of the bottom-up evaluation of* SuffWit *on* $T$*. Furthermore, each of them is a unique witness for one of the proponent's strategies: the opponent's strategy* $\{\mathtt{d}_1, \ldots, \mathtt{d}_i\}$*, with* $i \in \{1, \ldots, n\}$*, is the unique witness for the proponent's strategy* $\{\mathtt{b}_1, \ldots, \mathtt{b}_{i+1}\}$.

*Observe the following: if* $O$ *is an opponent's strategy belonging to the result of the bottom-up evaluation of* SuffWit *on* $T$ *and* $\mathtt{d}_i \in O$*, with* $i \in \{1, \ldots, n\}$*, then* $\mathtt{d}_j \in O$ *for every* $j \in \{1, \ldots, i-1\}$*. Informally speaking, there are no "gaps" in the obtained opponent's strategies. This is intentional: should the above condition be not satisfied by an opponent's strategy* $O$*, say,* $O = \{\mathtt{d}_1, \ldots, \mathtt{d}_i, \mathtt{d}_{i+k}\}$*, with* $i, i+k \in \{1, \ldots, n\}$*,* $k > 1$*, then* $O$ *is a witness for the same proponent's strategies as* $\{\mathtt{d}_1, \ldots, \mathtt{d}_i\}$*. This example motivates our choice of the operation* $\odot$ *as the one to be performed in the bottom-up procedure when traversing countermeasures against goals of the opponent.*

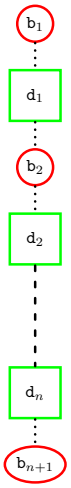


Figure 5. A schema generalizing the tree from Figure 4

Notice that Example 4 illustrates also the fact that, in general, in the set returned by the bottom-up evaluation of the SuffWit attribute there might be opponent's strategies that do not witness any proponent's strategy, or that witness the same proponent's strategies as other elements of the set. In other words, the set might contain more strategies than needed, which is not ideal. This drawback is outweighed, however, by the fact that the set contains at least one witness for each of the proponent's strategies in $T$, as we shall prove in Proposition 2.

Our proof of Proposition 2 relies on the property of the attribute domain $A_{\texttt{SuffWit}}$ stated in Lemma 2, which can be intuitively explained as follows. Suppose that each of the actors fixes a set of their actions to be executed. As long as it is possible, keep removing from these sets actions that do not contribute to any of the root goals of homogeneous subdags being achieved. Then, the remaining actions of the opponent, which are the only actions relevant for the realization of the scenario under the fixed behavior of the actors, belong to some intermediate result of the bottom-up evaluation of the SuffWit attribute.

**Lemma 2.** *Let $T = (V, E, \mathrm{L}, \lambda, \mathrm{actor}, \tau)$ be an ADTree, let $v \in V$ and let $\beta_T^{\mathtt{SW}}$ be the basic assignment of the* SuffWit *attribute defined by* (5)*. Let $T' = (V', E')$ be a rooted subdag of $T$, such that*

- $\mathrm{root}(T') = v$,
- *if* $v' \in V'$ *and* $\tau(v') = \mathtt{AND}$, *then* $\mathrm{ref}_T(v') \subseteq V'$,
- *if* $v' \in V'$ *and* $\tau(v') = \mathtt{OR}$, *then the intersection* $\mathrm{ref}_T(v') \cap V'$ *is not empty,*
- $E' = E \cap (V' \times V')$.

*Let*

$$\mathbb{B}^{\mathrm{o}_T}_{T'} := \{\lambda(v') \colon v' \in V', \mathrm{actor}(v') = \mathrm{o}_T, \tau(v') = \mathtt{N}\} \quad (6)$$

*be the set of all basic actions of the opponent in $T$ that appear in $T'$. If the set $\mathbb{B}^{\mathrm{o}_T}_{T'}$ is non-empty, then it belongs to* $\mathtt{SuffWit}(T, \beta_T^{\mathtt{SW}}, v)$.

Due to space restrictions, a proof of Lemma 2 can be found in the full version of this work [14].

We are now ready to state and prove the following result.

**Proposition 2.** *Let $T = (V, E, \mathrm{L}, \lambda, \mathrm{actor}, \tau)$ be an ADTree, $P$ be a proponent's strategy in $T$, and let $\beta_T^{\mathtt{SW}}$ be the basic assignment defined by* (5)*. If $O$ is a minimal non-empty witness for $P$ in $T$, then $O \in \mathtt{SuffWit}(T, \beta_T^{\mathtt{SW}})$.*

*Proof.* We begin with constructing an appropriate subdag of $T$ to which we then apply Lemma 2. Let

$$V_1 := \{v \in V \colon \mathrm{achieved}_{\mathrm{T}}(v, P) = 1 \text{ or } \mathrm{achieved}_{\mathrm{T}}(v, O) = 1\},$$
$$V_2 := \{v \in V_1 \colon \exists v_1, v_2, \ldots, v_m \in V, \text{ s.t. } v_1 v_2 \ldots v_m \text{ is a path in } T, v_1 = v, v_m = \mathrm{root}(T), \text{ and } v_i \in V_1, \text{ for } i \in \{1, \ldots, m\}\}.$$

Let $T_2$ be the subdag of $T$ induced by $V_2$. Observe that, since $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) = 1$, it follows from Corollary 3 that the root of $T$ belongs to the set $V_1$. Together with the definition of $V_2$, this implies that the subdag $T_2$ is connected and rooted at $\mathrm{root}(T)$[3]. Furthermore, the choice of $V_1$ and $V_2$ implies that $T_2$ satisfies the assumptions of Lemma 2 (as the subdag $T'$). Therefore, if the set $\mathbb{B}^{\mathrm{o}_T}_{T_2}$ defined by (6) is not empty, then $\mathbb{B}^{\mathrm{o}_T}_{T_2} \in \mathtt{SuffWit}(T, \beta_T^{\mathtt{SW}})$. To complete the proof it remains to show that $\mathbb{B}^{\mathrm{o}_T}_{T_2} = O$.

The inclusion $\mathbb{B}^{\mathrm{o}_T}_{T_2} \subseteq O$ follows immediately from the choice of $V_2$. To prove that the two sets are in fact equal, suppose that there exists a node $v \in V$ with $\lambda(v) \in O$ which does not belong to $V_2$. Then, since $v \in V_1$, it follows that on every path from $v$ to $\mathrm{root}(T)$ there is a node $v'$ other than $v$, such that $\mathrm{achieved}_{\mathrm{T}}(v', P) = 0$ and $\mathrm{achieved}_{\mathrm{T}}(v', O) = 0$. Since $O$ is a witness for $P$, we have $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) = 1$. Therefore, Lemma 1 implies that $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O \setminus \{\lambda(v)\}) = 1$. This contradicts the choice of $O$ as the minimal witness for $P$. Hence, $\mathbb{B}^{\mathrm{o}_T}_{T_2} = O$, completing the proof. □

Proposition 1 and 2 imply the following result.

**Corollary 5.** *Let $T = (V, E, \mathrm{L}, \lambda, \mathrm{actor}, \tau)$ be an ADTree and let $\beta_T^{\mathtt{SW}}$ be the basic assignment defined by* (5)*. The set*

$$\{P \subseteq \mathbb{B}^{\mathrm{p}_T} \colon \exists O \in \mathtt{SuffWit}(T, \beta_T^{\mathtt{SW}}) \text{ s.t. } P \text{ is a minimal set countering } O \text{ or } P \text{ is a minimal set countering } \emptyset\}$$

*consists of all the proponent's strategies in $T$.*

Corollary 5 shows that, in order to identify witnesses relevant for the computation of the defense semantics, it is sufficient to compute $\mathtt{SuffWit}(T, \beta_T^{\mathtt{SW}})$ instead of all possible

[3]In fact, $T_2$ is the component of the subdag of $T$ induced by the set $V_1$ that contains the root of $T$. Intuitively, $T_2$ models the (relevant part of the) particular realization of the scenario represented by $T$, when the opponent executes all of the actions in $O$, and the proponent — all of the actions in $P$.

opponent's strategies. To evaluate the impact of this observation in practice, we have tested it on some example ADTrees. The results are discussed in Section V.

With a sufficient set of witnesses identified, the next step of the defense semantics' construction is to determine the proponent's strategies. This can be achieved with the help of the attribute `CounterOpp`, formalized with the attribute domain $A_{\texttt{CounterOpp}} := (2^{2^{\mathbb{B}}}, \cup, \otimes, \otimes, \cup, \otimes, \oslash)$.

**Proposition 3.** *Let $T = (V, E, \mathrm{L}, \lambda, \mathrm{actor}, \tau)$ be an ADTree, let $v \in V$ and $O \subseteq \mathbb{B}^{\mathrm{o}_T}$. Let $\beta_T^O$ be the basic assignment for the* `CounterOpp` *attribute defined by*

$$\beta_T^O(\lambda(v)) := \begin{cases} \{\{\lambda(v)\}\}, \text{ if } \mathrm{actor}(v) = \mathrm{p}_T, \\ \emptyset, \text{ if } \mathrm{actor}(v) = \mathrm{o}_T,\ \lambda(v) \in O, \\ \{\emptyset\}, \text{ if } \mathrm{actor}(v) = \mathrm{o}_T,\ \lambda(v) \notin O. \end{cases} \tag{7}$$

*Let $P$ be a set of basic actions of the proponent such that $P \in \texttt{CounterOpp}(T, \beta_T^O, v)$. If $\mathrm{actor}(v) = \mathrm{p}_T$, then $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 1$; otherwise $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 0$.*

*Proof.* The proof is by induction on the structure of $T(v)$ – the maximal subdag of $T$ rooted at $v$.

For the base case, let $v$ be a non-refined node and assume that $\bar{v}$ does not exist. Since the set $\texttt{CounterOpp}(T, \beta_T^O, v)$ is not empty, the definition of the basic assignment $\beta_T^O$ implies that $\mathrm{actor}(v) = \mathrm{p}_T$ and $P = \{\lambda(v)\}$, or $\mathrm{actor}(v) = \mathrm{o}_T$ and $P = \emptyset$. In the former case, the claim follows immediately. In the latter, we have $\lambda(v) \notin O$, implying that

$$\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = \mathrm{achieved}_{\mathrm{T}}(v, O) = 0,$$

as required.

**Case 1.** The node $v$ is not refined and $\bar{v}$ exists.

**Case 1.1.** $\mathrm{actor}(v) = \mathrm{p}_T$.
Under the assumptions of this case, and since the set $\texttt{CounterOpp}(T, \beta_T^O, v)$ is not empty, formula (1), the definition of the `CounterOpp` domain, and the definition of the basic assignment $\beta_T^O$ imply that $\texttt{CounterOpp}(T, \beta_T^O, \bar{v}) \neq \emptyset$ and $P = \overline{P} \cup \{\lambda(v)\}$, for some set $\overline{P} \in \texttt{CounterOpp}(T, \beta_T^O, \bar{v})$. By the induction hypothesis, the equality $\mathrm{achieved}_{\mathrm{T}}(\bar{v}, \overline{P} \cup O) = 0$ holds, and so the definition of the *satisfiability* attribute domain implies that $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 1$.

**Case 1.2.** $\mathrm{actor}(v) = \mathrm{o}_T$.
In the case when $\lambda(v) \in O$, we have $\texttt{CounterOpp}(T, \beta_T^O, v) = \texttt{CounterOpp}(T, \beta_T^O, \bar{v})$, by formula (3) and the definition of $\beta_T^O$. Thus, $P \in \texttt{CounterOpp}(T, \beta_T^O, \bar{v})$, which together with the induction hypothesis implies $\mathrm{achieved}_{\mathrm{T}}(\bar{v}, P \cup O) = 1$. Now the equality $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 0$ follows from the definition of the *satisfiability* attribute domain.

If $\lambda(v) \notin O$, then $\beta_T^O(v) = \{\emptyset\}$, and so $\texttt{CounterOpp}(T, \beta_T^O, v) = \{\emptyset\}$, by formula (3). And indeed, since $\lambda(v) \notin O$, the demanded equality

$$\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = \mathrm{achieved}_{\mathrm{T}}(v, O) = 0$$

follows from the definition of the *satisfiability* attribute domain.

If $v$ is a refined node, we let $\mathrm{ref}_T(v) = \{v_1, \ldots, v_k\}$.

**Case 2.** The node $v$ is refined and $\tau(v) = \texttt{OR}$.

**Case 2.1.** $\mathrm{actor}(v) = \mathrm{p}_T$.
Depending on whether or not $\bar{v}$ exists, either $P = P_i \cup \overline{P}$ (if $\bar{v}$ does exist) or $P = P_i$ (if $\bar{v}$ does not exist), for some $i \in \{1, \ldots, k\}$, $P_i \in \texttt{CounterOpp}(T, \beta_T^O, v_i)$ and $\overline{P} \in \texttt{CounterOpp}(T, \beta_T^O, \bar{v})$. Thus, by the induction hypothesis, we have $\mathrm{achieved}_{\mathrm{T}}(\bar{v}, \overline{P} \cup O) = 0$ and $\mathrm{achieved}_{\mathrm{T}}(v_i, P_i \cup O) = 1$, implying that $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 1$.

**Case 2.2.** $\mathrm{actor}(v) = \mathrm{o}_T$.
Again, depending on the existence of $\bar{v}$, and, if it does exist, on whether or not the set $\texttt{CounterOpp}(T, \beta_T^O, \bar{v})$ is empty, either $P = (P_1 \cup \ldots \cup P_k)$, for some $P_i \in \texttt{CounterOpp}(T, \beta_T^O, v_i)$ or $P = \overline{P}$, for some $\overline{P} \in \texttt{CounterOpp}(T, \beta_T^O, \bar{v})$. In the latter case, we have $\mathrm{achieved}_{\mathrm{T}}(\bar{v}, \overline{P} \cup O) = 1$, by the induction hypothesis, and the equality $\mathrm{achieved}_{\mathrm{T}}(v, \overline{P} \cup O) = 0$ follows from the definition of the *satisfiability* attribute domain. Suppose now that the former of the two cases occurs. The induction hypothesis implies that $\mathrm{achieved}_{\mathrm{T}}(v_i, P_i \cup O) = 0$, for $i \in \{1, \ldots, k\}$. Now, it follows from Corollary 2 that $\mathrm{achieved}_{\mathrm{T}}(v_i, P \cup O) = 0$, for $i \in \{1, \ldots, k\}$. Thus, $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 0$.

**Case 3.** The node $v$ is refined and $\tau(v) = \texttt{AND}$.

**Case 3.1.** $\mathrm{actor}(v) = \mathrm{p}_T$.
Depending on the existence of the countermeasure $\bar{v}$, it either holds that $P = (P_1 \cup \ldots \cup P_k) \cup \overline{P}$ or $P = (P_1 \cup \ldots \cup P_k)$, for some $P_i \in \texttt{CounterOpp}(T, \beta_T^O, v_i)$ and $\overline{P} \in \texttt{CounterOpp}(T, \beta_T^O, \bar{v})$. The induction hypothesis implies that $\mathrm{achieved}_{\mathrm{T}}(v_i, P_i \cup O) = 1$, for $i \in \{1, \ldots, k\}$, and $\mathrm{achieved}_{\mathrm{T}}(\bar{v}, \overline{P} \cup O) = 0$. By applying Corollary 2, we get $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 1$.

**Case 3.2.** $\mathrm{actor}(v) = \mathrm{o}_T$.
If $\bar{v}$ does not exist or $\texttt{CounterOpp}(T, \beta_T^O, \bar{v}) = \emptyset$, then, by formula (3), $P = P_i$, for some $i \in \{1, \ldots, k\}$ and $P_i \in \texttt{CounterOpp}(T, \beta_T^O, v_i)$. Otherwise, it might hold that $P = \overline{P}$, for some $\overline{P} \in \texttt{CounterOpp}(T, \beta_T^O, \bar{v})$. In either case, the demanded equality follows from the induction hypothesis and the definition of the *satisfiability* attribute domain. □

Proposition 3 states, in particular, that every set belonging to $\texttt{CounterOpp}(T, \beta_T^O)$, with $\beta_T^O$ defined by (7), counters the set $O$ of basic actions of the opponent in $T$. With the next proposition we establish another useful fact: that every *minimal* set countering $O$ also belongs to $\texttt{CounterOpp}(T, \beta_T^O)$.

**Proposition 4.** *Let $T = (V, E, \mathrm{L}, \lambda, \mathrm{actor}, \tau)$ be an ADTree, $v \in V$, $O \subseteq \mathbb{B}^{\mathrm{o}_T}$ and let $\beta_T^O$ be the basic assignment defined by* (7)*. If*

- $\mathrm{actor}(v) = \mathrm{p}_T$ *and* $P \subseteq \mathbb{B}^{\mathrm{p}_T}$ *is a minimal set such that* $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 1$*, or*
- $\mathrm{actor}(v) = \mathrm{o}_T$ *and* $P \subseteq \mathbb{B}^{\mathrm{p}_T}$ *is a minimal set such that* $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 0$*,*

*then* $P \in \texttt{CounterOpp}(T, \beta_T^O, v)$.

The proof of Proposition 4 is again by induction on the structure of $T(v)$, and it can be found in the full version of this work [14].

Proposition 3 and 4 yield immediately the following result.

**Corollary 6.** *Let $T$ be an ADTree and $O$ be an opponent's strategy in $T$. With $\beta_T^O$ being the basic assignment defined by* (7)*, the minimal (w.r.t. the inclusion) sets from* $\texttt{CounterOpp}(T, \beta_T^O)$ *are the proponent's strategies in $T$ witnessed by $O$.*

The final ingredient of our algorithm for creation of the defense semantics is a method for determining minimal opponent's strategies countering a given proponent's strategy. Conceptually, this task is the same as the one achieved by the domain for the `CounterOpp` attribute, but it requires, informally speaking, switching of the actors: for an ADTree $T$, let $T'$ be the tree obtained by attaching the root of $T$ as a countermeasure to a new node belonging to $\mathrm{o}_T$. Assume that the new node bears a unique label, say $x$. Then, $\mathrm{p}_{T'} = \mathrm{o}_T$, $\mathrm{o}_{T'} = \mathrm{p}_T$, and for every proponent's strategy $P$ in $T$, there is a set $O'$ of basic actions of the opponent in $T'$ such that $O' = P$. Thus, when creating proponent's strategies countering $O'$ in $T'$, one in fact creates the opponent's strategies countering $P$ in $T$. That is, every opponent's strategy countering $O'$ in $T'$ is of the form $P' \cup \{x\}$, where $P' = O$, for some opponent's strategy $O$ in $T$ countering $P$.

More formally, we define the domain $A_{\texttt{CounterPro}} := (2^{2^{\mathbb{B}}}, \otimes, \cup, \cup, \otimes, \oslash, \otimes)$, where the respective operations of the two actors have been exchanged compared to $A_{\texttt{CounterOpp}}$. For an ADTree $T = (V, E, \mathrm{L}, \lambda, \mathrm{actor}, \tau)$ and a set $P \subseteq \mathbb{B}^{\mathrm{p}_T}$ of basic actions of the proponent, let

$$\beta_T^P(\lambda(v)) := \begin{cases} \{\{\lambda(v)\}\}, \text{ if } \mathrm{actor}(v) = \mathrm{o}_T, \\ \emptyset, \text{ if } \mathrm{actor}(v) = \mathrm{p}_T, \lambda(v) \in P, \\ \{\emptyset\}, \text{ if } \mathrm{actor}(v) = \mathrm{p}_T, \lambda(v) \notin P. \end{cases} \tag{8}$$

The above reasoning implies the following corollary.

**Corollary 7.** *Let $T$ be an ADTree and $P$ be a proponent's strategy in $T$. With $\beta_T^P$ being the basic assignment defined by* (8)*, the minimal (w.r.t. the inclusion) sets from* $\texttt{CounterPro}(T, \beta_T^P)$ *are the minimal opponent's strategies in $T$ countering $P$.*

The considerations of this section, in particular Corollary 5, 6, and 7, imply that Algorithm 1 is indeed suitable for creating the defense semantics of an ADTree.

**Corollary 8.** *On input ADTree $T$, Algorithm 1 outputs its defense semantics $\mathcal{D}(T)$.*

### C. Complexity of the defense semantics computation

We would like to draw the reader's attention to the following facts:

**Remark 1.** *Regarding the complexity of Algorithm 1,*

1) *In the worst case, the number of the opponent's strategies created using the* `SuffWit` *attribute domain is exponential in both the number of basic actions of the opponent and the number of the opponent's nodes in the tree.*
2) *The number of proponent's strategies witnessed by a given opponent's strategy can be exponential in both the number of basic actions of the proponent and the number of the proponent's nodes in the tree.*
3) *The number of minimal opponent's strategies countering a given proponent's strategy can be exponential in both the number of basic actions of the opponent and the number of the opponent's nodes in the tree.*

Examples 6, 7, and 8 illustrate the three observations from Remark 1.

**Example 6.** *Let $T$ be the ADTree depicted on Figure 6. Every set of basic actions of the defender that contains a set of the form $\{b_{1i}, b_{2j}, b_{3k}\}$, with $i, j, k \in \{1, 2\}$, is a non-empty defender's strategy in $T$. There are $3^3$ such defender's strategies in $T$, and each of them belongs to the result of the bottom-up evaluation of the* `SuffWit` *attribute on $T$.*

*A simple proof by induction shows that in the general case, when there are $n$* `OR` *nodes of the defender, there are $3^n$ non-empty defender's strategies.*

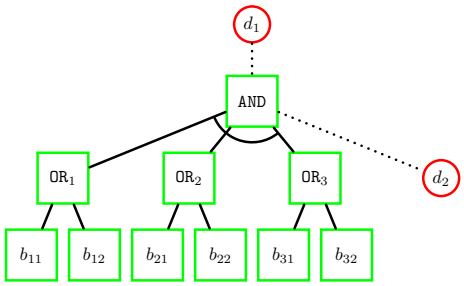


Figure 6. An example of a tree with the number of the opponent's strategies exponential in the number of basic actions

**Example 7.** *Let $T$ be the ADTree depicted on Figure 7. Each of the sets of the form $\{b_1, b_{1i}, b_{2j}, b_{3k}\}$, with $i, j, k \in \{1, 2\}$, is an attacker's strategy witnessed by the defender's strategy $\{d_1\}$. There are $2^3$ such strategies.*

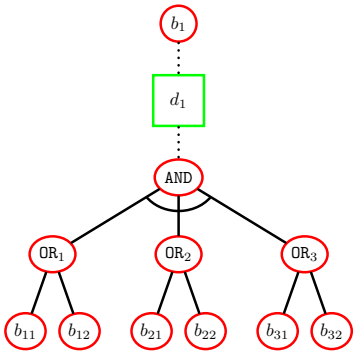


Figure 7. An example of a tree with the number of the proponent's strategies witnessed by a single opponent's strategy exponential in the number of basic actions

**Example 8.** *Let $T$ be the ADTree depicted on Figure 6. Each of the sets of the form $\{b_{1i}, b_{2j}, b_{3k}\}$, with $i, j, k \in \{1, 2\}$, is a minimal defender's strategy countering the attacker's strategy $\{d_1\}$. The number of such sets is $2^3$.*

The three observations from Remark 1 imply that, for a tree $T$ with $n$ nodes, the time needed for execution of each of the lines 1, 4 and 8 of Algorithm 1 is exponential in $n$, implying that our algorithm returns the defense semantics of $T$ in time exponential in $n$. We note that it seems however very difficult, if not impossible, to construct a tree for which each of the three lines would indeed require a number of operations exponential in the number of basic actions in the tree.

To study how Algorithm 1 performs in practice, we have developed a prototype tool implementing the defense semantics computation, and tested it on a number of synthetic ADTrees having various structure. We discuss the tooling and the results of our tests in Section V.

## IV. Optimal selection of countermeasures

We now exploit the defense semantics to design a generic framework for solving optimization problems expressed in terms of ILP. We analyze single parameter and multi-parameter cases, and we deal with proponent and opponent-related parameters. In Section IV-A, we present a mathematical model encoding the defense semantics with a number of propositional variables. Then, we show how to encode the optimization problems of interest using ILP. To ease the framework's understanding, we start in Section IV-B with the deterministic setting, where there is no uncertainty about the outcome of the actions of the opponent, i.e., where every action executed by the opponent succeeds, and contributes fully to all the goals that depend on it. Then, in Section IV-C, we move to a more realistic, probabilistic setting.

### A. The mathematical model

Given an ADTree $T$ and its defense semantics $\mathcal{D}(T)$, let

- $\mathtt{b}_1, \ldots, \mathtt{b}_p$ be the basic actions of the opponent present in $T$,
- $P_1, \ldots, P_n$ be the distinct proponent's strategies that appear in $\mathcal{D}(T)$,
- $O_1, \ldots, O_m$ be the distinct opponent's strategy that appear in $\mathcal{D}(T)$.

Furthermore, for $k \in \{1, \ldots, p\}$, $i \in \{1, \ldots, n\}$, and $j \in \{1, \ldots, m\}$, we set

$$A_{kj} = \begin{cases} 1, \text{ if } \mathtt{b}_k \in O_j, \\ 0, \text{ otherwise,} \end{cases} \qquad B_{ij} = \begin{cases} 1, \text{ if } (P_i, O_j) \in \mathcal{D}(T), \\ 0, \text{ otherwise.} \end{cases}$$

Every basic action $\mathtt{b}$ of the opponent is assumed to be assigned a non-negative integer cost value $\mathtt{cost}(\mathtt{b})$. The total budget available to the opponent is denoted by $\mathcal{B}$. The ways in which execution of particular actions contributes to the implementation of opponent's strategies, which, in turn, results in some proponent's strategies being countered, are modeled with inequalities involving the following Boolean variables:

- $x_k$, for $k \in \{1, \ldots, p\}$: $x_k = 1$ if and only if the opponent executes action $\mathtt{b}_k$,
- $z_i$, for $i \in \{1, \ldots, n\}$: $z_i = 1$ if and only if the proponent's strategy $P_i$ achieves the root node of $T$ in the presence of currently deployed countermeasures,
- $f_j$, for $j \in \{1, \ldots, m\}$: $f_j = 1$ if and only if the opponent does not execute at least one of the basic actions from the opponent's strategy $O_j$.

We are now ready to model the problems of interest with ILP.

### B. Optimization problems in the deterministic case

Let us start with a general form of an optimization problem where the goal of the opponent is to select countermeasures to be implemented, in a way that optimizes a linear function $F$ depending on variables $x_k$, $f_j$, and $z_i$. The total cost of the countermeasures cannot exceed the budget $\mathcal{B}$ available to the opponent. This problem is modeled as an ILP program in Figure 8. Its special case has been considered in [1].

**Optimization goal:**

$$\text{maximize } F(x_1, \ldots, x_p, f_1, \ldots, f_m, z_1, \ldots, z_n) \tag{9}$$

**Subject to:**

$$\sum_{k=1}^{p} \mathtt{cost}(\mathtt{b}_k) x_k \leq \mathcal{B} \tag{10}$$

$$f_j \geq \frac{\sum_{k=1}^{p} A_{kj}(1 - x_k)}{p}, \ 1 \leq j \leq m \tag{11}$$

$$f_j \leq \sum_{k=1}^{p} A_{kj}(1 - x_k), \ 1 \leq j \leq m \tag{12}$$

$$z_i \geq 1 + \sum_{j=1}^{m} B_{ij}(f_j - 1), \ 1 \leq i \leq n \tag{13}$$

$$z_i \leq \frac{\sum_{j=1}^{m} B_{ij} f_j}{\sum_{j=1}^{m} B_{ij}}, \ 1 \leq i \leq n \tag{14}$$

$$\begin{aligned} &x_k \in \{0, 1\}, \quad 1 \leq k \leq p, \\ &f_j \in \{0, 1\}, \quad 1 \leq j \leq m, \\ &z_i \in \{0, 1\}, \quad 1 \leq i \leq n. \end{aligned} \tag{15}$$

Figure 8. General ILP problem for an optimal selection of countermeasures

Constraint (10) ensures that the opponent's investment cannot exceed their budget. The next two families of constraints model the meaning of the variables $f_j$: constraints (11) ensure that if the opponent does not execute some of the actions from $O_j$, then $f_j = 1$; constraints (12) ensure that if $f_j = 1$, then the opponent does not execute some action from $O_j$. Next, we model the meaning of the variables $z_i$: constraints (13) ensure that if the opponent does not execute some action in any of the sets countering $P_i$ (i.e., $f_j = 1$ for every $j$ s.t. $B_{ij} = 1$), then $z_i = 1$; and constraints (14) ensure that if the opponent executes all the actions from at least one of the sets $O_j$ countering the proponent's strategy $P_i$ (i.e., there exists $j$

s.t. $B_{ij} = 1$ and $f_j = 0$), then $z_i = 0$[4]. More details regarding the form of the problem can be found in Remark 1 in [1].

Below, we discuss several specific instances of the general problem from Figure 8 that are relevant for risk analysis.

*1) Coverage problem:* Setting $F := -\sum_{i=1}^{n} z_i$ results in the so called *coverage problem*, where the goal is to cover the largest possible part of the attack surface, i.e., maximize the number of proponent's strategies countered by the opponent.

*2) Countering the most appealing proponent's strategies:* For an ADTree $T$, let $S\colon 2^{\mathbb{B}^{\mathrm{p}_T}} \to \mathbb{Z}^+$ be a *score function* used for comparing proponent's strategies. The higher the value of the score function of a proponent's strategy, the less appealing the strategy is for the proponent. If the opponent cannot fully protect the system, they can at least implement a set of countermeasures that maximizes the minimal value of the score function, among the proponent's strategies successful in the presence of this set. This is achieved by setting $F := C_S$, where $C_S \in \mathbb{Z}^+$ is a new variable, and by introducing constraints

$$C_S \leq z_i S(P_i) + 2(1 - z_i) \max_{P \in \{P_1, \ldots, P_n\}} S(P), \text{ for } i \in \{1, \ldots, n\}. \tag{16}$$

Constraints (16) relate the value of $C_S$ to the values of the score function attained by proponent's strategies not countered by the considered set of countermeasures. They ensure that $C_S$ is always bounded from above by the minimum of these values, i.e., that maximizing $C_S$ is beneficial for the opponent. Should all the proponent's strategies be countered by the opponent under some configuration of variables, then the optimal solution to the optimization problem would be

$$2 \max_{P \in \{P_1, \ldots, P_n\}} S(P).$$

The constant multiplier is a technical trick allowing for distinguishing the case when all the proponent's strategies can be countered (result exceeds the maximal of the scores of the proponent's strategies) from the case when they cannot (the result will correspond to the minimal of the scores among the proponent's strategies that are not countered).

We analyze two particularly useful instances of this problem.

*a) Countering the cheapest proponent's strategies:* Typical example of score function $S$ is the *cost of execution of a strategy*. Assume that the cost of the proponent's actions is modeled with non-negative integers, i.e., that there exists a function $\mathtt{cost}(\cdot)$ defined on $\mathbb{B}$, such that $\mathtt{cost}(\mathtt{b}) \in \mathbb{Z}^+$, for $\mathtt{b} \in \mathbb{B}^{\mathrm{p}_T}$. By solving the problem from Figure 8 extended with constraints (16) for $S(P) := \sum_{\mathtt{b} \in P} \mathtt{cost}(\mathtt{b})$, one obtains a set of countermeasures that maximizes the minimal investment of the proponent necessary to achieve the root goal.

*b) Countering Pareto optimal proponent's strategies:* Let us start with a generic mathematical setting. Suppose that there exists a partial order $\preceq$ defined on the set of proponent's strategies $\mathcal{P} = \{P_1, \ldots, P_n\}$, and that maximal elements w.r.t. this order correspond to the strategies most appealing to the proponent. For a given $P \in \mathcal{P}$, denote by $\#P_{\preceq} \in \mathbb{Z}^+$ the number of elements of a largest totally ordered subset of $\mathcal{P}$, in which $P$ is the minimal element[5]. The smaller the value of $\#P_{\preceq}$, the more appealing the proponent's strategy $P$ is for the proponent, because there are not many strategies that are better than $P$. The opponent's objective is thus to first counter the proponent's strategies $P$ for which the value of $\#P_{\preceq}$ is small. By applying the model from Figure 8 extended with constraints (16) for $S(P) := \#P_{\preceq}$, one identifies a set of countermeasures which maximizes the minimal number $\#P_{\preceq}$ over all proponent's strategies that are not countered, i.e., a set for which the uncountered strategies are as unattractive to the proponent as possible.

The setting described above applies to any partial order on the set of proponent's strategies. In particular, it can be used for countering Pareto optimal proponent's strategies that have been studied in [9]. That is, should each of the proponent's strategies be assigned a vector of values originating from partially ordered sets, one could introduce a partial order $\preceq$ on the set of strategies, were the maximal elements are the strategies that are Pareto optimal w.r.t. all the considered parameters. By instantiating the above generic setting with this order, one selects a set of countermeasures that focuses on countering the proponent's strategies that are Pareto optimal.

*3) Optimizing the opponent's investment without jeopardizing the system:* Assume now that the opponent's budget is not limited, but they do not want to spend on security more than necessary. Suppose that there exists a solution to the coverage problem, in which all counterable proponent's strategies are countered. The opponent can identify a cheapest set of countermeasures countering all counterable proponent's strategies by solving the problem from Figure 8, for $F := -\sum_{k=1}^{p} \mathtt{cost}(\mathtt{b}_k) x_k$, with the constraints (10) and (14) being removed, and with additional $n$ constraints $z_i \leq 0$, for $i \in \{1, \ldots, n\}$ being introduced.

### C. Stochastic model

All the problems considered in Section IV-B assume that the opponent always perform their actions successfully. However, in practice, this happens very rarely. Our framework can be generalized to a non-deterministic case, where the countermeasures may fail. Formally, we associate with every basic action $\mathtt{b}_k \in \mathbb{B}^{\mathrm{o}_T}$ of the opponent a random variable $\xi_k$ that is equal to 1 if $\mathtt{b}_k$ has been implemented successfully, and 0 otherwise (according to the Bernoulli distribution). After splitting the function $F$ into two parts $F = G + H$, with $G = G(x_1, \ldots, x_p)$, $H = H(f_1, \ldots, f_m, z_1, \ldots, z_n)$, the optimization problem from Figure 8 becomes then a stochastic programming problem (see, e.g., [15]) given in Figure 9, where $\xi := (\xi_1, \ldots, \xi_p)$. This general problem can be instantiated similarly as the deterministic one. It can be solved using variants of the well-studied *sampling average approximation*

[4]The denominator in the right hand side of constraints (14) is a constant, i.e., these constraints do not introduce any non-linearity into the problem.

[5]Note that $\#P_{\preceq}$ induces a total order on $\mathcal{P}$.

**Optimization goal:**

$$\text{maximize } G(x_1,\ldots,x_p)+\mathbb{E}[\mathcal{H}(x_1,\ldots,x_p,\xi)]$$

**Subject to:**

$$\sum_{k=1}^{p}\texttt{cost}(\texttt{b}_k)x_k \le \mathcal{B}$$
$$x_k\in\{0,1\},\ 1\le k\le p$$

where $\mathcal{H}(x_1,\ldots,x_p,\xi)$ is the optimal value of the problem

$$\text{maximize } H(f_1,\ldots,f_m,z_1,\ldots,z_n)$$

**Subject to:**

$$f_j \ge \frac{\sum_{k=1}^{p}A_{kj}(1-\xi_k x_k)}{p},\ 1\le j\le m$$
$$f_j \le \sum_{k=1}^{p}A_{kj}(1-\xi_k x_k),\ 1\le j\le m$$
$$z_i \ge 1+\sum_{j=1}^{m}B_{ij}(f_j-1),\ 1\le i\le n$$
$$z_i \le \frac{\sum_{j=1}^{m}B_{ij}f_j}{\sum_{j=1}^{m}B_{ij}},\ 1\le i\le n$$
$$f_j\in\{0,1\},\ 1\le j\le m,\quad z_i\in\{0,1\},\ 1\le i\le n.$$

Figure 9. General stochastic integer programming problem

approach [16], or other efficient heuristics, e.g., as in [17]. An interested reader is referred to [17] for more details.

## V. Tool support and case study

To automate the methodology developed in this paper, we have implemented a tool, called OSEAD: *Optimal Strategies Extractor for Attack–Defense trees*. It is an open source tool written in Python, available at https://people.irisa.fr/Wojciech.Widel/suftware/osead.zip. The main objective of OSEAD is to solve various optimization problems on ADTrees, including the problems IV-B1 and IV-B2a.

Below, we discuss the performance of OSEAD while creating the defense semantics and explain how the tool supports the selection of an optimal set of countermeasures.

### A. OSEAD: automating the defense semantics construction

The discussion in Section III-C shows that the task of computing the defense semantics might be complex. To study how Algorithm 1 performs in practice, we have implemented it in OSEAD. Our tool takes as input an ADTree written in the xml format generated using ADTool (standard software for creation and manipulation of ADTrees [18]) and extracts its defense semantics according to Algorithm 1. We tested the performance of our implementation on a number of synthetic ADTrees having various structure. The files storing our test trees are available at https://github.com/wwidel/defsem-tests, and an excerpt from the tests is given in Table I.

We remark that the number of strategies and the size of the defense semantics of an ADTree depend strongly on the structure of the tree (configurations of AND and OR nodes, number of distinct homogeneous subdags, number of clones, etc.), and less on the overall number of its nodes. This fact is illustrated in particular by the trees *random3* and *random4*: while both trees have the same number of nodes and similar numbers of basic actions of the actors, the differences in their structures translate into different sizes of objects created by Algorithm 1, and, in consequence, into drastically different running times.

### B. OSEAD: automating the optimal countermeasures selection

Amongst all its functionalities, OSEAD implements the selection of optimal sets of countermeasures, as formalized in Section IV. Our software can solve the *coverage problem*, as well as determine a set of countermeasures that *counter the cheapest strategies of the proponent*. To do so, the user needs to provide an ADTree and the necessary input values: opponent's overall budget, and costs of basic actions. After extracting the defense semantics from the ADTree, OSEAD translates it into an integer linear optimization problem. The latter is then solved by the LP_SOLVE solver [19] which works as a back-end of OSEAD, in a way transparent for the user. OSEAD returns its analysis results in a form of a textual file, listing the elements of an optimal set of countermeasures and recalling all input values that have been used. This allows a user to perform several simulations, for instance by varying the overall budget used for securing a system, in order to find the most suitable solution.

### C. OSEAD: real-life case study

We used OSEAD to validate our theoretical results on a real-life case study borrowed from industry, and described in [20]. The objective of this work was to analyze the scenario of tampering with an optical power meter to lower the recorded electricity consumption. Possible attacks and countermeasures were represented with an ADTree containing 68 nodes, including 5 repeated basic actions of the attacker and 3 repeated basic actions of the defender. OSEAD computed the defense semantics of this tree instantaneously, as highlighted by the blue frame in Figure 10. Translating the defense semantics into an optimization problem and solving it took OSEAD less than 1 second, as shown the red frame in Figure 10.

Interested reader will find additional details on the OSEAD tool and the tampering with a power meter case study in [20], where we also address the problem of estimating the necessary input values and discuss the reliability of the ADTree analysis, in general.

## VI. Related work

Plethora of attack tree-based models for analyzing security scenarios involving an attacker and a defender exist [21], [22], [23]. Below, we briefly discuss only some of the works that are closely related to our setting. Interested reader is referred to the recent survey [24], which provides a detailed comparison of

Table I
PARAMETERS OF INTEREST FOR THE COMPUTATION OF THE DEFENSE SEMANTICS ON SOME EXAMPLE ADTREES

| name of file storing tree $T$ | total number of nodes in $T$ | $\lvert\mathbb{B}^{\mathrm{P}T}\rvert$ | $\lvert\mathbb{B}^{\mathrm{O}T}\rvert$ | number of opponent's strategies in $T$ | $\lvert\mathtt{SuffWit}(T,\beta_T^{\mathtt{SW}})\rvert$ | number of proponent's strategies in $T$ | $\lvert\mathcal{D}(T)\rvert$ | performance of Algorithm 1 in sec. |
|---|---|---|---|---|---|---|---|---|
| synthetic1 | 30 | 16 | 8 | 256 | 256 | 192 | 384 | 1 |
| synthetic2 | 36 | 20 | 10 | 1024 | 1024 | 1024 | 2560 | 1 |
| synthetic3 | 44 | 24 | 12 | 4096 | 4096 | 3072 | 9216 | 83 |
| random1 | 30 | 8 | 14 | 2048 | 1536 | 36 | 168 | 1 |
| random2 | 48 | 12 | 23 | 32768 | 32768 | 22 | 92 | 17 |
| random3 | 94 | 36 | 30 | 2048 | 672 | 156 | 642 | 1 |
| random4 | 94 | 32 | 36 | $2^{21}$ | 147792 | 1376 | 11940 | 203 |

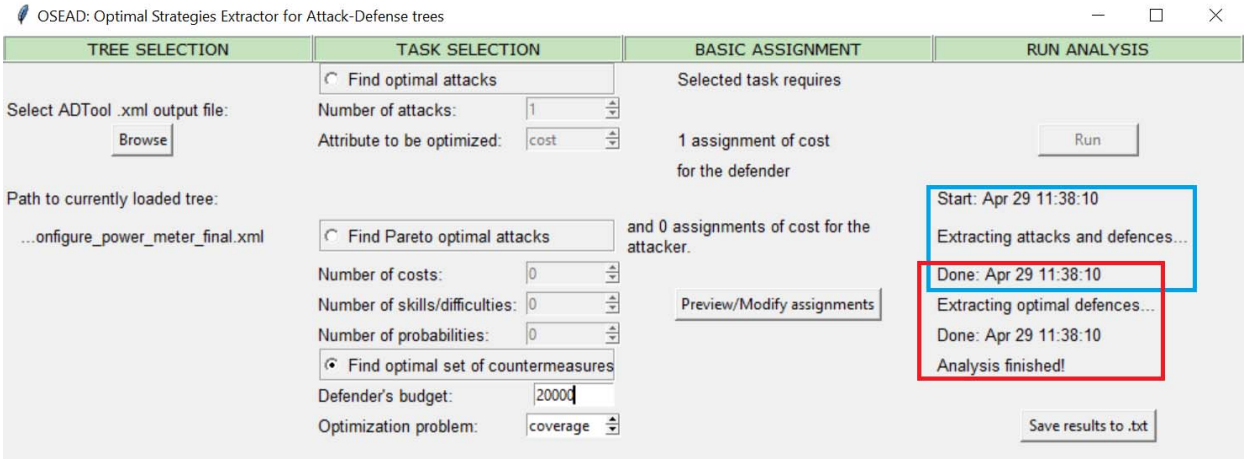


Figure 10. Extracting the defense semantics and solving the coverage problem on the tree from the case study [20], using the OSEAD tool

some solutions addressing the problem of optimal selection of countermeasures against potential attacks, including a number of frameworks based on attack trees and attack graphs.

This work builds upon [1], where the defense semantics has been introduced for the first time, and where an algorithm for its construction on ADTrees with no clones was developed. As can be seen in Section III, allowing for clones makes the reasoning about attack and defense strategies much more difficult, thus, the main research challenge in the current work was to devise a constructive method for the defense semantics in this general, and more realistic case. Furthermore, in [1] we formulated the *coverage* and the *countering the cheapest attack strategies* problems, but we did not consider their generalizations nor cases involving probability.

In [25], Roy et al. use attack countermeasure trees, which are attack trees augmented with countermeasure nodes composed of a detective and a mitigating parts. They exploit what can be seen as our defense semantics in a case of a single homogeneous subtree of the attacker, to which countermeasures are attached. The authors propose an ILP-based solution to the problem of *minimizing defender's investment* while covering some of the attack strategies, and *maximizing the defender's return on investment* (ROI). While the former can be applied to trees containing clones, the latter cannot, as it relies on the bottom-up computation of success probability that is known not to work in trees with clones. The main research focus of [25] is on algorithms for solving the optimization problems, while we mostly concentrate on extracting the defense semantics from the general model of ADTrees.

In [26], Muller et al. propose a branch-and-bound algorithm for *maximizing defender's ROI*. Their scenarios are modeled with attack–defense trees without clones of the attacker and where the countermeasures cannot be refined. In [27], Sendi et al. identify countermeasures that *maximize the security performance*, *minimize the attack impact*, and *minimize the defense cost*. Contrary to our work, neither [26] nor [27] allow for counterattacking the countermeasures.

Each of the approaches presented above is devoted to solving optimization problems on some variants of attack–defense trees. These variants are however considerably less expressive than the model used in this article, and thus significantly easier to analyze.

The complexity of our framework originates from the fact that the dependencies between basic actions of the actor are *encoded* in ADTrees; they are complex, and to make use of them, one needs to decode them. In the approaches developed for optimal selection of countermeasures by Brown et al. in [28] and by Khouzani et al. in [29], the relations between

behaviors of the actors are significantly simpler: in [28], every attacker's actions disables a set of defender's actions, and in [29], every defender's action impacts success probability of each of the attacker's actions. This simplicity allows for an immediate formulation of the optimization problems as bilevel mixed integer programming (MIP) programs [30], [31], which can be solved using standard methods. We note that, in the light of definition of achievement, the root node of an ADTree being achieved corresponds to a propositional formula being satisfied. The optimization problems considered in our work could thus have been expressed as variants of the satisfiability problem, which in turn could be directly encoded as MIP programs [32], [33]. Since the goals of the actors are conflicting (e.g., the attacker wants to minimize, and the defender wants to maximize the value of the objective function), and the defender is the first one to act, the result of such encoding would be a bilevel MIP problem resembling the ones considered in [28], [29]. A standard technique for dealing with bilevel programs involves replacing the inner problem with its dual or the dual of its linear relaxation [31], [28], [29]. If the inner problem is a linear programming problem or the integrality gap of its linear relaxation is 1, one eventually obtains a single level optimization problem equivalent to the initial one. In our case, the integrality gap is greater than 1, i.e., the difference between the optimal solution of the final program and the optimal solution of the initial one cannot be predicted. Therefore, even though this method would allow for omitting the computationally expensive construction of the defense semantics, we decided to pursue the approach yielding the exact optimal solutions. We believe that our framework could thus play an important role in assessing performance of heuristic methods for optimal selection of countermeasures in ADTrees developed in the future.

Artificial intelligence, and more precisely Stackelberg planning, has also been recently applied to address the optimization problems on large infrastructures [34], [35], [36]. In [35] a formal model for Stackelberg planning game between a leader and the follower (each of which could be the attacker or the defender) has been introduced. The goal is to find the leader/follower equilibria where *the leader's objective is to minimize its own cost while maximizing the cost of the follower's best response.* The Stackelberg planning approach has been applied to the penetration testing setting, in [34]. First, critical attack paths, i.e., those that maximize the attack probability within a given budget of the attacker are identified, and then the dominant mitigation strategies, i.e., a sequence of defender's actions that reduce the attack probability while not increasing the defense cost are found. The planning-based solution was also used in [36] to perform mitigation analysis of an e-mail infrastructure. In this work, each attack is associated with a reward that can be seen as an indicator of the severity of the attack. The defender's actions lower the attacker's reward and are associated with a positive real cost. The objective is to find the mitigation strategies that *minimize the attacker's reward while investing as little as possible*. Since the size of an e-mail infrastructure may be huge, the main concern of [36] is efficiency, i.e., to lower the total number of the considered mitigation strategies to make the problem tractable.

Both, the planning-based frameworks and our integer programing-based solution address some security-relevant attacker/defender optimization problems. However, several substantial differences between the two approaches should be noted. On the one hand, the solutions of [34], [35], [36] focus on specific attributes (cost, attack probability, reward) and our optimization problems are expressed in a generic way, see Section IV-B2 and IV-B3, and can thus be applied to various optimization functions. On the other hand, we provide a method to find one optimal solution, and the planning-based approach returns the entire Pareto frontier. While our main concern is to extract relevant attack and defense strategies from an industrially-relevant model, in the planning-based setting such strategies are given. In ADTrees, the subtrees rooted in a node belonging to one of the actors can be attached to any node of the other actor, including its refined nodes. The countermeasures may also be refined and counterattacked. In addition, a cloned defense may impact several attacks at once. This implies that a thorough analysis of the entire tree is necessary to identify the defense strategies countering a given attack strategy. This is not the case in [34], [35], [36], where defender's actions mitigate directly the actions of the attacker and which are limited to one level exchange between an action and a counteraction, i.e., arbitrarily long chains of attacker and defender exchanges are not taken into account. Finally, the planning-based approaches operate on a network infrastructure model, similar to an attack graph. It is however important to note that attack graphs are conceptually different from attack tree-based models: the former represent possible system states (nodes) and their modifications due to the attacker's actions (transitions), while the latter focus on the refinement of an actor's (attacker and/or defender) goal into basic actions. The edges in an attack(–defense) tree do not correspond to the actions of the actors and there is no temporal aspect between the children and its parent node, as in the case of attack graphs.

## VII. Conclusion

The main goal of the work presented in this paper was to tackle the issue of determining optimal sets of countermeasures in attack–defense scenarios modeled with ADTrees. To this end, we developed a novel method for extracting rational behavior of the actors from ADTrees possibly containing clones and countermeasures against countermeasures. We illustrated how the information stored in the resulting defense semantics can be employed for formulating numerous optimization problems in terms of (stochastic) integer linear programming.

By basing our framework on ADTrees – a derivative of attack trees that security experts are already familiar with – we expect its smooth acceptance by the potential end-users. To this end, we also implemented our solution in a prototype tool, which helped us to validate its practical applicability.

Despite a promising running time of our tool, the approach still leaves room for improvement, especially w.r.t. the size

of the defense semantics. It would be worthwhile to study possible ways of increasing the efficiency of our framework, perhaps by means of developing methods for approximating the defense semantics, i.e., creating its smaller variants without significant loss in the information stored.

Recently proposed approaches to address optimization problems and based on Stackelberg planning seem to enjoy very good performance. An interesting direction to investigate would be to devise a method of translating an ADTree in a format suitable for planning, and study how the efficiency of the two approaches compares on real-life ADTrees.

## References


[1] B. Kordy and W. Wideł, "How well can I secure my system?" in *iFM*, ser. LNCS, vol. 10510. Springer, 2017, pp. 332–347.

[2] W. Wideł, "Formal modeling and quantitative analysis of security using attack–defense trees," Ph.D. dissertation, INSA Rennes, IRISA, France, December 2019.

[3] B. Kordy, S. Mauw, S. Radomirovic, and P. Schweitzer, "Attack–defense trees," *Journal of Logic and Computation*, vol. 24(1), pp. 55 – 87, 2014.

[4] EAC Advisory Board and Standards Board, "Election Operations Assessment – Threat Trees and Matrices and Threat Instance Risk Analyzer," 2009. [Online]. Available: https://www.eac.gov/sites/default/files/eac_assets/1/28/Election_Operations_Assessment_Threat_Trees_and_Matrices_and_Threat_Instance_Risk_Analyzer_(TIRA).pdf

[5] National Electric Sector Cybersecurity Organization Resource (NESCOR), "Analysis of selected electric sector high risk failure scenarios, version 2.0," 2015. [Online]. Available: http://smartgrid.epri.com/doc/NESCOR%20Detailed%20Failure%20Scenarios%20v2.pdf

[6] A. Bossuat and B. Kordy, "Evil Twins: Handling Repetitions in Attack–Defense Trees – A Survival Guide," in *GraMSec 2017*, ser. LNCS, vol. 10744. Springer, 2018, pp. 17–37.

[7] B. Kordy, S. Mauw, and P. Schweitzer, "Quantitative Questions on Attack–Defense Trees," in *ICISC*, ser. LNCS, vol. 7839. Springer, 2012, pp. 49–64.

[8] B. Kordy and W. Wideł, "On quantitative analysis of attack–defense trees with repeated labels," in *POST*, ser. LNCS, vol. 10804. Springer, 2018, pp. 325–346.

[9] B. Fila and W. Wideł, "Efficient attack–defense tree analysis using Pareto attribute domains," in *CSF*. IEEE Computer Society, 2019, pp. 200–215.

[10] S. Mauw and M. Oostdijk, "Foundations of Attack Trees," in *ICISC 2005*, ser. LNCS, vol. 3935. Springer, 2005, pp. 186–198.

[11] B. Kordy, M. Pouly, and P. Schweitzer, "Computational aspects of attack–defense trees," in *SIIS*, ser. LNCS, vol. 7053. Springer, 2011, pp. 103–116.

[12] Z. Aslanyan and F. Nielson, "Pareto Efficient Solutions of Attack–Defense Trees," in *POST*, ser. LNCS, vol. 9036. Springer, 2015, pp. 95–114.

[13] O. Gadyatskaya, R. R. Hansen, K. G. Larsen, A. Legay, M. C. Olesen, and D. B. Poulsen, "Modelling Attack–defense Trees Using Timed Automata," in *FORMATS*, ser. LNCS, vol. 9884. Springer, 2016, pp. 35–50.

[14] B. Fila and W. Wideł, "On optimal countermeasure selection in attack–defense scenarios (extended version with proofs)," 2020. [Online]. Available: http://people.irisa.fr/Barbara.Kordy/papers/CSF20_extended.pdf

[15] A. Shapiro, D. Dentcheva, and A. Ruszczynski, *Lectures on Stochastic Programming - Modeling and Theory, Second Edition*, ser. MOS-SIAM Series on Optimization. SIAM, 2014, vol. 16.

[16] A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello, "The sample average approximation method for stochastic discrete optimization," *SIAM J. on Optimization*, vol. 12, no. 2, pp. 479–502, 2002.

[17] K. Zheng, L. A. Albert, J. R. Luedtke, and E. Towle, "A budgeted maximum multiple coverage model for cybersecurity planning and management," *IISE Transactions*, vol. 51, no. 12, pp. 1303–1317, 2019.

[18] O. Gadyatskaya, R. Jhawar, P. Kordy, K. Lounis, S. Mauw, and R. Trujillo-Rasua, "Attack trees for practical security assessment: Ranking of attack scenarios with ADTool 2.0," in *QEST*, ser. LNCS, vol. 9826. Springer, 2016, pp. 159–162.

[19] M. Berkelaar, K. Eikland, and P. Notebaert, "lp_solve: Open source (Mixed-Integer) Linear Programming system," 2005, http://lpsolve.sourceforge.net/5.5/, Version 5.5.2.5, from 2016-09-24.

[20] B. Fila and W. Wideł, "Attack–defense trees for abusing optical power meters: A case study and the OSEAD tool experience report," in *GraMSec*, ser. LNCS, vol. 11720. Springer, 2019, pp. 95–125.

[21] B. Kordy, L. Pièrre-Cambacédès, and P. Schweitzer, "DAG-based attack and defense modeling: Don't miss the forest for the attack trees," *Computer Science Review*, vol. 13-14, pp. 1–38, 2014.

[22] J. B. Hong, D. S. Kim, C. Chung, and D. Huang, "A survey on the usability and practical applications of graphical security models," *Computer Science Review*, vol. 26, pp. 1–16, 2017.

[23] W. Wideł, M. Audinot, B. Fila, and S. Pinchinat, "Beyond 2014: Formal methods for attack tree–based security modeling," *ACM Computing Surveys*, vol. 52, no. 4, pp. 75:1–75:36, Aug. 2019.

[24] P. Nespoli, D. Papamartzivanos, F. G. Mármol, and G. Kambourakis, "Optimal countermeasures selection against cyber attacks: A comprehensive survey on reaction frameworks," *IEEE Communications Surveys Tutorials*, vol. 20, no. 2, pp. 1361–1396, 2018.

[25] A. Roy, D. S. Kim, and K. S. Trivedi, "Scalable optimal countermeasure selection using implicit enumeration on attack countermeasure trees," in *DSN*. IEEE Computer Society, 2012, pp. 1–12.

[26] S. Muller, C. Harpes, and C. Muller, "Fast and optimal countermeasure selection for attack defence trees," in *RISK*, ser. LNCS, vol. 10224, 2016, pp. 53–65.

[27] A. S. Sendi, H. Louafi, W. He, and M. Cheriet, "Dynamic optimal countermeasure selection for intrusion response system," *IEEE Trans. Dependable Sec. Comput.*, vol. 15, no. 5, pp. 755–770, 2018.

[28] G. G. Brown, W. M. Carlyle, J. Salmerón, and R. K. Wood, "Defending critical infrastructure," *Interfaces*, vol. 36, no. 6, pp. 530–544, 2006.

[29] M. H. R. Khouzani, Z. Liu, and P. Malacaria, "Scalable min-max multi-objective cyber-security optimisation over probabilistic attack graphs," *European Journal of Operational Research*, vol. 278, no. 3, pp. 894–903, 2019.

[30] J. Moore and J. Bard, "The mixed integer linear bilevel programming problem," *Operations Research*, vol. 38, pp. 911–921, 10 1990.

[31] R. Wood, "Deterministic network interdiction," *Mathematical and Computer Modelling*, vol. 17, no. 2, pp. 1 – 18, 1993.

[32] J. N. Hooker, "A quantitative approach to logical inference," *Decision Support Systems*, vol. 4, no. 1, pp. 45–69, 1988.

[33] W. Guo, J. Wang, M. He, X. Ren, Q. Wang, and W. Tian, "An efficient method to transform a sat problem to a mixed integer linear programming problem," in *ICCC*, 2018, pp. 1992–1996.

[34] P. Speicher, M. Steinmetz, J. Hoffmann, M. Backes, and R. Künnemann, "Towards automated network mitigation analysis," in *SAC*. ACM, 2019, pp. 1971–1978.

[35] P. Speicher, M. Steinmetz, M. Backes, J. Hoffmann, and R. Künnemann, "Stackelberg planning: Towards effective leader-follower state space search," in *AAAI*. AAAI Press, 2018, pp. 6286–6293.

[36] P. Speicher, M. Steinmetz, R. Künnemann, M. Simeonovski, G. Pellegrino, J. Hoffmann, and M. Backes, "Formally reasoning about the cost and efficacy of securing the email infrastructure," in *EuroS&P*. IEEE, 2018, pp. 77–91.