2021 IEEE 34th Computer Security Foundations Symposium (CSF) | 978-1-7281-7607-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/CSF51468.2021.00041

Efficient Algorithms for Quantitative Attack Tree Analysis

Carlos E. Budde* Mariëlle Stoelinga*†
*University of Twente, Formal Methods and Tools, Enschede, the Netherlands.
†Radboud University, Department of Software Science, Nijmegen, the Netherlands.
{c.e.budde,m.i.a.stoelinga}@utwente.nl

***Abstract*—Numerous analysis methods for quantitative attack tree analysis have been proposed. These algorithms compute relevant security metrics, i.e. performance indicators that quantify how good the security of a system is, such as the most likely attack, the cheapest, or the most damaging one. This paper classifies attack trees in two dimensions: proper trees vs. directed acyclic graphs (i.e. with shared subtrees); and static vs. dynamic gates. For each class, we propose novel algorithms that work over a generic attribute domain, encompassing a large number of concrete security metrics defined on the attack tree semantics. We also analyse the computational complexity of our methods.**

I. Introduction

Attack trees are a popular method in decision making for security, supporting the identification, documentation and analysis of cyberattacks. They are part of many system engineering frameworks, e.g. *UMLsec* [1] and *SysMLsec* [2], and are supported by industrial tools such as Isograph's *AttackTree* [3].

An attack tree (AT) is a hierarchical diagram to systematically map potential attack scenarios of a system, see Figs. 1 and 2. The root at the top of the diagram models the attacker's goal, which is further refined into subgoals by means of gates: an AND gate indicates that an attack is successful iff all children attacks succeed; an OR gate indicates that any single child suffices. The leaves of the tree are basic attack steps (BAS), which model indivisible actions such as cutting a wire.

Static vs. dynamic attack trees: Extensions of classic ATs include the sequential-AND gate (SAND), indicating that subgoals must succeed in order from left to right [4, 5]. ATs without SAND gates are called *static*; those with SANDs are called *dynamic*. A formal approach requires different semantics to these two categories, as we explain below.

Tree vs. DAG attack trees: Despite their name, ATs are directed acyclic graphs (DAGs) rather than trees, since subtrees can be shared by several parent nodes—see Fig. 2b. As elaborated below, DAG-structured ATs are computationally more challenging than those with a proper tree structure.

AT metrics: Besides learning the essential components and structure that constitute a feasible attack scenario, a vast number of algorithms have been developed to compute a wide range of *security metrics*. These metrics comprise key performance indicators (KPIs) that quantify relevant security features, such as the time, cost, and likelihood of different attack scenarios. KPIs serve several purposes, e.g. allowing to compare different design alternatives w.r.t. the desired security features; compute the effectiveness of defensive measures; verify whether a solution meets its security requirements; etc.

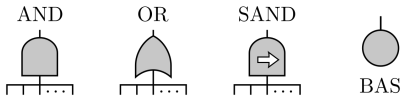


Figure 1: Nodes in an attack tree.

AT analysis: Numerous algorithms have been proposed to compute security metrics. These include methods to compute the cost and probability of an attack [6, 7], the time it takes [8, 9, 10], as well as Pareto analyses that study trade offs between different attributes [8, 11]. Such algorithms exploit a wide plethora of techniques, for instance Petri nets [12], model checking [5], and Bayesian networks [13]. While these algorithms provide good ways to compute metrics, they also suffer from several drawbacks: (1) Many of them are geared to specific attributes, such as attack time or probability, while the procedure could extend to other metrics; (2) Several algorithms do not exploit the acyclic structure of the AT, specially approaches based on model checking; (3) Since their application is mostly illustrated on small examples, it is unclear how these approaches scale to larger case studies.

Approach: We provide efficient and generic algorithms to compute AT metrics, by tailoring them to our 2-dimensional categorisation: static vs. dynamic ATs, and proper trees vs. DAG-structured ATs. These algorithms demand different semantics (that we provide) for dynamic attack trees.

Our algorithmic results are summarised in Table I. An elaborate comparison with related work is provided in Sec. IX.

Static trees: We start with the simplest category: static attack trees (SATs) with proper tree structure. As shown in a seminal paper by Mauw & Oosdijk [14], metrics can be computed for tree-structured SATs in a bottom-up fashion. This algorithm propagates values from the leaves to the top, using appropriate operators ∇ and $\triangle$ resp. for the OR and AND gates in the tree. We show this as Algo. 1. A key insight in [14] is that this procedure works whenever the algebraic structure $(V, \triangle, \nabla)$ constitutes a semiring. In particular, $\triangle$ must distribute over ∇.

We thank Sebastiaan Joosten for his help with the proof of Theo. 2; also Lars Kuijpers and Jarik Karsten for collaborations that led to our definition of ordering graphs and Algo. 3 resp. This work was partially funded by NWO project 15474 (*SEQUOIA*), and ERC Consolidator Grant 864075 (*CAESAR*).

Metric	Static tree	Dynamic tree	Static DAG	Dynamic DAG
min cost	BU [14, 15, 16]	BU [4]	MTBDD [17] $\mathcal{C}$-BU [18]	PTA [8]
min time	BU [14, 19]	APH [9] BU [4]	Petri nets [12]	PTA [8]
min skill	BU [14, 20]	BU [4]	$\mathcal{C}$-BU [18]	—
max damage	BU [14, 19, 20]	BU [4]	MTBDD [17] DPLL [7]	PTA [8]
probability	BU [6, 19]	APH [9]	BDD [21] DPLL [7]	I/O-IMC [5]
Pareto fronts	BU [22, 19]	**OPEN PROBLEM**	$\mathcal{C}$-BU [11]	PTA [8]
Any of the above	**Algo. 1:** $\texttt{BU}_{\texttt{SAT}}$	**Algo. 5:** $\texttt{BU}_{\texttt{DAT}}$	**Algo. 2:** $\texttt{BDD}_{\texttt{DAG}}$	**OPEN PROBLEM**
***k*-top metrics**	BU-projection [14]	**OPEN PROBLEM**	**Algo. 3:** `BDD shortest_paths`	**OPEN PROBLEM**

Table I: Efficient algorithms to compute security metrics on different AT classes (details in Sec. IX)

We provide an alternative proof of correctness for this result: while [14] deploys rewriting rules for attack trees, we work directly on the syntactic AT structure. Furthermore, we propose new classes of attribute domains, which extend the application of the bottom-up algorithm to compute popular security metrics, including stochastic and Pareto analyses.

Static DAGs: It is well-known that static attack trees with DAG structure cannot be analysed via a bottom-up procedure [21, 23]. Several algorithms have been devised to tackle with such ATs, mostly geared to specific metrics [5, 17, 12, 7, 18]. *A key contribution of this paper is a generic algorithm (Algo. 2) that works over any semiring attribute domain* $(V, \nabla, \triangle)$*, i.e. where* $\triangle$ *distributes over* ∇.

Concretely, we exploit a binary decision diagram representation (BDD) of the attack tree. Our algorithm visits each BDD node once and is thus linear in its size. The caveat is that BDDs can be of exponential size in the number of BAS, but one cannot hope for faster algorithms: as we show, computing a minimal attack is an NP-hard problem. Moreover, BDDs are known to be compact in practice [24], and allow parallel traversals [25], making them an overall efficient choice.

Dynamic trees: A challenge to compute metrics for dynamic attack trees (DATs) is to define them formally based on their semantics. Usually metrics are decoupled from semantics, and defined either on the syntactic AT structure, or ad hoc for the selected computation method [4, 8, 26, 18]. A main obstacle is to choose semantics for DATs in a way that supports a proper definition of metric, i.e. that is compatible with the notion of metric of static ATs, and that is generic as the attribute domains from [14]. In particular, the interaction among multiple SAND gates is nontrivial, because they may impose conflicting execution orders on the BAS of the tree. *One of our key contributions is to define a notion of well-formedness that rules out conflicting requirements.*

We give semantics to well-formed DATs in terms of partially ordered sets (*posets*). Each poset $\langle A, \prec \rangle$ represents an attack scenario, where A collects all attacks steps to be performed, and $a \prec b$ indicates that step a must be completed before step b starts. This set up enables us to define a notion of metric for DATs based on their semantics. *We then show that tree-structured DATs are analysable by extending the bottom-up algorithm with an additional operator (see Algo. 5).* Concretely, we use attribute domains with three operators: ∇, $\triangle$, $\rhd$, where $\rhd$ distributes over ∇ and $\triangle$, and $\triangle$ over ∇. We prove this algorithm correct in our formal semantics. Note that earlier algorithms do not provide explicit correctness results in terms of semantics. Our result is non-trivial, because the metrics are formally defined on the (poset) semantics of a DAT, while the algorithm works on its syntactic AT structure.

Dynamic DAGs: Efficient computation of metrics for DAG-structured DATs is left as future research challenge. A naïve, inefficient algorithm would enumerate all posets in the semantics. Instead, one could extend BDD-algorithms for static DAGs to dynamic ATs. This is non-trivial: BDDs ignore the order of attack steps. Thus, efficient analysis of DAG-structured dynamic ATs is an important open problem.

Contributions: In summary, our contributions are:

1. An efficient and generic BDD-based algorithm for DAG-SATs, working for semiring attribute domains $(V, \nabla, \triangle)$;
2. A theorem proving that computing a minimal successful attack is NP-hard;
3. An algorithm to compute the k-top best attacks;
4. A novel and intuitive poset semantics for dynamic attack trees that better matches the order behavior of SANDs;
5. A bottom-up algorithm for tree-structured DATs;
6. Future directions to analyse DAG-DATs efficiently (identified as an open problem).

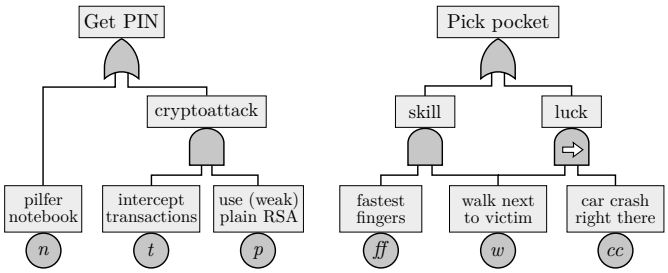
We place ourselves in the literature in Table I and Sec. IX.

Paper structure: We introduce all essential concepts and our formal syntax of attack trees in Sec. II. Secs. III to V study static ATs, and Secs. VI to VIII study dynamic ATs. The paper concludes in Sec. IX, revising related work.

II. Attack Trees

A. Attack tree models

Syntactically, an attack tree is a rooted DAG that models an undesired event caused by a malicious party, e.g. a security



(a) A static-tree AT: T_s (b) A dynamic-DAG AT: T_d

Figure 2: Attack tree models

breach. ATs show a top-down decomposition of a top-level attack—the unique root of the DAG—into simpler steps. The leaves are basic steps carried out by the attacker. The nodes between the basic steps and the root are intermediate attacks, and are labelled with gates to indicate how its input nodes (children) combine to make the intermediate attack succeed.

Basic Attack Steps: The leaves of the AT represent indivisible actions carried out by the attacker, e.g. smash a window, decrypt a file by brute-force attack, etc. These BAS nodes can be enriched with attributes, such as its execution time, the cost incurred, and the probability with which the BAS occurs. We model attributes via an attribution function $\alpha\colon \mathrm{BAS} \to V$.

Gates: Non-leaf nodes serve to model intermediate attacks that lead to the *top-level attack* (TLA). Each has a logical *gate* that describes how its children combine to make it succeed: an OR gate means that the intermediate attack will succeed if any of its child nodes succeeds; an AND gate indicates that all children must succeed, in any order or possibly in parallel; a SAND gate (viz. sequential-AND) needs all children to succeed sequentially in a left-to-right order.

Example 1. Fig. 2a shows a static attack tree, T_s, that models how a PIN code can be obtained by either pilfering a notebook, or via a cryptographic attack. The pilfering is considered atomic, while the cryptoattack consists of two steps which must both succeed: intercepting transactions, and abusing weak RSA encryption. Note that T_s has a plain tree structure. Instead, Fig. 2b shows a dynamic attack tree, T_d, with a DAG structure. Its TLA is to pick a pocket, which is achieved either by having "skill" or "luck." In both cases the attacker must walk next to the victim, so these gates share the BAS child w, making T_d not a tree. In the case of "luck" the order of events matters: if the attacker first walks next to the victim and then a traffic accident happens, the pick-pocket succeeds. Thus, this intermediate attack is modelled with a SAND gate. Instead, "fastest fingers" is an inherent attacker flair that is always present. It is thus meaningless to speak of an order w.r.t. the attacker-victim encounter, so an AND gate is used.

Security metrics: A key goal in quantitative security analysis is to compute relevant *security metrics*, which quantify how well a system performs in terms of security. Typical examples are the cost of the cheapest attack, the probability of the most likely one, the damage produced by the most harmful one, and combinations thereof. Security metrics for ATs are typically obtained by combining the attribute valuations $\alpha(a) \in V$ assigned to the BAS. For example, the cheapest attack is the attack where the sum of the cost of the BAS is minimal. *The key topic of this paper is to compute a large class of security metrics in a generic and efficient way*. For this we give different (formal) semantics to static and dynamic ATs, and introduce algorithms based on these semantics and the AT structure. We begin by formalising the notion of AT model.

B. Attack tree syntax

ATs are rooted DAGs with typed nodes: we consider types $\mathbb{T} = \{\mathtt{BAS}, \mathtt{OR}, \mathtt{AND}, \mathtt{SAND}\}$. For Booleans we use $\mathbb{B} = \{\mathtt{1}, \mathtt{0}\}$. The edges of an AT are given by a function ch that assigns to each node its (possibly empty) sequence of children. We use set notation for sequences, e.g. $e \in (e_1, \ldots, e_m)$ means $\exists i.\, e_i = e$, and we denote the empty sequence by ε.

Definition 1. An *attack tree* is a tuple $T = (N, t, ch)$ where:

- N is a finite set of *nodes*;
- $t\colon N \to \mathbb{T}$ gives the *type* of each node;
- $ch\colon N \to N^*$ gives the sequence of *children* of a node.

Moreover, T satisfies the following constraints:

- (N, E) is a connected DAG, where $E = \big\{(v, u) \in N^2 \mid u \in ch(v)\big\}$;
- T has a unique root, denoted R_T: $\exists!\, R_T \in N.\ \forall v \in N.\ R_T \notin ch(v)$;
- BAS_T nodes are the leaves of T: $\forall v \in N.\ t(v) = \mathtt{BAS} \Leftrightarrow ch(v) = \varepsilon$.

We omit the subindex T if no ambiguity arises, e.g. an attack tree $T = (N, t, ch)$ defines a set $\mathrm{BAS} \subseteq N$ of basic attack steps. If $u \in ch(v)$ then u is called a *child* of v, and v is a *parent* of u. Moreover we write $v = \mathrm{AND}(v_1, \ldots, v_n)$ if $t(v) = \mathtt{AND}$ and $ch(v) = (v_1, \ldots, v_n)$, and analogously for OR and SAND. We denote the universe of ATs by $\mathscr{T}$ and call $T \in \mathscr{T}$ *tree-structured* if $\forall v, u \in N.\ ch(v) \cap ch(u) = \varepsilon$; else we say that T has *DAG structure*.

III. Analysis of Static Attack Trees

In the absence of SAND gates the order of execution of the BAS is irrelevant. This allows for simple semantics given in terms of a Boolean function called *structure function*. The computation of security metrics, however, crucially depends on whether the AT structure is a tree or a DAG.

A. Semantics for static attack trees

The semantics of a static attack tree (SAT) is defined by its successful attack scenarios, in turn given by its structure function. First, we define the notions of attack and attack suite.

Definition 2. An *attack scenario*, or shortly an *attack*, of a static AT T is a subset of its basic attack steps: $A \subseteq \mathrm{BAS}_T$. An *attack suite* is a set of attacks $\mathcal{S} \subseteq 2^{\mathrm{BAS}_T}$. We denote by $\mathscr{A}_T = 2^{\mathrm{BAS}_T}$ the universe of attacks of T, and by $\mathscr{S}_T = 2^{2^{\mathrm{BAS}}}$ the universe of attack suites of T.

Intuitively, an attack suite $\mathcal{S} \in \mathbb{S}$ represents different ways in which the system can be compromised. From those, one is interested in attacks $A \in \mathcal{S}$ that actually represent a threat. For instance for T_s in Example 1 one such attack is $\{t, p\}$. In contrast, $\{t\}$ is an attack that does not succeed, i.e. it cannot cause a TLA. The structure function $f_T(v, A)$ indicates whether the attack $A \in \mathscr{A}$ succeeds at node $v \in N$ of T.

Definition 3. The *structure function* $f_T \colon N \times \mathscr{A} \to \mathbb{B}$ of a static attack tree T is given by:

$$f_T(v,A) = \begin{cases} 1 & \text{if } t(v) = \texttt{OR} \text{ and } \exists u \in ch(v).\, f_T(u,A) = 1, \\ 1 & \text{if } t(v) = \texttt{AND} \text{ and } \forall u \in ch(v).\, f_T(u,A) = 1, \\ 1 & \text{if } t(v) = \texttt{BAS} \text{ and } v \in A, \\ 0 & \text{otherwise.} \end{cases}$$

We let $f_T(A) \doteq f_T(R_T, A)$. An attack A is called *successful* if $f_T(A) = 1$, i.e. it makes the TLA of T succeed; if moreover no proper subset of A is successful then A is a *minimal attack*.

SATs are *coherent* [27], meaning that adding attack steps preserves success: if A is successful then so is $A \cup \{a\}$ for any $a \in \mathrm{BAS}$. Thus, the suite of successful attacks of an AT is characterised by its minimal attacks. This was first formalised in [14], and is called multiset semantics in [28]:

Definition 4. The *semantics of a static* AT T is its suite of minimal attacks: $[\![T]\!] = \{A \in \mathscr{A}_T \mid f_T(A) \wedge A \text{ is minimal}\}$.

Example 2. The static AT in Example 1, T_s (Fig. 2a), has three successful attacks: $\{n\}$, $\{t, p\}$, and $\{n, t, p\}$. The first two are minimal, so we have: $[\![T_s]\!] = \{\{n\}, \{t, p\}\}$.

An alternative characterisation of this semantics for tree-structured SATs is shown as Lemma 1, which also provides the key argument for correctness of the bottom-up procedure (Algo. 1 in Sec. IV). Lemma 1 can be used to compute the semantics of Def. 4 by recursively applying cases *1)–3)* to $[\![R_T]\!] \doteq [\![T]\!]$. However, BDD representations provide more compact encodings of this semantics (see Sec. V).

We formulate Lemma 1 for binary ATs; its extension to arbitrary trees is straightforward but notationally cumbersome. The proof uses induction on the structure of an AT, whose root is the node v in each left-hand side $[\![v]\!]$ of cases *1)–3)*. Case *4)* is trivial by the tree-structure. We give the full proof in [29].

Lemma 1. *Consider a* SAT *with nodes* $a \in \mathrm{BAS}, v_1, v_2 \in N$, *that has a proper tree structure. Then:*

1) $[\![a]\!] = \{\{a\}\}$;
2) $[\![\mathrm{OR}(v_1, v_2)]\!] = [\![v_1]\!] \cup [\![v_2]\!]$;
3) $[\![\mathrm{AND}(v_1, v_2)]\!] = \{A_1 \cup A_2 \mid A_1 \in [\![v_1]\!] \wedge A_2 \in [\![v_2]\!]\}$;
4) In cases 2) and 3) the $[\![v_i]\!]$ *are disjoint, and in case 3) moreover the* A_i *are pairwise disjoint.*

B. Security metrics for static attack trees

Lemma 1 allows for *qualitative analyses*, i.e. finding the minimal sets of BAS that lead to a TLA. To enable *quantitative analyses*, i.e. computing security metrics such as the minimal time and cost among all attacks, all BAS are enriched with attributes. We thus define security metrics in three steps: first an attribution α assigns a value to each BAS; then a security metric $\widehat{\alpha}$ assigns a value to each attack scenario; and finally the metric $\breve{\alpha}$ assigns a value to each attack suite.

Definition 5. Given an AT and a set V of values:

1) an *attribution* $\alpha \colon \mathrm{BAS} \to V$ assigns an *attribute value* $\alpha(a)$, or shortly an *attribute*, to each basic attack step a;
2) a *security metric* refers both to a function $\widehat{\alpha} \colon \mathscr{A}_T \to V$ that assigns a value $\widehat{\alpha}(A)$ to each attack A;
and to a function $\breve{\alpha} \colon \mathbb{S}_T \to V$ that assigns a value $\breve{\alpha}(\mathcal{S})$ to each attack suite $\mathcal{S}$.

We write $\breve{\alpha}(T)$ for $\breve{\alpha}([\![T]\!])$, setting the metric of an AT to the metric of its minimal attack suites.

Example 3. Let $V = \mathbb{N}$ denote time, so that $\alpha(a)$ gives the time required to perform the basic attack step a. Then the time needed to complete an attack A can be given by $\widehat{\alpha}(A) = \sum_{a \in A} \alpha(a)$, and the time of the fastest attack in a suite $\mathcal{S}$ is $\breve{\alpha}(\mathcal{S}) = \min_{A \in \mathcal{S}} \widehat{\alpha}(A)$. If instead $V = [0, 1] \subset \mathbb{R}$ denotes probability, then the probability of an attack is given by $\widehat{\alpha}(A) = \prod_{a \in A} \alpha(a)$, and the probability of the likeliest attack in a suite is $\breve{\alpha}(\mathcal{S}) = \max_{A \in \mathcal{S}} \widehat{\alpha}(A)$.

Def. 5 gives a lax notion of metric. For a more concise definition—that enables computation for static ATs, but does not depend on their tree/DAG-structure—one must resort to the semantics. For this we follow an approach similar to that of Mauw and Oostdijk [14]. Namely, we define a *metric function* $\breve{\alpha} \colon \mathscr{T} \to V$ that yields a value for each SAT based on its semantics, an attribution, and two binary operators $\triangledown$ and $\triangle$.

Definition 6. Let V be a set:

1) an *attribute domain* over V is a tuple $D = (V, \triangledown, \triangle)$, whose *disjunctive operator* $\triangledown \colon V^2 \to V$, and *conjunctive operator* $\triangle \colon V^2 \to V$, are associative and commutative;
2) the attribute domain is a *semiring*[1] if $\triangle$ distributes over $\triangledown$, i.e. $\forall x, y, z \in V.\ x \triangle (y \triangledown z) = (x \triangle y) \triangledown (x \triangle z)$;
3) let T be a static AT and α an attribution on V. The *metric for* T associated to α and D is given by:

$$\breve{\alpha}(T) = \underbrace{\bigtriangledown_{A \in [\![T]\!]}}_{\breve{\alpha}} \underbrace{\bigtriangleup_{a \in A}}_{\widehat{\alpha}} \alpha(a).$$

Example 4. Consider the static AT $T_s = \mathrm{OR}\big(n, \mathrm{AND}(t, p)\big)$ from Fig. 2a, and recall that $[\![T_s]\!] = \{\{n\}, \{t, p\}\}$. Let $V = \mathbb{N}$ denote time as in Example 3, and consider an attribution $\alpha = \{n \mapsto 1, t \mapsto 100, p \mapsto 0\}$. Then the metric for the fastest attack time is given by the attribute domain $(V, \min, +)$:

$$\begin{aligned} \breve{\alpha}(T_s) &= \bigtriangledown_{A \in \{\{n\},\{t,p\}\}} \bigtriangleup_{a \in A} \alpha(a) \\ &= \alpha(n) \triangledown \big(\alpha(t) \triangle \alpha(p)\big) = 1 \min (100 + 0) = 1, \end{aligned}$$

where $\min$ has infix notation, i.e. $x \min y = \min(x, y)$. For probability, let $V' = [0, 1]$ and $\alpha' = \{n \mapsto 0.07, p \mapsto 0.01,$

[1]Since we require $\triangle$ to be commutative, D is in fact a commutative semiring. Further, rings often include a neutral element for disjunction and an absorbing element for conjunction, but these are not needed in Def. 6.

$t \mapsto 0.95\}$. Then the attribute domain $(V', \max, *)$ allows to compute the probability of the likeliest attack: $\breve{\alpha}'(T_s) = \alpha'(n) \mathbin{\triangledown'} \big(\alpha'(t) \mathbin{\triangle'} \alpha'(p)\big) = 0.07 \max (0.95 * 0.01) = 0.07$.

IV. Computations for tree-structured SATs

Example 4 illustrates how to compute metrics for static ATs using Def. 6. However, this method requires to first compute the semantics of the attack tree, which is *exponential* in the number of nodes $|N|$—see Theo. 2 in Sec. V, or [18].

A key result in [14] is that metrics defined on attribute domains $(V, \triangledown, \triangle)$ that are semirings, can be computed via a bottom-up algorithm that is *linear* in $|N|$ as long as the static AT has a proper tree structure. We repeat this result here, giving a more direct proof of correctness, and extending it to dynamic attack trees in Sec. V.

A. Bottom-up algorithm

First we formulate the procedure as Algo. 1, which propagates the attribute values from the leaves of the SAT to its root, interpreting OR gates as $\triangledown$ and ANDs as $\triangle$. This algorithm is clearly linear in $|N|$ since each node in the tree T is visited once. Algo. 1 can be called on any node of T: to compute the metric $\breve{\alpha}(T)$ it must be called on its root node R_T.

Input: Static attack tree $T = (N, t, ch)$,
node $v \in N$,
attribution α,
semiring attribute domain $D = (V, \triangledown, \triangle)$.
Output: Metric value $\breve{\alpha}(T) \in V$.

if $t(v) = \texttt{OR}$ **then**
  return $\bigtriangledown_{u \in ch(v)} \texttt{BU}_{\texttt{SAT}}(T, u, \alpha, D)$
else if $t(v) = \texttt{AND}$ **then**
  return $\bigtriangleup_{u \in ch(v)} \texttt{BU}_{\texttt{SAT}}(T, u, \alpha, D)$
else // $t(v) = \texttt{BAS}$
  return $\alpha(v)$

Algorithm 1: $\texttt{BU}_{\texttt{SAT}}$ for a tree-structured SAT T

We state the correctness of Algo. 1 in Theo. 1, whose proof relies on Lemma 1, and therefore works by structural induction on the (binary) tree T—we provide the full proof in [29]. This result was proven in [14] via rewriting rules for ATs with a slightly different structure denoted "bundles." Our result concerns attack trees in the syntax from Def. 1, which is more conforming to the broad literature [15, 16, 6, 7, 30, 17, 8].

Theorem 1. *Let T be a static* AT *with tree structure, α an attribution on V, and $D = (V, \triangledown, \triangle)$ a semiring attribute domain. Then $\breve{\alpha}(T) = \texttt{BU}_{\texttt{SAT}}(T, R_T, \alpha, D)$.*

B. Metrics as semiring attribute domains

Many relevant metrics for security analyses on SATs can be formulated as semiring attribute domains. Table II shows examples, where $[0,1]_{\mathbb{Q}} = [0,1] \cap \mathbb{Q}$, and $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$ includes 0 and ∞. For instance "min cost" can be formulated in terms of $(\mathbb{N}_\infty, \min, +)$, which is a semiring attribute domain because $+$ distributes over min, i.e. $a + (b \min c) = (a + b) \min (a + c)$ for all $a, b, c \in \mathbb{N}_\infty$. Also, attribute domains can handle SAND gates providing that the execution order is irrelevant for the metric. This works for example with min skill and max damage.

Metric	V	$\triangledown$	$\triangle$
min cost	$\mathbb{N}_\infty$	min	$+$
min time (sequential)	$\mathbb{N}_\infty$	min	$+$
min time (parallel)	$\mathbb{N}_\infty$	min	max
min skill	$\mathbb{N}_\infty$	min	max
max challenge	$\mathbb{N}_\infty$	max	max
max damage	$\mathbb{N}_\infty$	max	$+$
discrete prob.	$[0,1]_{\mathbb{Q}}$	max	$*$
continuous prob.	$\mathbb{R} \to [0,1]_{\mathbb{Q}}$	max	$*$

Table II: SAT metrics with semiring attribute domains

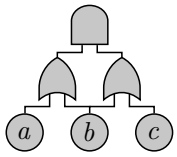
Non-semiring metrics: Nevertheless, some meaningful metrics do fall outside this category. For instance and as observed in [14], the cost to defend against all attacks is represented by $(\mathbb{N}_\infty, +, \min)$, but since min does not distribute over $+$ (i.e. in general $a \min (b + c) \neq (a \min b) + (a \min c)$) then this metric cannot be computed via Algo. 1. Less well-known is that the total attack probability—given by $\breve{\alpha}(T) = \sum_{A \in [\![T]\!]} \widehat{\alpha}(A)$ where $\widehat{\alpha}(A) = \big(\prod_{a \in A} \alpha(a)\big) \cdot \big(\prod_{a \notin A} (1 - \alpha(a))\big)$—can neither be formulated as an attribute domain. The problem is that $\widehat{\alpha}(A)$ does not have the shape $\bigtriangleup_{a \in A} \alpha(a)$. Interestingly though, this probability can still be computed via a bottom-up procedure by taking $\breve{\alpha}(\text{AND}(v_1, v_2)) = \breve{\alpha}([\![v_1]\!]) * \breve{\alpha}([\![v_2]\!])$ and $\breve{\alpha}(\text{OR}(v_1, v_2)) = \breve{\alpha}([\![v_1]\!]) + \breve{\alpha}([\![v_2]\!]) - \breve{\alpha}([\![v_1]\!] \cap [\![v_2]\!])$.

Stochastic analyses: Semirings are closed under finite and infinite products [31]: this allows to propagate not only tuples of attribute values, but also functions over them. In particular, *cumulative density functions* that assign a probability $t \mapsto P[X \leqslant t]$ constitute a semiring [9]. Such functions are useful, e.g. to consider attack probabilities, cost, or damage, as functions that evolve on time.

Pareto analyses: Moreover, *Pareto frontiers* can be formulated as semirings. Pareto analysis is a cornerstone in multi-parameter optimisation, that seeks the dominant (i.e. best-performing) solutions over multiple attributes. A solution is called *Pareto-efficient* if it is not dominated by any other solution in the ordering relation [32]. For example consider three attack scenarios: A_1 that takes 2 time units and has cost 3; A_2 with time 1 and cost 3; and A_3 with time 2 and cost 1. Then attack A_1 is not Pareto-efficient because A_2 is faster at same cost. On the other hand, A_2 and A_3 are incomparable because the former is faster while the latter is cheaper. So among these three attack scenarios, A_2 and A_3 are in the Pareto frontier. Pareto frontiers are sets of Pareto-efficient solutions: for AT metrics these are cross-products of semiring attribute domains, which preserve the semiring property [31].

V. Computations for DAG-structured SATs

Attack trees with shared subtrees cannot be analysed via a bottom-up procedure on its (DAG) structure, as we illustrate next in Example 5. This is a classical result from fault tree analysis [33], later rediscovered for attack trees e.g. in [18].



Let: $\alpha(a) = 3$ $\quad V = \mathbb{N}_\infty$
$\alpha(b) = 1$ $\quad \triangledown = \min$
$\alpha(c) = 4$ $\quad \triangle = +$
$\Big\}\ D$

The cheapest attack is $\{b\}$: $\widehat{\alpha}(\{b\}) = 1$.

Figure 3: Metrics cannot be computed bottom-up on ATs with DAG structure. For min cost in this static AT, Algo. 1 yields: $\mathtt{BU}_{\mathtt{SAT}}(T, R_T, \alpha, D) = (3 \min 1) + (1 \min 4) = 2 \neq 1 = \breve{\alpha}(T)$. The miscomputation stems from counting $\alpha(b)$ twice.

Various methods to analyse DAG-structured ATs have been proposed: see Table I for contributions over the last 15 years, including [5, 17, 12, 7, 18]. These methods are often geared towards specific metrics, e.g. cost, time, or probability [6, 7, 9]. Others use general-purpose techniques of high complexity and low efficiency, such as model checking [12, 8].

We present a novel algorithm based on a binary decision diagram (BDD) representation of the structure function of the static AT. BDDs offer a very compact encoding of Boolean functions, and are heavily used in model checking [34, 35, 36], as well as for probabilistic fault tree analysis [21, 37].

Our BDD-based approach works for semiring attribute domains (with neutral elements for the operators $\triangledown$ and $\triangle$) regardless of the AT structure. It thus extends the generic and efficient result of [14]—that works for tree-structure SATs only—to include DAG-structured SATs as well.

Our algorithm traverses the BDD bottom-up, which makes it linear in its size. BDDs, however, can be exponential in the tree size [38]. Below, we show that the problem of computing metrics is NP-hard, so no asymptotically-faster algorithms can be found. Moreover, BDDs are among the most efficient approaches in terms of practical performance [24, 17].

A. Computational complexity

We first show why the bottom-up procedure fails to compute metrics for ATs that have shared subtrees.

Example 5. Fig. 3 shows how the bottom-up approach can fail when applied to DAG-structured attack trees. Intuitively, the problem is that a visit to node v in Algo. 1—or any bottom-up procedure that operates on the AT structure—can only aggregate information on its descendants. So, the recursive call for v cannot determine whether a sibling node in the AT (i.e. any node v' which is not an ancestor nor a descendant of v) shares a BAS descendant with v. As a result, recursive computations for both v and v' may select a shared descendant $b \in \mathrm{BAS}$, and use $\alpha(b)$ in (both) their local computations. This causes the miscomputation in Fig. 3.

Workarounds to this issue include keeping track of the BAS selected at each step by the metric [18], and operating on the AT semantics [14]. In all cases the worst-case scenario has exponential complexity on the number of AT nodes: for [18] this is in the input of the algorithm, i.e. determining the sets of necessary and optional clones; for [14] and our Def. 6 the complexity lies in the computation of the semantics.

In general, one cannot hope for faster algorithms: Theo. 2 shows that the core problem—computing minimal attacks of DAG-structured attack trees—is NP-hard even in the simplest structure: plain attack trees with AND/OR gates. The proof (provided in full in [29]) reduces the satisfiability of logic formulae in conjunctive normal form, to the computation of minimal attacks in general SATs.

Theorem 2. *The problem of computing the smallest minimal attack of a DAG-structured static AT is NP-hard.*

Note that the attribute domain $(\mathbb{N}_\infty, \min, +)$ allows for an attribution α, s.t. the BAS that constitute the resulting metric can be extracted in polynomial time from its value. This observation underpins the following corollary of Theo. 2:

Corollary 1. *Computing a metric for an attribute domain in a DAG-structured SAT is NP-hard.*

B. Binary decision diagrams

BDDs offer an extremely compact representation of Boolean functions, whose size can grow linearly in the number of variables, i.e. the BAS of the AT [24]. Although this depends on the variable ordering, and there exist functions where every BDD is of exponential size, DAG-structures that represent Boolean functions—such as fault trees and ATs—often have small BDD encodings [17, 38].

A BDD is a rooted DAG B_f that, intuitively, represents a Boolean function $f\colon \mathbb{B}^n \to \mathbb{B}$ over variables $\mathit{Vars} = \{x_i\}_{i=1}^n$. The terminal nodes of B_f represent the outcomes of f: $\mathtt{0}$ or $\mathtt{1}$. A nonterminal node $w \in W$ represents a subfunction f_w of f via its Shannon expansion. That means that w is equipped with a variable $\mathit{Lab}(w) \in \mathit{Vars}$ and two children: $\mathit{Low}(w) \in W$, representing f_w in case that the variable $\mathit{Lab}(w)$ is set to $\mathtt{0}$; and $\mathit{High}(w)$, representing f_w if $\mathit{Lab}(w)$ is set to $\mathtt{1}$.

Definition 7. A *BDD* is a tuple $B = (W, \mathit{Low}, \mathit{High}, \mathit{Lab})$ over a set Vars where:

- The *set of nodes* W is partitioned into terminal nodes (W_t) and nonterminal nodes (W_n);
- $\mathit{Low}\colon W_n \to W$ maps each node to its *low child*;
- $\mathit{High}\colon W_n \to W$ maps each node to its *high child*;
- $\mathit{Lab}\colon W \to \{\mathtt{0}, \mathtt{1}\} \cup \mathit{Vars}$ maps terminal nodes to Booleans, and nonterminal nodes to variables:

$$\mathit{Lab}(w) \in \begin{cases} \{\mathtt{0}, \mathtt{1}\} & \text{if } w \in W_t, \\ \mathit{Vars} & \text{if } w \in W_n. \end{cases}$$

Moreover, B satisfies the following constraints:

- (W, E) is a connected DAG, where
$$E = \{(w, w') \in W^2 \mid w' \in \mathit{Low}(w) \cup \mathit{High}(w)\};$$
- B has a unique root, denoted R_B:
$$\exists!\, R_B \in W.\ \forall w \in W_n.\ R_B \notin \mathit{Low}(w) \cup \mathit{High}(w).$$

Reduced ordered BDDs: We operate with *reduced ordered BDDs*, simply denoted BDDs. This requires a total order $<$ over the variables. For Def. 7 this means that:

- Vars comes equipped with a total order, so B_f is actually defined over a pair $\langle \mathit{Vars}, < \rangle$;

- the variable of a node is of lower order than its children: $\forall\, w \in W_n.\ Lab(w) < Lab(Low(w)), Lab(High(w))$;
- the children of nonterminal nodes are distinct nodes;
- all terminal nodes are distinctly labelled.

This has the following consequences in the BDD:

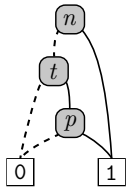
- there are exactly two terminal nodes: $W_t = \{\bot, \top\}$, with $Lab(\bot) = \mathtt{0}$ and $Lab(\top) = \mathtt{1}$;
- the label of the root node R_B has the lowest order;
- in any two paths from R_B to $\bot$ or $\top$, the variables appear in the same (increasing) order.

Encoding static ATs as BDDs: The key idea behind BDDs is that evaluating a Boolean function f on an input $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{B}^n$ is equivalent to following the corresponing path from R_B to a terminal node: when visiting node $w \in W_n$ with $x_i = Lab(w)$, the path goes to the child $Low(w)$ if $x_i = \mathtt{0}$ in $\boldsymbol{x}$; else it goes to $High(w)$. The result $f(\boldsymbol{x}) \in \mathbb{B}$ is the label of the terminal node reached.

This is used to encode fault trees as BDDs via their structure function [21], and extends to ATs by letting $\mathrm{BAS} = Vars$. Technically, this exploits the Boolean function $\boldsymbol{x} \mapsto f_T(A_{\boldsymbol{x}})$, where the attack $A_{\boldsymbol{x}}$ contains the BAS in whose position (determined by the total order $<$) the input $\boldsymbol{x}$ is $\mathtt{1}$.

Finally and importantly, since the metrics are defined on the set of minimal attacks of an AT T, the BDD B_T must exclusively represent the minimal attacks in T. This is achieved by using a variant of the Shannon expansion of the structure function f_T [38], which evaluates to $\mathtt{1}$ only when including the BAS which are essential for the current attack under consideration. Formally: $\boldsymbol{x} \mapsto \big(x_1 \wedge f(\boldsymbol{x}_1) \wedge \neg f(\overline{\boldsymbol{x}_1})\big) \vee \big(\neg\overline{x_1} \wedge f(\overline{\boldsymbol{x}_1})\big)$, where one has $\boldsymbol{x}_1 \doteq (\mathtt{1}, x_2, \ldots, x_n)$ and $\overline{\boldsymbol{x}_1} \doteq (\mathtt{0}, x_2, \ldots, x_n)$.

Example 6. Let $n < t < p$ in T_s from Example 1: the resulting BDD (B_{T_s}) is illustrated to the right. As usual, the children of a node appear below it (so the root node is on top), and a dashed line from w to a child w' means that $w' = Low(w)$, and a solid line means that $w' = High(w)$.



C. BDD-based algorithm for DAG-structured SATs

Algo. 2 computes metrics for DAG-structured trees given an attribute domain $D_\star = (V, \triangledown, \triangle, 1_\triangledown, 1_\triangle)$, where $1_\triangledown, 1_\triangle \in V$ are neutral elements for $\triangledown$ and $\triangle$: $\forall x \in V.\ 1_\triangledown \triangledown x = 1_\triangle \triangle x = x$. Also and just like $\mathtt{BU_{SAT}}$, Algo. 2 requires $(V, \triangledown, \triangle)$ in $D_\star$ to be a semiring attribute domain.

It is common for semiring definitions to require the presence of neutral elements [31]: they are needed for DAG-structured SATs, although not for tree-structured SATs. Examples of neutral elements in Table II are $1_\triangledown = \infty$ and $1_\triangle = 0$ for min cost, and $1_\triangledown = 0$ and $1_\triangle = 1$ for (max) discrete probability.

The algorithm: The idea behind Algo. 2 is to traverse the BDD top-down (or, equivalently, bottom-up), accumulating via $\triangle$ the values of the BASs included in the attack under consideration. For that, at each node w visited in the BDD B_T, $\mathtt{BDD_{DAG}}$ recursively computes the metric value $\breve{\alpha}(T_w)$ for the AT whose BDD B_{T_w} is the sub-BDD of B_T with root w.

So, starting at the root R_B of the BDD B_T, algorithm $\mathtt{BDD_{DAG}}$ considers the only two possible types of attack:

- Those that include $Lab(R_B) = v \in \mathrm{BAS}$:
  - the metric for this suite of attacks is computed in a recursive call of $\mathtt{BDD_{DAG}}$ on the child $High(R_B) = h$;
  - these attacks use $v \in \mathrm{BAS}$ so their metrics use $\alpha(v) \in V$, accumulated via $\triangle$ (which distributes over $\triangledown$);
  - the result is $\breve{\alpha}(T_h) \triangle \alpha(v) \in V$, where B_{T_h} represents the suite of attacks of T that require v to succeed.
- Those that exclude $Lab(R_B)$:
  - the metric is computed by recursion on $Low(R_B) = \ell$;
  - these attacks exclude v and therefore do not use $\alpha(v)$;
  - the result is $\breve{\alpha}(T_\ell) \in V$, where B_{T_ℓ} represents the suite of successful attacks of T that exclude v.
- The final metric for T is the disjunction of these the two recursive calls: $\breve{\alpha}(T_\ell) \triangledown \big(\breve{\alpha}(T_h) \triangle \alpha(v)\big)$.
- The base cases of the recursive calls are the BDD leaves:
  - $Lab(\bot) = \mathtt{0}$ is given the neutral element $1_\triangledown \in V$;
  - $Lab(\top) = \mathtt{1}$ is given the neutral element $1_\triangle \in V$.

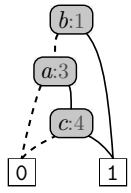
The pseudocode of this procedure is given as Algo. 2.

Input: BDD $B_T = (W, Low, High, Lab)$,
node $w \in W$,
attribution α,
semiring attribute domain $D_\star = (V, \triangledown, \triangle, 1_\triangledown, 1_\triangle)$.
Output: Metric value $\breve{\alpha}(T) \in V$.

if $Lab(w) = \mathtt{0}$ **then**
  return $1_\triangledown$
else if $Lab(w) = \mathtt{1}$ **then**
  return $1_\triangle$
else // $w \in W_n$
  return $\mathtt{BDD_{DAG}}(B_T, Low(w), \alpha, D_\star) \triangledown$
    $\big(\mathtt{BDD_{DAG}}(B_T, High(w), \alpha, D_\star) \triangle \alpha(Lab(w))\big)$

Algorithm 2: $\mathtt{BDD_{DAG}}$ for a DAG-structured SAT T

Example 7. For the DAG-structured SAT shown in Fig. 3, the order $b < a < c$ of its BAS yields the BDD to the right. To compute the min cost (like in Fig. 3) we employ the attribution $\alpha = \{a \mapsto 3, b \mapsto 1, c \mapsto 4\}$ and the domain $(\mathbb{N}_\infty, \min, +)$. Moreover, to use Algo. 2, we choose the neutral elements $1_\triangledown = \infty$ for min and $1_\triangle = 0$ for $+$, i.e. we use the attribute domain $D_\star = (\mathbb{N}_\infty, \min, +, \infty, 0)$. Let the nonterminal nodes of the BDD B_T be $W_n = \{w_a, w_b, w_c\}$. For $w \in W$ let $\mathtt{BU}(w) = \mathtt{BDD_{DAG}}(B_T, w, \alpha, D_\star)$, then we compute the metric:



$$\begin{aligned}
\mathtt{BU}(R_B) &= \mathtt{BU}(w_a) \min \big(\mathtt{BU}(\top) + \alpha(b)\big) \\
&= \mathtt{BU}(w_a) \min \big(1_\triangle + 1\big) \\
&= \mathtt{BU}(w_a) \min 1 \\
&= \big(\mathtt{BU}(\bot) \min \big(\mathtt{BU}(w_c) + \alpha(a)\big)\big) \min 1 \\
&= \big(1_\triangledown \min \big(\mathtt{BU}(w_c) + 3\big)\big) \min 1 \\
&= \big(\mathtt{BU}(w_c) + 3\big) \min 1
\end{aligned}$$

$$= \big((\texttt{BU}(\bot) \min (\texttt{BU}(\top) + \alpha(c))) + 3\big) \min 1$$
$$= \big((1_\triangledown \min (1_\triangle + 4)) + 3\big) \min 1$$
$$= (4 + 3) \min 1 \;=\; 1.$$

To compute instead the (max) discrete probability we use the attribution $\alpha' = \{a \mapsto 0.1, b \mapsto 0.05, c \mapsto 0.6\}$ and the attribute domain $D'_\star = ([0,1]_\mathbb{Q}, \max, *, 0, 1)$. Then computations are as before until the last line, which here becomes: $(\alpha'(c) * \alpha'(a)) \max \alpha'(b) = (0.6 * 0.1) \max 0.05 = 0.06$.

Theo. 3 states the correctness of Algo. 2, i.e. that it yields the metric for a static AT given in Def. 6 regardless of its structure. We prove Theo. 3 in [29], by induction in the number of levels of the BDD B_T: this is the cardinality of its set of nodes W, whose labels are the BASs of T. Our proof relies on the fact that the leaf $\top$ in B_T cannot be the *Low* child of a node, and analogously $\bot$ cannot be a *High* child. The intuition behind this is that visiting $Low(w)$ symbolises the act of excluding the node $Lab(w) \in$ BAS from an attack. Since static ATs are coherent, *excluding a BAS cannot make an attack succeed.* Therefore, taking the *Low* child of w cannot lead to $\top$; the reasoning for $\bot$ and *High* is analogous.

Theorem 3. *Let T be a static* AT*, B_T its* BDD *encoding over $\langle \text{BAS}, < \rangle$, α an attribution on V, and $D_\star = (V, \triangledown, \triangle, 1_\triangledown, 1_\triangle)$ an attribute domain with neutral elements resp. for $\triangledown$ and $\triangle$. Then $\breve{\alpha}(T) = \texttt{BDD}_{\texttt{DAG}}(B_T, R_{B_T}, \alpha, D_\star)$.*

BDDs to compute semantics: The BDD encoding of a static AT T can also be used to compute $[\![T]\!]$. Consider a path $\pi = a_1 \overline{a_2} \cdots a_\ell$ from the root of B_T to its $\top$-leaf: a_i (resp. $\overline{a_i}$) denotes that π goes to the *High* (resp. *Low*) child of the BDD node labelled with $a_i \in$ BAS. Then π represents a successful attack $A \doteq \{a_i \in \text{BAS} \mid a_i \text{ appears positive in } \pi\} \in \mathscr{A}_T$. To compute all successful attacks: 1) find all distinct paths $\{\pi_j\}_{j=1}^n$ in the graph of B_T, from its root node to its $\top$-leaf; 2) let $A_j = \{\text{positive BAS in } \pi_j\}$. Providing that B_T encodes minimal attacks only, the result is $\{A_j\}_{j=1}^n = [\![T]\!]$.

D. Computing the k-top metric values

The approach described above can be extended to efficiently compute the k-top values for a given metric. This problem asks not only the min/max value of the metrics from Table II, but also the next $k-1$ min/max values, e.g. the cost of the k cheapest attacks, or the probability of the k most likely ones.

Such k-top values can be computed by weighing the *High* edges of the BDD with their corresponding (source-) BAS attributes, and finding the k-shortest-weighted paths from the root of the BDD to its $\top$-leaf. We present this idea as Algo. 3.

Algo. 3 relies on an implementation of `shortest_paths`: the k-shortest-paths algorithm for DAGs. This is a well-known extension of the Dijkstra (or Thorup) algorithm [39, 40]. For a DAG G with edges weighted by the matrix Q, $\texttt{shortest\_paths}(G, Q, s, t, k, \circ)$ returns the weight of the k-shortest paths from a (source) node s of G, to a (target) node t, using operator $\circ$ to accumulate weight.

Algo. 3 works for $\triangledown \in \{\min, \max\}$, using a sign change to compute max-top values, in which case the implementation

Input: BDD $B_T = (W, Low, High, Lab)$,
number of values to compute $k \in \mathbb{N}$,
attribute domain $D = (V, \triangledown, \triangle)$,
attribution α.
Output: k-top metric values of T for α and D.

Q := 0-filled $|W| \times |W|$ matrix
if $\triangledown = \min$ **then** $sgn := 1$ **else** $sgn := -1$; // $\triangledown = \max$
foreach *nonterminal node* $w \in W_n$ **do**
  $Q[w][High(w)] := sgn * \alpha(Lab(w))$
return $sgn * \texttt{shortest\_paths}(B_T, Q, R_{B_T}, \top, k, \triangle)$

Algorithm 3: `k_top` metric values for a SAT T

of `shortest_paths` must support negative weights. The correctness of the algorithm is a direct consequence of the (correct) encoding of the minimal attacks of T by the BDD B_T, and the `shortest_paths` algorithm.

Example 8. Consider the DAG-structured SAT from Fig. 3, $T = \text{AND}\big(\text{OR}(a,b), \text{OR}(b,c)\big)$. To compute its 2 cheapest attacks under the attribution $\alpha = \{a \mapsto 3, b \mapsto 1, c \mapsto 4\}$, let $b < a < c$ s.t. B_T is as in Example 7. The *Low* edge of the root b (that encodes "not performing b") is labelled with cost 0, and the *High* edge with cost $\alpha(b) = 1$; the same is done for a and c. Then the shortest-weight path from the root of B_T to its 1-labelled leaf is $\pi_1 = b$, which yields the cheapest attack $A_1 = \{b\}$ with cost $\widehat{\alpha}(A_1) = \alpha(b) = 1$. Second to that we find the path $\pi_2 = \overline{b}ac$, which yields the second-cheapest attack $A_2 = \{a, c\}$ with cost $\widehat{\alpha}(A_2) = \alpha(a) \triangle \alpha(c) = 3 + 4 = 7$.

VI. Analysis of Dynamic Attack Trees

In the presence of SAND gates, the execution order of the BAS becomes relevant. This affects primarily the semantics, i.e. what it means to perform a successful attack, but also security metrics become sensitive to the sequentiality of events.

A. Partially-ordered attacks and well-formedness

As for the static case, the semantics of a dynamic attack tree (DAT) is defined by its successful attack scenarios. However, DATs necessitate a formal notion of order, because a sequential gate $\text{SAND}(v_1, \ldots, v_n)$ succeeds only if every v_i child is completely executed before v_{i+1} starts.

Such constructs model dependencies in the order of events. E.g. in Håstad's broadcast attack, n messages must first be intercepted, from which an n-th root (the secret key) may be computed. In this standard *ordered interpretation*, an activated BAS is uninterruptedly completed. This rules out constructs that introduce circular dependencies such as $\text{SAND}(a, b, a)$.[2]

Therefore, an attack scenario that operates with SAND gates is not just a set $A \subseteq$ BAS, but rather a partially-ordered set: a *poset* $\langle A, \prec \rangle$, where $a \prec b$ indicates that $a \in A$ must be carried out strictly before $b \in A$. Incomparable basic attack steps can be executed in any order, or in parallel.

[2] Cf. Kumar et al. (2015), who separates activation from execution of a BAS and can therefore operate with $\text{SAND}(a, b, a)$ [8].

Thus, the attack $\langle A, \prec \rangle$ indicates that all BAS in A must be executed, and their execution order will respect $\prec$. This succinct construct can represent combinatorially many execution orders of BAS. For instance $\langle \{a, b\}, \{(a, a), (b, b)\} \rangle$ allows three executions: the sequence (a, b), and (b, a), and the parallel execution $a \| b$. Instead, $\langle \{a, b\}, \{(a, a), (b, b), (a, b)\} \rangle$ only allows the execution sequence (a, b).

Partial orders are reflexive and transitive, so for instance $\mathrm{SAND}(a, b, c)$ gives rise to $\prec = \{(a, a), (b, b), (c, c), (a, b), (b, c), (a, c)\}$. We use an abbreviated notation that depicts their transitive reduction, so the previous case becomes $\{a \prec b, b \prec c\}$. This is a textual equivalent to the (unique) Hasse diagram that represents the poset.

Example 9. Consider the dynamic attack tree from Fig. 2b: $T_d = \mathrm{OR}\big(\mathrm{AND}(\mathit{ff}, w), \mathrm{SAND}(w, cc)\big)$. The posets $\langle \{w, cc\}, \{w \prec cc\} \rangle$ and $\langle \{\mathit{ff}, w\}, \varnothing \rangle$ are attack scenarios for T_d, where $\prec = \varnothing$ in the latter implies that ff and w can be executed in any order, even in parallel. So this poset represents (among others) the BAS execution sequence (w, ff), which results in a TLA of T_d. Similarly, the poset $\langle \{w, cc\}, \varnothing \rangle$ allows the execution sequence (cc, w): this violates the gate $\mathrm{SAND}(w, cc)$ so it cannot be considered a valid attack for T_d.

Since successful attacks $\langle A, \prec \rangle$ must ensure all the sequential orders imposed by SAND gates, it is possible to express infeasible requirements. For example, $\mathrm{SAND}(a, b, a)$ indicates that a must precede b, and b must precede a. To rule out these cases, we operate with well-formed DATs only. A DAT is well-formed if, for every $\mathrm{SAND}(v, v')$, all the BASs below v are executed before any of the BASs below v'.

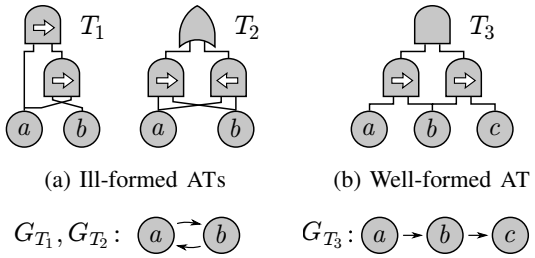
Definition 8 (Well-formedness)**.** The *BAS descendants* of a node $v \in N$ are $\mathrm{BAS}(v) = \{v\}$ if $t(v) = \texttt{BAS}$, and $\mathrm{BAS}(v) = \bigcup_{u \in ch(v)} \mathrm{BAS}(u)$ otherwise. The *ordering graph* of T is the directed graph $G_T = \big(\mathrm{BAS}_T, \xrightarrow{T}\big)$ s.t. $a \xrightarrow{T} b$ iff there is a SAND gate $v = \mathrm{SAND}(v_1, \ldots, v_n)$ with $a \in \mathrm{BAS}(v_i)$ and $b \in \mathrm{BAS}(v_{i+1})$ for some $0 < i < n$. T is *well-formed* if G_T is acyclic; otherwise T is *ill-formed*.

Example 10. Fig. 4a presents two ill-formed dynamic ATs: T_1 and T_2. In contrast, T_3 (Fig. 4b) and T_d (Fig. 2b) are examples of well-formed dynamic attack trees.

Our well-formedness criterion can rule out DATs for which successful sequential attacks do exist. In Fig. 4a, the execution sequence (a, b) makes the TLA of T_2 succeed. But T_2 is a modelling error under our ordered interpretation of SAND gates, because its subtree $\mathrm{SAND}(b, a)$ indicates that b must be completed to enable a. Nevertheless, such execution makes sense under an interpretation of the OR gate that allows the parallel execution of both children, and sees who finishes first. To cover these cases, future work can relax our assumptions.

B. Semantics for dynamic attack trees

The transitive reduction of the ordering graph G_T is a Hasse diagram, that represents the poset of all BAS nodes and SAND gates of T. This matches the notion of poset that has been



(a) Ill-formed ATs (b) Well-formed AT

(c) Ordering graphs, with transitive reduction of $\rightarrow$

Figure 4: Well-formedness of dynamic Attack Trees

intuitively introduced as an attack, and that we formalise in Def. 9. This definition also lifts the *successful* and *minimal attacks* of SATs to the category of posets. The resulting notion of dynamic attack, which underpins our DAT semantics, can thus be seen as an extension of the standard concepts for SATs, conservative w.r.t. our notion of well-formedness (Def. 8).

Interestingly, well-formedness plus the structure function of static ATs suffices to define successful attacks in DATs: these must (1) respect all SAND gates, and (2) be successful in the corresponding static AT, obtained by transforming SAND gates into ANDs. As a consequence, we need not introduce a new structure function for dynamic ATs:

Definition 9 (Attacks in dynamic ATs)**.** Let T be a well-formed DAT with ordering graph $G = (\mathrm{BAS}, \rightarrow)$:

- An *attack scenario*, or shortly an *attack*, of T is a poset $\langle A, \prec \rangle$ s.t. $A \subseteq \mathrm{BAS}$, and $\prec = \rightarrow\restriction_A$ restricts the edge relation $\rightarrow$ to A, i.e. $\forall a, b \in A.\, (a \prec b) \Leftrightarrow (a \rightarrow b)$;
- An attack $\langle A, \prec \rangle$ is *successful* if $f_{T'}(A) = 1$, where $f_{T'}$ is the structure function of the SAT T', which is obtained by replacing every SAND gate in T by an AND;
- A successful attack $\langle A, \prec \rangle$ is *minimal* if both A and $\prec$ are minimal, i.e. $\nexists$ successful $\langle A', \prec' \rangle.\, (A' \subsetneq A) \vee (\prec \subsetneq \prec')$.

Example 11. The ordering graph of the dynamic attack tree T_d from Fig. 2b has the single edge $w \rightarrow cc$. Therefore, three successful attacks for T_d are: $\langle \{w, cc\}, \{w \prec cc\} \rangle$, $\langle \{\mathit{ff}, w\}, \varnothing \rangle$, and $\langle \{\mathit{ff}, w, cc\}, \{w \prec cc\} \rangle$. The first two are minimal attacks. Instead, the attack $\langle \{\mathit{ff}, cc\}, \varnothing \rangle$ is not successful, and the poset $\langle \{w, cc\}, \{cc \prec w\} \rangle$ is not an attack since $(cc, w) \in \prec \setminus \rightarrow$, where $\setminus$ denotes set difference.

In minimising also over the partial order $\prec$, Def. 9 makes minimal attacks the least restrictive in terms of sequential dependencies. Moreover, an *attack suite* $\mathcal{S}$ of a dynamic AT T is a set of attacks, just like for SATs. Also $\mathscr{A}_T$ denotes the universe of attacks of T, and $\mathscr{S}_T$ its universe of attack suites.

Unlike for SATs, however, the execution order imposed by SAND gates makes dynamic ATs *non-coherent* in general. Consider $\mathrm{SAND}(a, \mathrm{OR}(b, c))$, where $\langle \{a, b\}, \{a \prec b\} \rangle$ is a successful attack but $\langle \{a, b, c\}, \{c \prec a, a \prec b\} \rangle$ is not, even though the latter extends the former with $c \in \mathrm{BAS}$.

Coherence is a desired property: it means that all successful attacks of a tree are characterised by its minimal attacks. To maintain this property in the presence of SAND gates, Def. 9 forces the partial order of an attack $\langle A, \prec \rangle$ to be a restriction

(to A) of the edge relation of the corresponding ordering graph. Posets that either *omit a required execution order* (e.g. the last one in Example 9), or *add an invalid execution order* (e.g. the last one in Example 11), are not attacks of T. This restriction in Def. 9 results in the coherence of DATs:

Proposition 1. *A well-formed dynamic* AT T *is coherent: if* $\langle A_1, \prec_1\rangle, \langle A_2, \prec_2\rangle \in \mathscr{A}_T$ *and* $\langle A_1, \prec_1\rangle$ *is a successful attack, then* $A_1 \subseteq A_2$ *implies that* $\langle A_2, \prec_2\rangle$ *is also a successful attack.*

Proof. Let $\langle A_1, \prec_1\rangle, \langle A_2, \prec_2\rangle \in \mathscr{A}_T$. By Def. 9, if $\langle A_1, \prec_1\rangle$ is a successful attack of T then $f_{T'}(A_1) = 1$, where T' is the static AT obtained by transforming all SAND gates of T to ANDs. Since SATs are coherent: $A_1 \subseteq A_2 \Rightarrow f_{T'}(A_2) = 1$. Finally by Defs. 8 and 9, $\prec_2 = \overset{T}{\rightarrow}\restriction_{A_2}$ implies that the sequences of execution of $A_2 \subseteq$ BAS represented by $\langle A_2, \prec_2\rangle$ respect the order imposed by the SAND gates of T. □

This means that, analogously to static ATs, the semantics of dynamic ATs can be given by their minimal attacks:

Definition 10. The *semantics of a well-formed* DAT T, denoted $[\![T]\!] \in \mathcal{S}_T$, is its suite of minimal attacks.

A price to pay for this result, and for such straightforward extensions of static concepts, is a strict notion of well-formed dynamic AT: besides discarding modeling errors such as T_2 in Fig. 4a, it also discards DATs where the children of a SAND gate share subtrees. To see this let $T = \mathrm{SAND}(v_1, v_2) = \mathrm{SAND}\big(\mathrm{AND}(a,b), \mathrm{AND}(b,c)\big)$, whose ordering graph G_T has edges from every descendant $\mathrm{BAS}(v_1) = \{a, b\}$ to every descendant $\mathrm{BAS}(v_2) = \{b, c\}$. But then G_T has a self-loop in the BAS $b \to b$, which means that T is ill-formed.

Our semantics also entails a strict notion of (successful) attack, that rules out some interleavings in the execution of high-level SAND gates. Consider e.g. $T' = \mathrm{SAND}\big(a, \mathrm{AND}(b,c)\big)$, where Def. 8 forces a to occur before any of $\{b, c\}$. Then $b \to a$ is *not an edge* in $G_{T'}$, so our attacks *exclude the order* $b \prec a$, even though (b, a, c) is a valid execution sequence in T'.

To relax this we need a more complex notion of ordering graph, as we discuss in Sec. IX. However, this work is about the efficient computation of metrics, and as we show next these metrics are invariant for the different valid orders of execution of BAS. Therefore, here we use the stricter but simpler semantics that stems from Def. 8 of well-formedness.

Finally, Lemma 2 characterises the semantics resulting from Defs. 8 to 10, analogously to how Lemma 1 does it for static ATs. This is key to prove the correctness of linear-time algorithms that compute metrics on tree-structured DATs.

Lemma 2. *Consider a well-formed* DAT *with nodes* $a \in$ BAS, $v_1, v_2 \in N$, *that has a proper tree structure. Then:*

1) $[\![a]\!] = \{\langle\{a\}, \varnothing\rangle\}$;
2) $[\![\mathrm{OR}(v_1, v_2)]\!] = [\![v_1]\!] \cup [\![v_2]\!]$;
3) $[\![\mathrm{AND}(v_1, v_2)]\!] = \{\langle A_1 \cup A_2, \prec_1 \cup \prec_2\rangle \mid \langle A_i, \prec_i\rangle \in [\![v_i]\!]\}$;
4) $[\![\mathrm{SAND}(v_1, v_2)]\!] = \{\langle A_1 \cup A_2\,,\ \prec_1 \cup \prec_2 \cup A_1 \times A_2\rangle \cdots$
$\cdots \mid \langle A_i, \prec_i\rangle \in [\![v_i]\!]\}$;
5) In cases 2)–4) above the $[\![v_i]\!]$ *are disjoint, and in cases 3) and 4) moreover the* A_i *are pairwise disjoint.*

We prove this lemma in [29]: just like for Lemma 1, we use induction in the structure of an AT whose root is the node v on the left-hand side $[\![v]\!]$ of the equalities. In particular, cases *1)*, *2)*, and *5)*, are trivial extensions of Lemma 1; case *3)* uses the minimality of the $\prec$ relation; and case *4)* moreover considers the order requirements imposed by the SAND gate on the BAS descendants of v_1 and v_2.

Comparison with literature: The semantics for dynamic ATs resulting from Defs. 8 to 10 resembles the so-called *series-parallel graphs* from [4]. We define dynamic attacks as posets for a number of reasons:

- they are a succinct, natural lifting of the SAT concepts, that facilitate the extension of earlier results such as the characterisation of $[\![\cdot]\!]$ in Lemma 2;
- metrics can be formally defined on this semantics, decoupling specific algorithms from a notion of correctness;
- in particular, this allows us to define algorithms to compute metrics regardless of the tree- or DAG-structure of the DAT.

The latter is different for [4], which does not work for DAG-structured DATs as noted in [18]. This can be illustrated in $T_3 = \mathrm{AND}\big(\mathrm{SAND}(a,b), \mathrm{SAND}(b,c)\big)$, the AT from Fig. 4b whose series-parallel graph is $SP_3 = (a \cdot b) \parallel (b \cdot c)$. Attributes and metrics are also defined in [4], choosing operators for AND and SAND gates which are resp. mapped to $\parallel$ and $\cdot$ in SP. Let the operator be $+$, e.g. to compute attack cost, and consider the attribution $\alpha = \{a \mapsto 1, b \mapsto 4, c \mapsto 8\}$: the metric obtained for SP_3 is $(1+4)+(4+8) = 17$. But the expected result is 13, i.e. execute every BAS once.

In contrast, posets entail a formal definition of metric over DAT semantics—given now in Sec. VI-C—which in particular yields the expected result even for DAG-structured DATs.

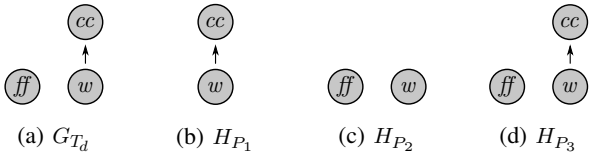
C. Security metrics for dynamic attack trees

The same fundamental concepts of metric for static ATs work for dynamic ATs: from the attributes of every BAS, obtain a metric for each attack in $[\![T]\!]$, and from these values compute the metric for T. Thus, the generic notion of metric given by Def. 5 in Sec. III-B carries on to this section.

However, attribute domains do not suffice for DATs: metrics such as min attack time are sensitive to order dependencies among BAS. This requires an additional *sequential operator* $\triangleright\colon V^2 \to V$, to compute values of sequential parts in an attack. Therefore, metric computations gain an extra step:

0) first, an attribution α assigns a value to each BAS;
1) then, a sequential metric $\vec{\alpha}$ uses the operator $\triangleright$ to assign a value to each sequential part of an attack;
2) then, a parallel metric $\widehat{\alpha}$ uses $\triangle$ to assign a value to each attack, as the parallel execution of all its sequential parts;
3) finally, the metric $\check{\alpha}$ uses $\triangledown$ to assign a value to the whole attack suite, by considering all its constituting attacks.

This can be pictured on the Hasse diagrams that represent the posets: for every attack $\langle A, \prec\rangle \in [\![T]\!]$, its (unique) Hasse diagram $H_A^{\prec}$ is the restriction of the ordering graph G_T to the nodes in $A \subseteq$ BAS—see e.g. Fig. 5 for T_d from Example 1. So $H_A^{\prec}$ is a set of nodes, some of which are connected by



(a) G_{T_d} (b) H_{P_1} (c) H_{P_2} (d) H_{P_3}

Figure 5: Ordering graph and Hasse diagrams of attacks of T_d: ordering graph G_{T_d}, attack $P_1 = \langle\{w, cc\}, \{w \prec cc\}\rangle$, attack $P_2 = \langle\{ff, w\}, \varnothing\rangle$, attack $P_3 = \langle\{ff, w, cc\}, \{w \prec cc\}\rangle$

edges and form a connected component C. In the 4-steps computation described above, this means that:

1) $\vec{\alpha}$ uses $\triangleright$ on each connected component $\{C_i\}_{i=1}^{n_A}$ of $H_A^{\prec}$, yielding one value $s_i \in V$ for each C_i;
2) $\widehat{\alpha}$ uses $\triangle$ on $\{s_i\}_{i=1}^{n_A}$, yielding a metric for the attack $H_A^{\prec}$;
3) $\breve{\alpha}$ uses $\triangledown$ on the metrics of all attacks in $[\![T]\!]$, yielding the metric for the dynamic attack tree T.

We now formalise these concepts, and write $\breve{\alpha}(T)$ for the unique value $\breve{\alpha}([\![T]\!])$ of the dynamic AT T, thus mapping Def. 11 to the generic notion of metric given in Def. 5.

Definition 11. Let $\triangledown, \triangle, \triangleright$ be three associative and commutative operators over a set V: we call $D = (V, \triangledown, \triangle, \triangleright)$ a *dynamic attribute domain*. Let T be a well-formed dynamic AT and α an attribution on V. The *metric for* T associated to D and α is given by:

$$\breve{\alpha}(T) = \underbrace{\bigtriangledown_{\langle A,\prec\rangle\in[\![T]\!]}}_{\breve{\alpha}} \underbrace{\bigtriangleup_{C\in H_A^{\prec}}}_{\widehat{\alpha}} \underbrace{\bigrhd_{a\in C}}_{\vec{\alpha}} \alpha(a)$$

where $H_A^{\prec}$ is the Hasse diagram of attack $\langle A, \prec\rangle$, and $a \in C$ ranges over the nodes of the connected component C of $H_A^{\prec}$.

Example 12. The semantics of the dynamic AT from Example 1 is $[\![T_d]\!] = \{\langle\{w, cc\}, \{w \prec cc\}\rangle, \langle\{ff, w\}, \varnothing\rangle\}$. The Hasse diagrams of these attacks—which resp. have one and two connected components—are shown in Figs. 5b and 5c. To compute the *min time* metric of T_d consider the attribution $\alpha = \{ff \mapsto 3, w \mapsto 15, cc \mapsto 1\}$ and the dynamic attribute domain $D = (\mathbb{N}, \min, \max, +)$. Then the time of the fastest attack for D and α is:

$$\begin{aligned}\breve{\alpha}(T_d) &= \bigtriangledown_{\langle A,\prec\rangle\in[\![T]\!]} \bigtriangleup_{C\in H_A^{\prec}} \bigrhd_{a\in C} \alpha(a)\\ &= \left(\bigtriangleup_{C\in H^{\varnothing}_{\{ff,w\}}} \bigrhd_{a\in C} \alpha(a)\right) \triangledown \left(\bigtriangleup_{C\in H^{w\prec cc}_{\{w,cc\}}} \bigrhd_{a\in C} \alpha(a)\right)\\ &= \big(\alpha(ff) \triangle \alpha(w)\big) \triangledown \big(\alpha(w) \rhd \alpha(cc)\big)\\ &= (3 \max 15) \min (15 + 1) \;=\; 15.\end{aligned}$$

In that computation, attack $\langle\{ff, w\}, \varnothing\rangle \equiv H^{\varnothing}_{\{ff,w\}}$ has two parallel steps: two connected components with one node each—see Fig. 5c—so operator $\triangle$ has two operands with one node each: $C = \{ff\}$ and $C' = \{w\}$. In contrast, $\langle\{w, cc\}, \{w \prec cc\}\rangle \equiv H^{w\prec cc}_{\{w,cc\}}$ has one connected component with two nodes—see Fig. 5b—so operator $\triangle$ has one operand but $\rhd$ has two: $\alpha(w)$ and $\alpha(cc)$. Finally, the *min time* of T_d is the $\triangledown = \min$ of these two metrics: the one for $\langle\{ff, w\}, \varnothing\rangle$.

Now consider the attributes $\alpha' = \{ff \mapsto 42, w \mapsto 10, cc \mapsto 0\}$ of *min skill* required for each BAS of T_d. Min skill is oblivious of sequential order: the skill needed to perform a task is independent of whether it must wait for the completion of other tasks. So, to compute the *min skill* metric of T_d we use the dynamic attribute domain $D' = (\mathbb{N}, \min, \max, \max)$, where the operators $\triangle$ and $\rhd$ are the same. This results in:

$$\begin{aligned}\breve{\alpha}'(T_d) &= \big(\alpha'(ff) \triangle' \alpha'(w)\big) \triangledown' \big(\alpha'(w) \rhd' \alpha'(cc)\big)\\ &= (42 \max 10) \min (10 \max 0) \;=\; 10.\end{aligned}$$

Example 13. Consider the DAG-structured dynamic AT from Fig. 4b, $T_3 = \mathrm{AND}\big(\mathrm{SAND}(a, b), \mathrm{SAND}(b, c)\big)$, whose ordering graph is $G_{T_3} = a \to b \to c$ which yields the semantics $[\![T_3]\!] = \{\langle\{a, b, c\}, \{a \prec b, b \prec c\}\rangle\}$. To compute the min attack cost let $\triangle = \rhd = +$ and $\alpha = \{a \mapsto 1, b \mapsto 4, c \mapsto 8\}$ as in the comparison with [4]. The Hasse diagram of the poset in $[\![T_3]\!]$ has one connected component with three nodes, so:

$$\begin{aligned}\breve{\alpha}(T_3) &= \bigtriangledown_{\langle A,\prec\rangle\in[\![T_3]\!]} \bigtriangleup_{C\in H_A^{\prec}} \bigrhd_{a\in C} \alpha(a)\\ &= \alpha(a) \rhd \alpha(b) \rhd \alpha(c) \;=\; 1 + 4 + 8 \;=\; 13.\end{aligned}$$

Many metrics are like min skill and cost in Examples 12 and 13: insensitive to the sequentiality of events. Therefore, reproducing Table II for dynamic ATs will introduce a third column for operator $\rhd$ which resembles the column for $\triangle$. A main relevant exception is min attack time, where $\rhd = +$ because each BAS in an order-dependency chain must wait for the completion of its predecessor, whereas $\triangle = \max$ yields the time of the slowest parallel part of the attack.

Note also that *the order of execution* of the BAS in the connected components of an attack *is irrelevant for the computation of a metric*. This is a direct consequence of the commutativity of the operator $\rhd$.

VII. Computations for tree-structured DATs

A precondition for our results is that the dynamic ATs are well-formed as per Def. 8. Algo. 4 checks this by building the edge relation $\to$ of the ordering graph $G = (\mathrm{BAS}, \to)$, and invoking a routine that checks whether G has directed cycles. Algo. 4 terminates after at most $O(n^2 m)$ steps (i.e. additions of pairs to $\to$), where $n = |\mathrm{BAS}|$ and m is the number of SAND gates. Ideally one would operate with the transitive reduction of $\to$, computable in less than $O(n^{2.5})$ [41].

Input: Dynamic attack tree $T = (N, t, ch)$.
Output: Whether T is a well-formed DAT.
edges := $\varnothing$
foreach $\mathrm{SAND}(v_1, \ldots, v_{n+1}) \in N$ **do**
  for $i = 1$ **to** n **do**
    edges := *edges* $\cup \big(\mathrm{BAS}(v_i) \times \mathrm{BAS}(v_{i+1})\big)$
return $\nexists$ *directed cycle in* $G = (\mathrm{BAS}, \mathit{edges})$

Algorithm 4: `is_well_formed`(T)

Earlier in Example 12, the computation of metrics for dynamic ATs was illustrated using Def. 11, which is worst-case exponential in the number of nodes. However and as for SATs, there is a linear bottom-up algorithm to compute metrics for tree-structured DATs. We present a recursive version in Algo. 5, and state its correctness in Theo. 4.

To prove the SAND case of the theorem, operator $\triangleright$ must distribute over $\triangledown$ and $\triangle$; the rest are trivial extensions—to attacks as posets—of the same cases from Theo. 1. In [29] we give the full proof of Theo. 4 by structural induction on T.

Thus and importantly, besides the tree-structure of the dynamic AT, the correctness of Algo. 5 requires the presence of three semiring algebraic structures: not only $(V, \triangledown, \triangle)$ as in the static case, but also $(V, \triangledown, \triangleright)$ and $(V, \triangle, \triangleright)$.

Definition 12. A *semiring dynamic attribute domain* is a dynamic attribute domain $D = (V, \triangledown, \triangle, \triangleright)$ where operator $\triangleright$ distributes over $\triangle$ and $\triangledown$, and also $\triangle$ distributes over $\triangledown$.

Input: Dynamic attack tree $T = (N, t, ch)$,
node $v \in N$,
attribution α,
semiring dynamic attr. dom. $D = (V, \triangledown, \triangle, \triangleright)$.
Output: Metric value $\breve{\alpha}(T) \in V$.

if $t(v) = \texttt{OR}$ **then**
  return $\bigtriangledown_{u \in ch(v)} \texttt{BU}_{\texttt{DAT}}(T, u, \alpha, D)$
else if $t(v) = \texttt{AND}$ **then**
  return $\bigtriangleup_{u \in ch(v)} \texttt{BU}_{\texttt{DAT}}(T, u, \alpha, D)$
else if $t(v) = \texttt{SAND}$ **then**
  return $\bigrhd_{u \in ch(v)} \texttt{BU}_{\texttt{DAT}}(T, u, \alpha, D)$
else // $t(v) = \texttt{BAS}$
  return $\alpha(v)$

Algorithm 5: $\texttt{BU}_{\texttt{DAT}}$ for a tree-structured DAT T

Theorem 4. *Let T be a well-formed tree-structured* DAT*, α an attribution on V, and $D = (V, \triangledown, \triangle, \triangleright)$ a semiring dynamic attribute domain. Then $\breve{\alpha}(T) = \texttt{BU}_{\texttt{DAT}}(T, R_T, \alpha, D)$.*

VIII. Computations for DAG-structured DATs

Algo. 5 does not work for dynamic ATs with a DAG-structure, for the same reasons exposed for static ATs in Sec. V. Neither is it possible to propose algorithms based on standard BDD theory: even though the structure function of SATs was reused in Def. 10, the computation of metrics for DATs intrinsically needs a notion of order among their BAS, that is not present in standard BDD-based data types.

As discussed in Sec. I, some earlier general approaches do exist to compute metrics on DAG-structured DATs [8, 5]. However, these often overshoot in terms of computation complexity. For static ATs and from a procedural (rather than semantic) angle, [18] proposes a more efficient, ingenious approach that computes and then corrects a metric value by traversing the AT bottom-up multiple times. It may be possible to extend this algorithm to consider SAND gates as well [28].

Alternatively, Def. 11 of metric for DATs could be encoded into a naïve algorithm. This would enumerate all posets from $[\![T]\!]$, and compute the metric value $\breve{\alpha}(T)$ by means of three nested loops that traverse all these Hasse diagrams. We do not expect such approach to be computationally efficient.

Instead and as in the static case, we expect that BDD encodings of the DAT offer better solutions. This requires BDD structures that are somehow sensitive to variable orderings. In that sense, the so-called sequential-BDDs recently presented for dynamic fault trees seem promising [42]. A first challenge would be to extend them to attributes other than failure (viz. attack) probability. Harder to tackle is the combinatorial explosion, that stems for the different possible orderings of BAS descendants of SAND gates.

In view of these considerations, we regard the algorithmic analysis for DAG-structured dynamic attack trees as an important open problem for future research.

IX. Conclusions

This paper presents algorithms to compute quantitative security metrics on attack trees. Our approach is formal: we classify AT models based on their structure and components, and then for each class we: (1) revise and consolidate its semantics in line with the literature, (2) define metrics generically on these semantics, (3) present algorithms to compute them, and (4) show the correctness of our algorithms, as well as their optimality in terms of computational complexity.

Algo. 2 is a prominent result: it computes metrics efficiently in DAG-structured static ATs, from a given semiring attribute domain with neutral elements. Another key contribution is the poset semantics defined for DATs in Sec. VI: it lifts the concepts used for SATs in a simple manner, which nevertheless allows computations on DAG-structured models.

We noted that our DAT semantics rules out some interleavings in the execution of SAND gates, e.g. (b, a, c) for $\text{SAND}\big(a, \text{AND}(b, c)\big)$. To allow such sequences it is necessary to use formulae—rather than individual BAS—as nodes of an ordering graph. For the DAT above, this would yield the edge $a \rightarrow (b \wedge c)$, which allows (b, a, c) because the formulae in that sequence are satisfied in the order "first a, then $b \wedge c$."

Interestingly, such formula-based ordering graphs preserve the coherence of our semantics, because Prop. 1 does not depend on the objects represented by the nodes: it just requires that traversing edges on the ordering graph represents valid execution orders of the children of SAND gates. Therefore, such ordering graphs are a promising research direction.

Further lines for future work also include: developing efficient algorithms to compute metrics on DAG-structured dynamic ATs; extending our AT syntax to include *sequential-*OR *gates* [8, 43]; extending our metrics to consider attacker profiles; and combining tree and DAG structures in a clever way, e.g. computing values linearly for the tree components, and plugging these into the rest of the (DAG) structure.

Related work

Surveys on attack trees are [44, 28]: the latter covers AT analysis via formal methods, from which we are close to

Metric	Static tree	Dynamic tree	Static DAG	Dynamic DAG
min cost	BU [14, 15, 16]	BU [4]	MTBDD [17] $\mathcal{C}$-BU [18]	PTA [8]
min time	BU [14, 19]	APH [9] BU [4]	Petri nets [12]	PTA [8]
min skill	BU [14, 20]	BU [4]	$\mathcal{C}$-BU [18]	—
max damage	BU [14, 19, 20]	BU [4]	MTBDD [17] DPLL [7]	PTA [8]
probability	BU [6, 19]	APH [9]	BDD [21] DPLL [7]	I/O-IMC [5]
Pareto fronts	BU [22, 19]	**OPEN PROBLEM**	$\mathcal{C}$-BU [11]	PTA [8]
Any of the above	**Algo. 1:** $\texttt{BU}_{\texttt{SAT}}$	**Algo. 5:** $\texttt{BU}_{\texttt{DAT}}$	**Algo. 2:** $\texttt{BDD}_{\texttt{DAG}}$	**OPEN PROBLEM**
***k*-top metrics**	BU-projection [14]	**OPEN PROBLEM**	**Algo. 3:** `BDD shortest_paths`	**OPEN PROBLEM**

Table III: Algorithms for metrics on different AT classes (replica of Table I)

BU: bottom-up on the AT structure. **APH**: acyclic phase-type (time distribution). **BDD**: binary decision diagram. **MTBDD**: multi-terminal BDD. **$\mathcal{C}$-BU**: repeated BU, identifying clones. **DPLL**: DPPL SAT-solving in the AT formula. **PTA**: priced time automata (semantics). **I/O-IMC**: input/output interactive Markov chains (semantics).

quantitative model checking—cf. simulation studies such as [12, 45]. Concrete case studies have been reported in [46].

Table III condenses literature references on quantitative analyses of ATs, classified by the structure and (dynamic) gates of the ATs where they are applicable. For each metric and AT class, in this table we cite the earliest relevant contributions that include concrete computation procedures.

Works [6, 7] are among the first to model and compute the cost and probability of attacks: their algorithms have EXPTIME complexity regardless of the AT structure. In [8, 10] an attack is moreover characterised by the time it takes. This allows for richer Pareto analyses but introduces one clock per BAS in the Priced Time Automata semantics: algorithms have thus EXPTIME & PSPACE complexity [47, 48]. The current work improves these bounds via specialised procedures tailored for the specific AT class, e.g. Algos. 1 and 5 resp. for tree-structured SATs and DATs have LINTIME complexity.

Indeed, all algorithms specialised on tree-structured ATs implement a bottom-up traversal on its syntactic structure: we denote these BU in Table III. Pareto analyses are polynomial, where the exponent is the number of parameters being optimised. Most works are on static ATs, with the relevant exception of [9, 4] which include sequential AND gates.

For DAG-structured static ATs the algorithmic spectrum is broader, owing to the NP-hardness of the problem (see Sec. V-A). Such algorithms range from classical BDD encodings for probabilities, and extensions to multi-terminal BDDs, to logic-based semantics that exploit DPLL, including an encoding of SATs as generalised stochastic Petri nets. A prominent contribution is [18, Alg. 1]: after computing so-called optional and necessary clones, its computations are exponential on the number of shared BAS (only).

In contrast, the computation of security metrics for dynamic attack trees is more recent than for SATs: here we find open problems in the literature, indicated in four cells of Table III. For tree-structured DATs and to the best of our knowledge, no work addresses directly the computation of Pareto frontiers. For this, our Algo. 5 ($\texttt{BU}_{\texttt{DAT}}$) could be embedded in the static setting of [22, 19]: the gist would be to carry around pairs of values instead of only one, removing dominated solutions at each step. As for k-top metric algorithms, our Algo. 5 could be extended with priority lists updated during the tree traversal.

We thus propose to tackle two open problems on tree-structured DATs, by simple combinations or extensions of other methods (from the literature or introduced in this work). In contrast, the open problems for DAG-structured DATs are less easy to overcome. To compute attack probability, [5] encodes these attack trees as a variant of Markov chains. For other metrics, [8] encodes the AT as a network of PTA and solves the resulting cost-optimal reachability problem. As earlier stated, these very powerful and general approaches are in detriment of computational efficiency.

A recent related approach encodes dynamic fault trees as so-called sequential-BDDs, to compute the probability of system failure [42]. Such safety-oriented works do not map directly to security analysis such as AT metrics, because: 1) they can compute probability—and possibly parallel time—only; 2) the dynamic gates are not the same than those in dynamic ATs; 3) the standard logical gates are interpreted differently. Still, it might be feasible to adapt [42] to compute AT metrics, e.g. to compare it against the algorithms here presented. Probably the main detriment is that sequential-BDDs expand sequence dependencies of every pair of events, adding a combinatorial blow-up on top of the already exponential explosion incurred by BDD representations of DAGs. This leads us to believe that even the EXPTIME complexity of our Algo. 2 should be more time-efficient.

References

[1] Y. Roudier and L. Apvrille, "SysML-Sec: A model driven approach for designing safe and secure systems," in *MODELSWARD*, pp. 655–664. IEEE, 2015. ISBN 978-989-758-136-6

[2] L. Apvrille and Y. Roudier, "SysML-sec: A sysML environment for the design and development of secure embedded systems," in *APCOSEC*, 2013. [Online]. Available: http://www.eurecom.fr/publication/4186

[3] Isograph, *AttackTree*. [Online]. Available: https://www.isograph.com/software/attacktree/

[4] R. Jhawar, B. Kordy, S. Mauw, S. Radomirović, and R. Trujillo-Rasua, "Attack Trees with Sequential Conjunction," in *SEC*, ser. IFIPAICT, vol. 455, pp. 339–353. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-18467-8_23

[5] F. Arnold, D. Guck, R. Kumar, and M. Stoelinga, "Sequential and Parallel Attack Tree Modelling," in *SAFECOMP*, ser. LNCS, vol. 9338, pp. 291–299. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-24249-1_25

[6] A. Buldas, P. Laud, J. Priisalu, M. Saarepera, and J. Willemson, "Rational choice of security measures via multi-parameter attack trees," in *CRITIS*, ser. LNCS, vol. 4347, pp. 235–248. Springer Berlin Heidelberg, 2006. DOI: 10.1007/11962977_19

[7] A. Jürgenson and J. Willemson, "Computing exact outcomes of multi-parameter attack trees," in *OTM*, ser. LNCS, vol. 5332, pp. 1036–1051. Springer Berlin Heidelberg, 2008. DOI: 10.1007/978-3-540-88873-4_8

[8] R. Kumar, E. Ruijters, and M. Stoelinga, "Quantitative Attack Tree Analysis via Priced Timed Automata," in *FORTE*, ser. LNCS, vol. 9268, pp. 156–171. Springer International Publishing, 2015. DOI: 10.1007%2F978-3-319-22975-1_11

[9] F. Arnold, H. Hermanns, R. Pulungan, and M. Stoelinga, "Time-dependent analysis of attacks," in *POST*, ser. LNCS, vol. 8414, pp. 285–305. Springer Berlin Heidelberg, 2014. DOI: 10.1007/978-3-642-54792-8_16

[10] R. Kumar, S. Schivo, E. Ruijters, B. Yildiz, D. Huistra, J. Brandt, A. Rensink, and M. Stoelinga, "Effective Analysis of Attack Trees: A model-driven approach," in *FASE*, ser. LNCS, vol. 10802, pp. 56–73. Springer, 2018. DOI: 10.1007/978-3-319-89363-1_4

[11] B. Fila and W. Wideł, "Efficient attack-defense tree analysis using Pareto attribute domains," in *CSF*, pp. 200–215, 2019. DOI: 10.1109/CSF.2019.00021

[12] Dalton, Mills, Colombi, and Raines, "Analyzing attack trees using generalized stochastic Petri nets," in *2006 IEEE Information Assurance Workshop*, pp. 116–123, 2006. DOI: 10.1109/IAW.2006.1652085

[13] M. Gribaudo, M. Iacono, and S. Marrone, "Exploiting Bayesian networks for the analysis of combined attack trees," *Electronic Notes in Theoretical Computer Science*, vol. 310, pp. 91–111, 2015. DOI: 10.1016/j.entcs.2014.12.014

[14] S. Mauw and M. Oostdijk, "Foundations of Attack Trees," in *ICISC*, ser. LNCS, vol. 3935, pp. 186–198. Springer Berlin Heidelberg, 2006. DOI: 10.1007/11734727_17

[15] J. Weiss, "A system security engineering process," in *Proceedings of the 14th National Computer Security Conference*, ser. Information System Security: Requirements & Practices, vol. 249, pp. 572–581, 1991.

[16] B. Schneier, "Attack trees," *Dr. Dobb's journal*, vol. 24, no. 12, pp. 21–29, 1999.

[17] A. Bobbio, L. Egidi, and R. Terruggia, "A methodology for qualitative/quantitative analysis of weighted attack trees," *IFAC Proceedings Volumes*, vol. 46, no. 22, pp. 133–138, 2013. DOI: 10.3182/20130904-3-UK-4041.00007

[18] B. Kordy and W. Wideł, "On quantitative analysis of attack–defense trees with repeated labels," in *POST*, ser. LNCS, vol. 10804, pp. 325–346. Springer International Publishing, 2018. DOI: 10.1007/978-3-319-89722-6_14

[19] O. Henniger, L. Apvrille, A. Fuchs, Y. Roudier, A. Ruddle, and B. Weyl, "Security requirements for automotive on-board networks," in *ITST*, pp. 641–646. IEEE, 2009. DOI: 10.1109/ITST.2009.5399279

[20] E. J. Byres, M. Franz, and D. Miller, "The use of attack trees in assessing vulnerabilities in SCADA systems," in *IISW*, pp. 3–10. IEEE, 2004.

[21] A. Rauzy, "New algorithms for fault trees analysis," *Reliability Engineering & System Safety*, vol. 40, no. 3, pp. 203–211, 1993. DOI: 10.1016/0951-8320(93)90060-C

[22] Z. Aslanyan and F. Nielson, "Pareto efficient solutions of attack-defence trees," in *POST*, ser. LNCS, vol. 9036, pp. 95–114. Springer Berlin Heidelberg, 2015. DOI: 10.1007/978-3-662-46666-7_6

[23] A. Bossuat and B. Kordy, "Evil twins: Handling repetitions in attack–defense trees," in *GraMSec*, ser. LNCS, vol. 10744, pp. 17–37. Springer International Publishing, 2018. DOI: 10.1007/978-3-319-74860-3_2

[24] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986. DOI: 10.1109/TC.1986.1676819

[25] W. Oortwijn, T. v. Dijk, and J. v. d. Pol, "Distributed binary decision diagrams for symbolic reachability," in *SPIN*, pp. 21–30. ACM, 2017. DOI: 10.1145/3092282.3092284

[26] Z. Aslanyan, F. Nielson, and D. Parker, "Quantitative verification and synthesis of attack-defence scenarios," in *CSF*, pp. 105–119. IEEE Computer Society, 2016. DOI: 10.1109/CSF.2016.15

[27] R. E. Barlow and F. Proschan, *Statistical theory of reliability and life testing: probability models*, ser. Intl. series in decision processes. Holt, Rinehart and Winston, 1975. ISBN 0030858534

[28] W. Wideł, M. Audinot, B. Fila, and S. Pinchinat, "Beyond 2014: Formal methods for attack tree–based security modeling," *ACM Comput. Surv.*, vol. 52, no. 4, 2019. DOI: 10.1145/3331524

[29] C. E. Budde and M. Stoelinga, "Efficient algorithms for quantitative attack tree analysis," *arXiv e-prints*, vol. abs/2105.07511, 2021. [Online]. Available: https://arxiv.org/abs/2105.07511

[30] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer, "Foundations of attack–defense trees," in *FAST*, ser. LNCS, vol. 6561, pp. 80–95. Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-642-19751-2_6

[31] S. MacLane, *Categories for the working mathematician*. Springer-Verlag New York, 1971. ISBN 0387900357

[32] J. Legriel, C. Le Guernic, S. Cotton, and O. Maler, "Approximating the Pareto front of multi-criteria optimization problems," in *TACAS*, ser. LNCS, vol. 6015, pp. 69–83. Springer Berlin Heidelberg, 2010. DOI: 10.1007/978-3-642-12002-2_6

[33] W. Lee, D. Grosh, F. Tillman, and C. Lie, "Fault tree analysis, methods, and applications: A review," *IEEE Transactions on Reliability*, vol. R-34, no. 3, pp. 194–203, 1985. DOI: 10.1109/TR.1985.5222114

[34] H. Hermanns and M. Siegle, "Bisimulation algorithms for stochastic process algebras and their BDD-based implementation," in *AMAST*, ser. LNCS, vol. 1601, pp. 244–264. Springer, 1999. DOI: 10.1007/3-540-48778-6_15

[35] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT Press, 2008.

[36] M. Z. Kwiatkowska and D. Parker, "Advances in probabilistic model checking," in *Software Safety and Security – Tools for Analysis and Verification*, ser. NATO Science for Peace and Security Series – D: Information and Communication Security. IOS Press, 2012, vol. 33, pp. 126–151.

[37] E. Ruijters and M. Stoelinga, "Fault Tree Analysis: A survey of the state-of-the-art in modeling, analysis and tools," *Computer Science Review*, vol. 15–16, pp. 29–62, 2015. DOI: 10.1016/j.cosrev.2015.03.001

[38] A. Rauzy and Y. Dutuit, "Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within Aralia," *Reliability Engineering & System Safety*, vol. 58, no. 2, pp. 127–144, 1997. DOI: 10.1016/S0951-8320(97)00034-3

[39] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959. DOI: 10.1007/BF01386390

[40] M. Thorup, "Undirected single-source shortest paths with positive integer weights in linear time," *Jour. ACM*, vol. 46, no. 3, pp. 362–394, 1999. DOI: 10.1145/316542.316548

[41] A. Aho, M. Garey, and J. Ullman, "The transitive reduction of a directed graph," *SIAM Journal on Computing*, vol. 1, pp. 131–137, 1972. DOI: 10.1137/0201008

[42] H. Yu and X. Wu, "A method for transformation from dynamic fault tree to binary decision diagram," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, pp. 1–15, 2020. DOI: 10.1177/1748006X20974187

[43] H. Hermanns, J. Krämer, J. Krčál, and M. Stoelinga, "The value of Attack-Defence Diagrams," in *POST*, ser. LNCS, vol. 9635, pp. 163–185. Springer Berlin Heidelberg, 2016. DOI: 10.1007/978-3-662-49635-0_9

[44] B. Kordy, L. Piètre-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. DOI: 10.1016/j.cosrev.2014.07.001

[45] Y. Wadhawan, A. AlMajali, and C. Neuman, "A comprehensive analysis of smart grid systems against cyber-physical attacks," *Electronics*, vol. 7, no. 10, 2018. DOI: 10.3390/electronics7100249

[46] M. Fraile, M. Ford, O. Gadyatskaya, R. Kumar, M. Stoelinga, and R. Trujillo-Rasua, "Using attack-defense trees to analyze threats and countermeasures in an ATM: A case study," in *PoEM*, ser. LNCS, vol. 267, pp. 326–334. Springer, 2016. DOI: 10.1007/978-3-319-48393-1_24

[47] R. Alur and D. L. Dill, "A theory of Timed Automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994. DOI: 10.1016/0304-3975(94)90010-8

[48] G. Behrmann, K. G. Larsen, and J. I. Rasmussen, "Priced Timed Automata: Algorithms and applications," in *FMCO*, ser. LNCS, vol. 3657, pp. 162–182. Springer Berlin Heidelberg, 2005. DOI: 10.1007/11561163_8