Semantics of Attack-Defense Trees for Dynamic Countermeasures and a New Hierarchy of Star-free Languages

Thomas Brihaye^{1[0000-0001-5763-3130]}, Sophie Pinchinat², and Alexandre Terefenko^{1,2}

¹ University of Mons, Belgium

² University of Rennes, France

Abstract. We present a mathematical setting for attack-defense trees, a classic graphical model to specify attacks and countermeasures. We equip attack-defense trees with (trace) language semantics allowing to have an original dynamic interpretation of countermeasures. Interestingly, the expressiveness of attack-defense trees coincides with star-free languages, and the nested countermeasures impact the expressiveness of attack-defense trees. With an adequate notion of countermeasure-depth, we exhibit a strict hierarchy of the star-free languages that does not coincides with the classic one. Additionally, driven by the use of attackdefense trees in practice, we address the decision problems of trace membership and of non-emptiness, and study their computational complexities parameterized by the countermeasure-depth.

1 Introduction

Security is nowadays a subject of increasing attention as means to protect critical information resources from disclosure, theft or damage. The informal model of *attack trees* is due to Bruce Schneier³ to graphically represent and reason about possible threats one may use to attack a system. Attack trees have then been widespread in the industry and are advocated since the 2008 NATO report to govern the evaluation of the threat in risk analysis. The attack tree model has attracted the interest of the academic community in order to develop their mathematical theory together with formal methods (see the survey [24]).

Originally in [20], the model of attack tree aimed at describing how an attack goal refines into subgoals, by using two operators OR and AND to coordinate those refinements. The subgoals are understood in a "static" manner in the sense that there is no notion of temporal precedence between them. Still, with this limited view, many analysis can be conducted (see for example [7,5]). Then, the academic community considered two extensions of attack trees. The first one, called *attack-defense tree* (adt, for short), is obtained by augmenting attack trees with nodes representing countermeasures [10,8]. The second one, initiated

³ https://www.schneier.com/academic/archives/1999/12/attack_trees.html

by [16,6], concerns a "dynamic" view of attacks with the ability to specify that the subgoals must be achieved in a given order. This way to coordinate the subgoals is commonly specified by using operator SAND (for Sequential AND). In [1], the authors proposed a *path semantics* for attack trees with respect to a given a transition system (a model of the real system). However, a unifying formal semantics amenable to the coexistence of both extensions of attack trees – namely with the defense and the dynamics – has not been investigated yet.

In this paper, we propose a formal language semantics of adts, in the spirit of the trace semantics by [2] (for defenseless attack trees), that allows countermeasure features via the new operator CO (for "countermeasure"). Interestingly, because in adts, countermeasures of countermeasures exist, we define the *countermeasure-depth* (maximum number of nested CO operators) and analyze its role in terms of expressiveness of the model.

First, we establish the Small Model Property for adds with countermeasuredepth bounded by one (Theorem 1), which ensure the existence of small traces in a non-empty semantics. This not so trivial result is a stepping stone to prove further results.

Second, since our model of adts is very close to star-free extended regular expressions (SEREs for short), that are star-free regular expressions extended with intersection and complementation, we provide a two-way translation from the former to the latter (Theorem 2). It is known that the class of languages denoted by SEREs coincides with the class of star-free languages [18], that can also be characterized as the class of languages definable in first-order logic over strings (FO[<]). We make explicit a translation from adts into FO[<] (Lemma 1) to shed light on the role played by the countermeasure-depth. Our translation is reminiscent of the constructions in [13] for an alternative proof of the result in [22] that relates the classic dot-depth hierarchy of star-free languages and the FO[<] quantifier alternation hierarchy. In particular, we show (Lemma 2) that any language definable by an adt with countermeasure-depth less than equal to k is definable in Σ_{k+1} , the (k+1)-th level of the first-order quantifier alternation hierarchy.

Starting from the proof used in [22] to show the strictness of the dot-depth hierarchy, we demonstrate that there exists an infinite family of languages whose definability by an adt requires arbitrarily large countermeasure-depths. It should be noticed that our notion of countermeasure-depth slightly differs from the complementation-depth considered in [21] for extended regular expressions⁴, because the new operator AND is rather a relative complementation. As a result, the countermeasure-depth of adts induces a new hierarchy of all star-free languages, that we call the *ADT-hierarchy*, that coincides (at least on the very first levels) neither with the dot-depth hierarchy, nor with the first-order logic quantifier alternation hierarchy.

Third, we study three natural decision problems for adts, namely the *membership problem* (ADT-MEMB), the *non-emptiness problem* (ADT-NE) and the equivalence problem (ADT-EQUIV). The problem ADT-MEMB is to determine if

⁴ arbitrary regular expressions extended with intersection and complementation.

a trace is in the semantics of an adt. From a practical security point of view, ADT-MEMB addresses the ability to recognize an attack, say, in a log file. The problem ADT-NE consists of, given an adt, deciding if its semantics is non-empty. Otherwise said, whether the information system can be attacked or not. Finally, the problem ADT-EQUIV consists of deciding whether two adts describe the same attacks or not. Our results are summarized in Table 1.

The paper is organized as follows. Section 2 proposes an introductory example. Next, we define our model of adds in Section 3, their trace semantics and countermeasure-depth, and present the Small Model Property for adds with countermeasure-depth bounded by one. We then show in Section 4 that adds coincide with star-free languages. We next study the novel hierarchy induced by the countermeasure-depth (Section 5) and study decision problems on adds (Section 6).

2 Introductory Example

Consider a thief (the proponent) who wants to steal two documents inside two different safes (Safe 1 and Safe 2), without being seen. The safes are located in two different but adjacent rooms (Room 1 and Room 2) in a building; the entrance/exit door of the building leads to Room 1. The rooms are separated by a door and each room has a window. Initially, the thief is outside of the building. A strategy for the proponent to steal the documents is to attempt to open Safe 1 until it succeeds, then open Safe 2 until it succeeds (and finally to exit the building). However, this strategy can be easily countered by the company, say by hiring a security guard visiting the rooms on some regular basis.

Security experts would commonly use an adt to describe how the proponent may achieve her goal and, at the same time, the ways its opponent (the company) may prevent the proponent from reaching her goal. An informal adt expressing the situation is given in Figure 1a, where traditionally goals of the proponent are represented in red circles, while countermeasures of the opponent are represented in green squares. An arrow from a left sibling to a right sibling specifies that the former goal must be achieved before starting the latter. A countermeasure targeting a proponent goal is represented with dashed lines.

As said, the graphical model of Figure 1a is informal and cannot be exploited by any automated tool for reasoning. With the setting proposed in this contribution, we make it formal, and in particular we work out a new binary operator CO for "countermeasure", graphically reflected with a curved dashed line between two siblings: the left sibling is a proponent's goal while the right sibling is the opponent's countermeasure – with this convention, we can unambiguously retrieve the player's type of an adt node. The more formal version of the adt in Figure 1a is drawn in Figure 1b (details for its construction can be found in Example 4).

The proposed semantics for adts also allows us to consider nested CO operators to express countermeasures of countermeasures. For example, a proponent countermeasure against the company countermeasure could be to be disguised



(a) A countermeasure from the Company. (b) Formal representation of the adt

Fig. 1: Adts for the thief and company problem.

as an employee working in the building; we formalise this situation in Example 4. It should be observed that nested countermeasure is a core aspect of our contribution and the main subject of Section 5.

3 Attack-Defense Trees and Countermeasure-Depth

Preliminary notations For the rest of this paper, we fix Prop a finite set of propositions and we assume that the reader is familiar with propositional logic. We use typical symbol γ for propositional formulas over Prop and write a valuation of the propositional variables a as an element of $\Sigma := 2^{Prop}$, that will be viewed as an alphabet. A *trace* t over Prop, is finite word over Σ , that is a finite sequence of valuations. We denote the empty trace by ε , and we define $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$. For a trace $t = a_1...a_n$, we define |t| := n, its the *length*, as the number of valuations appearing in t, and we let $t[i] := a_i$, for each $i \in \{1, ..., n\}$. We define the classic *concatenation* of traces: given two traces $t = a_1...a_n$ and $t' = a'_1...a'_m$, we define $t \cdot t' := a_1...a_na'_1...a'_m$. We also lift this operator to sets of traces in the usual way: given two sets of traces L and $L' \subseteq \Sigma^*$, we let $L \cdot L' := \{t \cdot t' : t \in L \text{ and } t' \in L'\}$. For a trace $t = a_1...a_n \in \Sigma^*$ and $1 \le i \le n$, the trace $t' = a_1...a_i$ is a *prefix* of t, written $t' \prec t$.

We define attack-defense trees (adts) over *Prop*, as well as their *trace semantics* and their *countermeasure-depth*, and develop enlightening examples. Adts are standard labeled finite trees with a dedicated set of labels based on the special ϵ label and propositional formulas for leaves and on the set {OR, SAND, AND, CO} for internal nodes.

Definition 1. The set ADT of adds over Prop is inductively defined by:

- the empty-word leaf ϵ and every propositional formula γ over Prop are in ADT;

- if trees $\tau_1, ..., \tau_n$ are in ADT, so are $OR(\tau_1, ..., \tau_n)$, $SAND(\tau_1, ..., \tau_n)$, and $AND(\tau_1, ..., \tau_n)$;
- if trees τ and τ' are in ADT, so is $CO(\tau, \tau')$.

The size of an adt τ , written $|\tau|$ is defined as the sum of the sizes of its leaves, provided the size of ϵ is 1, while the size of γ is its size when seen as a propositional formula.

Regarding the semantics, adts describe a set of traces over alphabet $\Sigma := 2^{Prop}$, hence their trace semantics. Formally, for an adt τ , we define the language $Traces(\tau) \subseteq \Sigma^*$. First, we set $Traces(\epsilon) = \{\epsilon\}$. Now, a leaf adt hosting formula γ denotes the reachability goal γ , that is the set of traces ending in a valuation satisfying γ (we use the classic notations $\top = p \lor \neg p$ and $\bot = \neg \top$ with $p \in Prop$). We make our trace semantics compositional by providing the semantics of the four operators OR, SAND, CO, and AND in terms of how the subgoals described by their arguments interact. Operator OR tells that at least one of the subgoals has to be achieved. Operator SAND requires that all the subgoals need being achieved in the left-to-right order. The binary operator CO requires to achieve the first subgoal without achieving the second one. Finally, operator AND tells that all subgoals need to be achieved, regardless of the order. Without any countermeasure, AND can be seen as a relaxation of the SAND, but it is not true in general (see example 3).

At the level of the property described by an adt, i.e. a trace language, the operators correspond to specific language operations: OR corresponds to union, SAND to concatenation, CO to a relativized complementation. Only AND corresponds to a less classic operation: a trace t belongs to the language of $AND(\tau_1, \tau_2)$ if t belongs to the language of τ_1 and has a prefix in the language of τ_2 or vice-versa. Formally:

Definition 2. Let L_1 , L_2 be two languages over alphabet Σ . The each of L_1 and L_2 is the language $L_1 \sqcap L_2 := (L_1 \cdot \Sigma^* \cap L_2) \cup (L_2 \cdot \Sigma^* \cap L_1)$.

Because operator \sqcap is associative, we can define $L_1 \sqcap L_2 \sqcap \ldots \sqcap L_n$ that amounts to being equal to $\bigcup_{i \in \{1,\ldots,n\}} (L_i \cap \bigcap_{j \neq i} L_j \cdot \Sigma^*)$.

Example 1. A word $w = a_1 a_2 \dots a_m$ belongs to $L_1 \sqcap L_2 \sqcap L_3$ whenever there are three (possibly equal) positions $i_1, i_2, i_3 = m$ such that, for each $j \in \{1, 2, 3\}$, the word prefix $a_1 \dots a_{i_j} \in L_{\pi(j)}$, for some permutation π of (1, 2, 3).

We can now formally define the adt semantics.

Definition 3.

- $Traces(\epsilon) := \{\varepsilon\} \text{ and } Traces(\gamma) := \{a_1...a_n \in \Sigma^* : a_n \models \gamma\};$ In particular, $Traces(\top) = \Sigma^+ \text{ and } Traces(\bot) = \emptyset;$
- $Traces(OR(\tau_1, ..., \tau_n)) := Traces(\tau_1) \cup ... \cup Traces(\tau_n);$
- $Traces(SAND(\tau_1, ..., \tau_n)) := Traces(\tau_1) \cdot ... \cdot Traces(\tau_n);$
- $Traces(co(\tau_1, \tau_2)) := Traces(\tau_1) \setminus Traces(\tau_2);$

 $- Traces(AND(\tau_1, ..., \tau_n)) = Traces(\tau_1) \sqcap ... \sqcap Traces(\tau_n).$

In the rest of the paper, we say for short that an adt is non-empty, written $\tau \neq \emptyset$, whenever $Traces(\tau) \neq \emptyset$. We say that two adts τ and τ' are equivalent, whenever $Traces(\tau) = Traces(\tau')$.

Remark 1. Since all operators \cup , \cdot and \sqcap over trace languages are associative, the trees of the form $\mathsf{OP}(\tau_1, \mathsf{OP}(\tau_2, \tau_3))$, $\mathsf{OP}(\mathsf{OP}(\tau_1, \tau_2), \tau_3)$, and $\mathsf{OP}(\tau_1, \tau_2, \tau_3)$ are all equivalent, when OP ranges over {OR, SAND, AND}. As a consequence, we may sometime assume that nodes with such operators are binary.

We now introduce some notations for particular adds to ease our exposition and provide some examples of adds with their corresponding trace property.

We define a family of adts of the form $|\geq \ell|, |<\ell|, |=\ell|$, where ℓ is a nonzero natural. We let $|\geq \ell| := \text{SAND}(\top, ..., \top)$ where \top occurs ℓ times; $|<\ell| := \text{CO}(\top, |\geq \ell|)$; and $|=\ell| := \text{CO}(|\geq \ell|, |\geq \ell + 1|)$. It is easy to establish that adt $|\geq \ell|$ (resp. $|<\ell|, |=\ell|$) denotes the set of traces of length at least (resp. at most, exactly) ℓ . We also consider particular adts and constructs for them.

- ALL := $OR(\epsilon, \top)$,
- NOT (τ) := CO(ALL, τ), and INTER (τ_1, τ_2) := NOT $(OR(NOT(\tau_1), NOT(\tau_2)))$,
- $]\tau [:= \text{SAND}(\text{ALL}, \tau, \text{ALL}),]\tau := \text{SAND}(\text{ALL}, \tau) \text{ and } \tau [:= \text{SAND}(\tau, \text{ALL}).$
- Given a formula γ over *Prop*, we let $\gamma := \operatorname{co}(\gamma, |\geq 2|)$.

Based on these notations, we develop further examples.

Example 2. $- Traces(ALL) = \Sigma^*;$

- $Traces(NOT(\tau)) = \Sigma^* \setminus Traces(\tau);$
- $Traces(INTER(\tau_1, \tau_2)) = Traces(\tau_1) \cap Traces(\tau_2);$
- $Traces(\underline{\gamma})$ is set of one-length traces whose unique valuation satisfies γ ; in particular, when a valuation a is understood as a formula, namely formula $\bigwedge_{p \in a} p \land \bigwedge_{p \notin a} \neg p$, the adt $\underline{a} := \operatorname{co}(a, |\geq 2|)$ is such that $Traces(\underline{a}) = \{a\}$;
- For a valuation a,
 - $Traces(\underline{a}) = Traces(\underline{a}) = Traces(\underline{a}) = \Sigma^* a \Sigma^*;$
 - $Traces(]\underline{a}) = Traces(]\underline{a}) = Traces(\underline{a}) = \Sigma^* a;$
 - also, $Traces(\underline{a}[) = a\Sigma^*$.

Example 3. Note that AND cannot be seen as a kind of relaxation of SAND. For the set of propositions $\{p\}$, if we consider the formula p as a leaf, $Traces(\underline{p}) = \{p\}$ and $Traces(\underline{\neg}p) = \{\emptyset\}$. Thus $SAND(\underline{\neg}p, \underline{p}) = \{t\}$ with trace $t = \emptyset p$. However $AND(\underline{\neg}p, \underline{p}) = \emptyset$. Let us notice that the construction of this example uses the CO operator (hidden in p and $\neg p$).

Example 4. We come back to the situation of our introductory example (Section 2). First, we discuss the formal semantics of the informal tree in fig. 1a. To do so, we propose the following set of propositions: $Prop = \{E, S_1, S_2, G\}$ where E holds when the thief is entering the building, S_1 (resp. S_2) holds when the first

(resp. second) safe is open, and G is true if a guard is in the building. The situation can be described by the following adt: $\tau_{ex_1} = \text{SAND}(E, \text{CO}(S_1 \land S_2, G[))$, represented in Figure 1b, where we distinguish SAND with a curved line and CO with a dashed line. We have $Traces(\tau_{ex_1}) = \{a_1...a_n \in \Sigma^* : a_n \models E\} \cdot (\{a_1...a_n \in \Sigma^* : a_n \models S_1 \land S_2\} \setminus \{a_1...a_n \in \Sigma^* : \exists i \text{ such that } a_i \models G\})$. If we write $\gamma_{\varphi} = \{a \in 2^{Prop} : a \models \varphi\}$, we have $Traces(\tau_{ex_1}) = \Sigma^* \gamma_E \cdot (\Sigma^* \gamma_{S_1 \land S_2} \setminus \Sigma^* \gamma_G \Sigma^*)$. In other words, we want all traces where E holds at some point and, after it, G cannot be true and finish by a valuation where $S_1 \land S_2$ holds.

In order to illustrate the nesting of countermeasures, we now allow the thief to disguise himself as an employee (assuming that when disguised, the guard does not identify him as a thief). To do so, we extended the set of propositions: $Prop' = \{E, S_1, S_2, G, D\}$, where D holds when the thief is disguised. The situation is now described by the following adt: $\tau_{ex_2} = \text{SAND}(E, \text{CO}(S_1 \land S_2, |\text{CO}(G, D)|))$. The semantics for τ_{ex_2} is all traces where E holds at some point and, after it, G cannot be true, except if D holds at the same time, and finish by a valuation where $S_1 \land S_2$ holds. A representation of τ_{ex_2} can be found in Appendix A

We now stratify the set ADT of adds according to their *countermeasure-depth* that denotes the maximum number of nested countermeasures.

Definition 4. The countermeasure-depth of an adt τ , written $\delta(\tau)$, is inductively defined by:

 $\begin{array}{l} - \ \delta(\epsilon) := \delta(\gamma) = 0; \\ - \ \delta(\mathcal{OP}(\tau_1, ..., \tau_n)) := \max\{\delta(\tau_1), ..., \delta(\tau_n)\} \ for \ every \ \mathcal{OP} \in \{\text{OR}, \text{SAND}, \text{AND}\}; \\ - \ \delta(\text{CO}(\tau_1, \tau_2)) := \max\{\delta(\tau_1), \delta(\tau_2) + 1\} \end{array}$

We let $ADT_k := \{\tau \in ADT : \delta(\tau) \leq k\}$ be the set of adds with countermeasuredepth at most k. Clearly $ADT_0 \subseteq ADT_1 \subseteq \ldots ADT_k \subseteq ADT_{k+1} \subseteq \ldots$, and $ADT = \bigcup_{k \in \mathbb{N}} ADT_k$.

Example 5. We list a couple of examples. $\delta(\text{NOT}(\tau)) = 1 + \delta(\tau); \delta(\text{INTER}(\tau_1, \tau_2)) = 2 + \max\{\delta(\tau_1), \delta(\tau_2)\}; \delta(\text{CO}(\text{CO}(\gamma_1, \gamma_2), \gamma_3)) = 1 \text{ while } \delta(\text{CO}(\gamma_3, \text{CO}(\gamma_2, \gamma_3))) = 2; \delta(|<\ell|) = \delta(|=\ell|) = 1 \text{ while } \delta(|\geq\ell|) = 0; \delta(\underline{\gamma}) = 1; \text{ In Example 4, } \delta(\tau_{ex_1}) = 1 \text{ and } \delta(\tau_{ex_2}) = 2.$

Also, $\delta(|\geq \ell|) = 0$, $\delta(|<\ell|) = \delta(|=\ell|) = 1$, so that $|\geq \ell| \in ADT_0$; $|<\ell|$ and $|=\ell| \in ADT_1$. Moreover, $\tau_{ex_1} \in ADT_1$ and $\tau_{ex_2} \in ADT_2$.

We say that a language L is ADT_k -definable (resp. ADT-definable), written $L \in ADT_k$ (resp. ADT), whenever $Traces(\tau) = L$, for some $\tau \in ADT_k$ (resp. for some k).

It can be established that non-empty adds in ADT_1 enjoy small traces, i.e. smaller than the size of the tree.

Theorem 1 (Small model property for ADT_1). An adt $\tau \in ADT_1$ is non-empty if, and only if, there is a trace $t \in Traces(\tau)$ with $|t| \leq |\tau|$.

We here only sketch the proof, whose details can be found in Appendix B. The technique we employ consists in defining a slight variant of the classic relation of super-word in language theory, that we call the *lift* binary relation. We prove that if $\tau \in ADT_1$, then there exists a finite set of generators, denoted $gen(\tau)$, which is sufficient to describe $Traces(\tau)$ through the lift relation. Next, we can prove that the traces in $gen(\tau)$ have size bounded by the number of leaves of τ . Notice that the result also holds for ADT_0 , since $ADT_0 \subseteq ADT_1$.

4 Adts, Star-free Languages, and First-Order Logic

We prove that adds coincide with star-free languages and first order formulas.

4.1 Reminders on Star-free Languages and First-Order Logic

The class of star-free languages introduced by [11,4,15] (over alphabet Σ) is obtained from the finite languages (or alternatively languages consisting of a single one-length word in Σ) by finitely many applications of Boolean operations (\cup , \cap and \sim for the complement) and the concatenation product (see [18, Chapter 7]). Alternatively, one characterizes star-free languages by first considering extended regular expressions – that are regular expressions augmented with intersection and complementation, and second by restricting to star-free extended regular expressions (SERES, for short) that are extended regular expressions with no Kleene-star operator. Regarding computational complexity aspects, we recall the following the subclass of SERES of extended regular expressions. The word membership problem (i.e., whether a given word belongs to the language denoted by a SERE) is in PTIME [9, Theorem 2], while the non-emptiness problem (i.e., is the denoted language empty?) and the equivalence problem (i.e., do two SEREs denote the same language?) are hard, both non-elementary [21, p. 162].

We now recall classical results on the first-order logic on finite words FO[<](see details in [14, Chapter 29]). The signature of FO[<], say for words over an alphabet Σ , is composed of a unary predicate a(x) for each $a \in \Sigma$, whose meaning is the "letter at position x of the word is a", and the binary predicate x < y that states "position x is strictly before position y in the word". For a FO[<]-formula, we define its size $|\psi|$ as the size of the expression considered as a word. A language L is FO[<]-definable whenever there exists a FO[<]-formula ψ such that a word $w \in L$ if, and only if, w is a model of ψ . Similarly, we say that an adt τ is FO[<]-definable if $Traces(\tau)$ is FO[<]-definable. It is well-known that FO[<]-definable languages coincide with star-free languages [11,22,13].

Also, for a fine-grained inspection of FO[<], let us denote by Σ_{ℓ} (resp. Π_{ℓ}) the fragments of FO[<] consisting of formulas with at most ℓ alternation of \exists and \forall quantifier blocks, starting with \exists (resp. \forall). The folklore results regarding satisfiability of FO[<]-formulas [21,12] are: (a) The satisfiability problem for FO[<] is non-elementary; (b) The satisfiability for Σ_{ℓ} is in $(\ell - 1)$ -EXPSPACE.⁵

⁵ with the convention that 0-EXPSPACE = PSPACE.

We are not aware of any result that establishes a tight lower bound complexity for the satisfiability problem on the Σ_{ℓ} fragments of FO[<].

We lastly recall the definition of the *dot-depth hierarchy* of star-free languages: level 0 of this hierarchy is $B_0 := \{L \subseteq 2^{\Sigma} : L \text{ is finite or co-finite}\}$, and evel ℓ is $B_\ell := \{L \subseteq 2^{\Sigma} : L \text{ is a Boolean combination of languages of the$ $form <math>L_1 \cdot \ldots \cdot L_n$ where $L_1, \ldots, L_n \in B_{\ell-1}\}$. The dot-depth hierarchy has a tight connection with FO[<] fragments [22]: for every $\ell > 0$, $\Sigma_\ell \subseteq B_\ell \subseteq \Sigma_{\ell+1}$.

4.2 Expressiveness of Adts

The first result of this section consist in showing that adds and star-free extended regular expressions share the same expressiveness.

Theorem 2. A language L is star-free if, and only if, L is ADT-definable.

For the "only if" direction of Theorem 2, we reason by induction on the class of star-free languages. For a language of the form $\{a\}$ where $a \in \Sigma$ one can take the adt <u>a</u> (that is $\operatorname{CO}(a, |\geq 2|)$). Now we can inductively build adequate adts for compound star-free languages by noticing that language operations of union and concatenation are captured by adts operators OR and SAND respectively, while complementation and intersection are obtained from the NOT(.) and INTER(.,.) as formalized in Example 2. One easily verifies that that the size of the adt corresponding to an SERE E is in O(|E|), where |E| denotes the size (number of characters) of E.

For the "if" direction of Theorem 2, it is easy to translate an adt into an SERE: the leaf ϵ translates into ϵ , a leaf adt γ translates into $\bigcup_{a \models \gamma} a$ – notice that this translation is exponential. For non-leaf adts, since every operator occurring in the adt has its language-theoretic counterpart the translation goes smoothly. However, the translation is exponential because of the adt operator AND, see Definition 2.

We now dig into the ADT-hierarchy induced by the countermeasure-depth and compare it with the FO[<] fragments Σ_{ℓ} and Π_{ℓ} .

We first step design a translation from ADT into FO[<], inductively over adts. The translation of an adt τ is written ψ_{τ} . For the base cases of adts ϵ and γ , and we let: $\psi_{\epsilon} := \forall x \perp$ and $\psi_{\gamma} := \exists x (\forall y \neg (x < y) \land \bigvee_{a \models \gamma} a(x)).$

Now, regarding compound adts, and not surprisingly, operator OR is reflected by the logical disjunction: $\psi_{OR(\tau_1,\tau_2)} := \psi_{\tau_1} \lor \psi_{\tau_2}$, while operator CO is reflected by means of the logical conjunction with the negated second argument: $\psi_{CO(\tau_1,\tau_2)} := \psi_{\tau_1} \land \neg \psi_{\tau_2}$. On the contrary, the two remaining operators SAND and AND require to split the trace into pieces, which can be captured by the folklore operation of *left (resp. right) position relativizations* of FO[<]-formulas w.r.t. a position [13, Proposition 2.1] (see also formulas of the form $\phi^{[x,y]}$ in [19]). Formally, given a position x in the trace t and an FO[<]-formula ψ , we define formula $\psi^{\leq x}$ (resp. $\psi^{>x}$) that holds of t if the prefix (resp. suffix) of t up to (resp. from) position x satisfies ψ , as follows. For $\bowtie \in \{\leq, >\}$, we let:

$$\begin{array}{ll} a(y)^{\bowtie x} = a(y) & (\psi \lor \psi')^{\bowtie x} = \psi^{\bowtie x} \lor \psi'^{\bowtie x} \\ (y < z)^{\bowtie x} = (y < z) & (\neg \psi)^{\bowtie x} = \neg \psi^{\bowtie x} \\ p(y)^{\bowtie x} = p(y) & (\exists y \psi)^{\bowtie x} = \exists y \ (y \bowtie x \land \psi^{\bowtie x}) \end{array}$$

Additionally, we write $\psi^{\leq 0}$ and $\psi^{>0}$ as the formulas obtained from $\psi^{\leq x}$ and $\psi^{>x}$ by replacing every occurrence of expressions $y \leq x$ and y > x by \perp and \top respectively.

Remark 2. For every formula $\psi \in \Sigma_{\ell}$, we also have $\psi^{\leq x}, \psi^{>x} \in \Sigma_{\ell}$.

We can now complete the translation from ADT into FO[<] by letting (w.l.o.g., by Remark 1, we can consider binary SAND and AND):

$$\begin{split} \psi_{\text{SAND}(\tau_1,\tau_2)} &:= \exists x [\psi_{\tau_1} \le x \land \psi_{\tau_2} > x] \lor (\psi_{\tau_1} \le 0 \land \psi_{\tau_2}) \\ \psi_{\text{AND}(\tau_1,\tau_2)} &:= \exists x [(\psi_{\tau_1} \le x \land \psi_{\tau_2}) \lor (\psi_{\tau_2} \le x \land \psi_{\tau_1})] \lor (\psi_{\tau_1} \le 0 \land \psi_{\tau_2}) \lor (\psi_{\tau_2} \le 0 \land \psi_{\tau_1}). \end{split}$$

Lemma 1. – For any $t \in \Sigma^*$, $t \models \psi_{\tau}$ iff $t \in Traces(\tau)$. – For any adt τ , formula ψ_{τ} is of size exponential in $|\tau|$.

In the rest of the paper, we use mere inclusion symbol \subseteq between subclasses of ADT and subclasses of FO[<], with the canonical meaning regarding the denoted trace languages.

An accurate inspection of the translation $\tau \mapsto \psi_{\tau}$ entails that every ADT_k definable adt can be equivalently represented by a Σ_{k+2} -formula, namely $ADT_k \subseteq \Sigma_{k+2}$. However, we significantly refine this expressiveness upperbound for ADT_k .

Lemma 2. 1. $ADT_0 \subseteq \Sigma_2 \cap \Pi_2$ – with an effective translation. 2. For every k > 0, $ADT_k \subseteq \Sigma_{k+1}$ – with an effective translation.

Regarding Item 1 of Lemma 2, it can be observed that, whenever $\tau \in ADT_0$, the quantifiers \forall and \exists commute in ψ_{τ} (see Appendix C). Now, for Item 2, the proof is conducted by induction over k (see Appendix C). We sketch here the case k > 1. First, remark that if $\tau_1, ..., \tau_n \in ADT_{k-1}$ are Σ_k -definable, then $OR(\tau_1, ..., \tau_n)$, SAND $(\tau_1, ..., \tau_n)$ and AND $(\tau_1, ..., \tau_n)$ remain Σ_k -definable, as formulas are obtained from conjunctions or disjunctions of Σ_k -definable formulas. Moreover, if τ_1 and τ_2 are Σ_k -definable, then $\tau = CO(\tau_1, \tau_2)$ is Σ_{k+1} -definable as a formula can be obtained from a boolan combination of two Σ_k -definable formulas. Finally, with k still fixed, it can be shown by induction over the size of an adt τ that, if $\tau \in ADT_k$, since all its countermeasures operators are of the form $CO(\tau_1, \tau_2)$ where $\tau_1 \in ADT_k$ and $\tau_2 \in ADT_{k-1}$, we have that adt τ is also Σ_{k+1} -definable, which concludes.

We can also establish lowerbounds in the ADT-hierarchy.

Lemma 3. 1. $B_{\ell} \subseteq ADT_{2\ell+2}$, and therefore $\Sigma_{\ell} \subseteq ADT_{2\ell+2}$ 2. $\Sigma_1 \subseteq ADT_0$ – with an effective translation. Item 1 of Lemma 3 is obtained by an induction of ℓ . Regarding Item 2, the translation consists in putting the main quantifier-free subformula of a Σ_1 formula in disjunctive normal form, and to focus for each conjunct on the set of "ordering" literals of the form x < y or $\neg(x < y)$ (leaving aside the other literals of the form a(x) or $\neg a(x)$ for a while). Each ordering literal naturally induces a partial order between the variables. We expend this partial order constraint over the variables as a disjunction of all its possible linearizations. For example, the conjunct $x < y \land \neg(y < z)$ is expended as the equivalent formula $(x < y \land y =$ $z) \lor (x < z \land z < y) \lor (z = x \land x < y) \lor (z < x \land x < y)$. Now, each disjunct of this new formula, together with the constraints a(x) (or $\neg a(x)$), can easily be specified by a SAND-rooted adt (ie. the root is a SAND). The initial Σ_1 -formula then is associated with the OR-rooted tree that gathers all the aforementioned SAND-rooted subtrees (see Remark 3 in Appendix B). Notice that the translation may induce at least an exponential blow-up.

The reciprocal of Item 2 in Lemma 3 is an open question. Still, we have little hope that it holds because ψ_{γ} seems to require a property expressible in $\Sigma_2 \setminus \Sigma_1$.

5 Strictness of the ADT-Hierarchy

One can notice that the adt $|\langle 2| \in ADT_1$ defines the finite language of traces of length at most 1, while it can be established that languages arising from adts in ADT_0 are necessarily infinite – if inhabited by a non-empty word (see Lemma 4 of Appendix B). Thus, we can easily deduce $ADT_0 \subsetneq ADT_1$. This section aims at showing that the entire ADT-hierarchy is strict:

Proposition 1. For every $k \in \mathbb{N}$, $ADT_k \subsetneq ADT_{k+1}$, even if $Prop = \{p\}$.

To show that $ADT_k \subseteq ADT_{k+1}$, we use a family of languages, originally introduced in [23], that we write $\{W_k\}_{k\in\mathbb{N}}$ over the two-letter alphabet Σ obtained from $Prop = \{p\}$. For readability, we use symbol a (resp. b) for the valuation $\{p\}$ (resp. \emptyset) of Σ . Formally, we define $W_k \subseteq \Sigma^*$ as follows – where ||w|| denotes the number of occurrences of a minus the number of occurrences of b in the word w:

We let $w \in W_k$ whenever all the following holds.

- $\|w\| = 0;$
- for every $w' \preceq w, 0 \le ||w'|| \le k;$
- there exists $w'' \preceq w$ s.t. ||w''|| = k.

In [23, Theorem 2.1], it is shown that $W_k \in B_k \setminus B_{k-1}$, for all $k \ge 1$.

We now determine the position of W_k languages in the ADT-hierarchy. We show Proposition 2.

Proposition 2. For each k > 0, $W_k \in ADT_{k+1} \setminus ADT_{k-2}$.

First, $W_k \notin ADT_{k-2}$ because W_k is not Σ_{k-1} -definable ([23]). By an inductive argument over k (see Appendix D), we can build an adt $\mu_k \in ADT_{k+1}$

that captures W_k . We only sketch here the case k = 1. For W_1 , we set $\mu_1 := \operatorname{CO}(b, \operatorname{OR}(\underline{b}[,]\operatorname{CO}(|\geq 2|, \operatorname{AND}(a[, b[))[)))$, depicted in Figure 2a. Note that $\delta(\mu_1) = 2$ and that by a basic use of semantics, we have $Traces(\mu_1) = (ab)^+ = W_1$.



(a) Representation of μ_1

(b) Summary of the results on ADT-hierarchy

Now we have all the material to prove Proposition 1. Indeed, assuming the hierarchy collapses at some level will contradict Proposition 2. Notice that this argument is not constructive as we have no witness of $ADT_k \subsetneq ADT_{k+1}$ but for k = 1 where it can be shown that $(ab)^+ \in ADT_2 \setminus ADT_1$ (see Appendix B, Example 7 and Proposition 3). Our results about the ADT-hierarchy are depicted on Figure 2b.

6 Decision Problems on Attack-Defense Trees

We study classical decision problems on languages, through the lens of adts, with a focus on the role played by the countermeasure-depth in their complexities. The problems are the following.

- The membership problem, written ADT-MEMB, is defined by: **Input:** τ an attack-defense tree and t a trace. **Output:** "YES" if $t \in Traces(\tau)$, "NO" otherwise. - The non-emptiness problem, written ADT-NE, is defined by: **Input:** τ an attack-defense tree. **Output:** "YES" if $Traces(\tau) \neq \emptyset$, "NO" otherwise.

We use notations ADT_k -MEMB and ADT_k -NE whenever the input adds of the respective decision problems are in ADT_k , with a fixed k. Our results are summarized in Table 1, and we below comment on them, row by row.

	ADT_0	ADT_1	$ADT_k \ (k \ge 2)$	ADT
ADT-memb	Ptime	Ptime	Ptime	Ptime
ADT-NE	NP-comp	NP-comp	(k+1)-Expspace	non-elem
			$ ^1 \ge \operatorname{NSPACE}(g(k-5, c\sqrt{\frac{n-1}{3}}))$	
ADT-Equiv	coNP-comp	4-Expspace	(k+2)-EXPSPACE	non-elem
			$ ^2 \ge \operatorname{NSPACE}(g(k-4, c\sqrt{\frac{n-1}{3}}))$	
1; f $h > 5$				

 1 if $k \geq 5.$ 2 if $k \geq 4.$ Table 1: Computational complexities of decision problems on adts.

Regarding ADT-MEMB (first row of Table 1), we recall that adds and SERES are expressively equivalent, but with a translation (Theorem 2) from the former to the latter that is not polynomial. We therefore cannot exploit [9, Theorem 2] for a PTIME complexity of the word membership problem for SERES, and have instead developed a dedicated alternating logarithmic-space algorithm in Appendix E.

Regarding ADT-NE (second row of Table 1), and because SEREs can be translated as adds (see "if" direction in the proof of Theorem 2), the problem ADT-NE inherits from the hardness of the non-emptiness of SERES [21, p. 162]. In its full generality, ADT-NE is therefore non-elementary (last column). Moreover, by our exponential translation of ADT_k into the FO[<]-fragment Σ_{k+1} (Lemma 2), we obtain the (k+1)-EXPSPACE upper-bound complexity for ADT_k -NE (recall satisfiability problem for Σ_ℓ is $(\ell-1)$ -EXPSPACE). Additionally, a lower-bound for ADT_k -NE is directly given by [21, Theorem 4.29]: for SERES that linearly translate as adds with countermeasure-depth k, their non-emptiness is at least NSPACE $(g(k-5, c\sqrt{\frac{n-1}{3}}))$. Interestingly, the Small Model Property (Theorem 1) yields an NP upper-bound complexity for ADT_1 -NE, also applicable for ADT_0 -NE, that is optimal since one can reduce the NP-complete satisfiability of propositional formulas to the non-emptiness of leaf adts.

Finally regarding ADT-EQUIV (last row of Table 1), one can observe that ADT-NE and ADT-EQUIV are very close. First, ADT-NE is a particular case of ADT-EQUIV with the second input adt $\tau_2 := \emptyset$. As a consequence, ADT-EQUIV inherits from the hardness of ADT-NE, and is therefore non-elementary (last column). Also, because deciding the equivalence between τ_1 and τ_2 amounts to

deciding whether $OR(CO(\tau_1, \tau_2), CO(\tau_2, \tau_1)) = \emptyset$, we get reduction from ADT_k -EQUIV into ADT_{k+1} -NE which provides the results announced in Columns 1-3.

7 Discussion

First, we discuss our model of adds with regard to the literature. However, we do not compare with settings where adds leaves are actions [10], as they yield only finite languages, and address other issues [24].

Our adds have particular features, but remain somehow standard. Regarding the syntax, firstly, even though we did not type our nodes as proponent/opponent, the countermeasure operator fully determines the alternation between attack and defense. Also, we introduced the non-standard leaf ϵ for the singleton empty-trace language, not considered in the literature. Still, it is a very natural object in the formal language landscape, and anyhow does not impact our overall computational complexity analysis. Regarding the semantics, it can be shown that the definition of our operator AND together with the reachability goal semantics of the leaves coincides with the acknowledged semantics considered in [1,2].

Second, we discuss our results. We showed the strictness of the ADT-hierarchy in a non-constructive manner. However, exhibiting an element of $ADT_{k+1} \setminus ADT_k$ $(k \geq 2)$ is still an open question. Since $W_1 \in ADT_2 \setminus ADT_1$, languages W_k are natural candidates. We conjecture it is the case and we are currently working on Erenfeucht-Fraissé(EF)-like games for adts (in the spirit of [23] for SERES) to prove it. Moreover, EF-like games for adts may also help to better compare the ADT-hierarchy and the FO[<] alternation hierarchy, in particular, whether the hierarchies eventually coincide. For now, finding a tighter inclusion of Σ_{ℓ} in some ADT_k for each ℓ seems difficult; recall that we established $\Sigma_{\ell} \subseteq ADT_{2\ell+2}$. Any progress in this line would be of great help to obtain tight complexity bounds for ADT_k -NE and ADT_k -EQUIV. Finally, determining the level of a language in the ADT-hierarchy seems as hard as determining its level in the dot-depth hierarchy, recognised as a difficult question [17,3].

References

- M. Audinot, S. Pinchinat, and B. Kordy. Is my attack tree correct? In European Symposium on Research in Computer Security, pages 83–102. Springer, 2017.
- T. Brihaye, S. Pinchinat, and A. Terefenko. Adversarial formal semantics of attack trees and related problems. In P. Ganty and D. D. Monica, editors, *Proceedings* of the 13th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2022, Madrid, Spain, September 21-23, 2022, volume 370 of EPTCS, pages 162–177, 2022.
- V. Diekert and P. Gastin. First-order definable languages. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives [in Honor* of Wolfgang Thomas], volume 2 of Texts in Logic and Games, pages 261–306. Amsterdam University Press, 2008.
- 4. S. Eilenberg. Automata, languages, and machines. Academic press, 1974.

- O. Gadyatskaya, R. R. Hansen, K. G. Larsen, A. Legay, M. C. Olesen, and D. B. Poulsen. Modelling attack-defense trees using timed automata. In *Formal Modeling* and Analysis of Timed Systems: 14th International Conference, FORMATS 2016, Quebec, QC, Canada, August 24-26, 2016, Proceedings 14, pages 35–50. Springer, 2016.
- R. Jhawar, B. Kordy, S. Mauw, S. Radomirović, and R. Trujillo-Rasua. Attack trees with sequential conjunction. In *IFIP International Information Security and Privacy Conference*, pages 339–353. Springer, 2015.
- B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer. Attack-defense trees. Journal of Logic and Computation, 24(1):55–87, 2014.
- B. Kordy, M. Pouly, and P. Schweitzer. Computational aspects of attack-defense trees. In Security and Intelligent Information Systems: International Joint Conferences, SIIS 2011, Warsaw, Poland, June 13-14, 2011, Revised Selected Papers, pages 103–116. Springer, 2012.
- O. Kupferman and S. Zuhovitzky. An improved algorithm for the membership problem for extended regular expressions. In Mathematical Foundations of Computer Science 2002: 27th International Symposium, MFCS 2002 Warsaw, Poland, August 26-30, 2002 Proceedings 27, pages 446-458. Springer, 2002.
- S. Mauw and M. Oostdijk. Foundations of attack trees. In International Conference on Information Security and Cryptology, pages 186–198. Springer, 2005.
- R. McNaughton and S. A. Papert. Counter-Free Automata (MIT research monograph no. 65). The MIT Press, 1971.
- A. R. Meyer. Weak monadic second order theory of succesor is not elementaryrecursive. In Logic Colloquium: Symposium on Logic Held at Boston, 1972–73, pages 132–154. Springer, 2006.
- D. Perrin and J.-E. Pin. First-order logic and star-free sets. Journal of Computer and System Sciences, 32(3):393–406, 1986.
- 14. J. Pin, editor. *Handbook of Automata Theory*. European Mathematical Society Publishing House, Zürich, Switzerland, 2021.
- J. E. Pin and M. P. Schützenberger. Variétés de langages formels, volume 17. Masson Paris, 1984.
- 16. S. Pinchinat, M. Acher, and D. Vojtisek. Towards synthesis of attack trees for supporting computer-aided risk analysis. In C. Canal and A. Idani, editors, Software Engineering and Formal Methods SEFM 2014 Collocated Workshops: HOFM, SAFOME, OpenCert, MoKMaSD, WS-FMDS, Grenoble, France, September 1-2, 2014, Revised Selected Papers, volume 8938 of Lecture Notes in Computer Science, pages 363–375. Springer, 2014.
- T. Place and M. Zeitoun. The tale of the quantifier alternation hierarchy of firstorder logic over words. ACM SIGLOG News, 2(3):4–17, 2015.
- G. Rozenberg and A. Salomaa. Handbook of Formal Languages: Volume 3 Beyond Words. Springer Science & Business Media, 2012.
- I. Schiering and W. Thomas. Counter-free automata, first-order logic, and starfree expressions extended by prefix oracles. Developments in Language Theory, II (Magdeburg, 1995), Worl Sci. Publishing, River Edge, NJ, pages 166–175, 1996.
- 20. B. Schneier. Attack trees. Dr. Dobb's journal, 24(12):21–29, 1999.
- 21. L. J. Stockmeyer. The complexity of decision problems in automata theory and logic. PhD thesis, Massachusetts Institute of Technology, 1974.
- 22. W. Thomas. Classifying regular events in symbolic logic. Journal of Computer and System Sciences, 25(3):360–376, 1982.
- W. Thomas. An application of the ehrenfeucht-fraïssé game in formal language theory. Bull. Soc. Math. France, 16(1):1–21, 1984.

 W. Wideł, M. Audinot, B. Fila, and S. Pinchinat. Beyond 2014: Formal methods for attack tree–based security modeling. ACM Computing Surveys (CSUR), 52(4):1– 36, 2019.

A Figure of τ_{ex_2} in Example 4



B Proof of Theorem 1

We consider the *lift* binary relation between traces (Definition 5), a slight variant of the classic relation of super-word in language theory. We use the lift to define generators of a set of traces (see Definition 6) for τ an adt. We define (in Definition 7) $gen(\tau)$, a finite set of traces, and we show in Proposition 3 that, if $\tau \in ADT_1$, then $gen(\tau)$ is a set of generators of the semantics of τ . We also show in Lemma 5 that an element of $gen(\tau)$ has its size bounded by the number of leaves of τ . Lemma 5 and Proposition 3 are enough to deduce the small model property of ADT_1 , stated in Theorem 1.

Definition 5. A trace t is a lift of a trace $t' = a_1...a_n$, written $t' \sqsubseteq t$, whenever $t \in \Sigma^* a_1...\Sigma^* a_n$.

Notice that \sqsubseteq is an order over Σ^* and that lift relation is like the relation of super-word where the last letters of the respective traces are identical. It can be shown by induction of trees that every $\tau \in ADT_0$, the set $Traces(\tau)$ is \sqsubseteq -upward closed, and is therefore infinite:

Lemma 4. Let $\tau \in ADT_0$. If $t \in Traces(\tau)$, then for each trace t' such that $t \sqsubseteq t'$, we have $t' \in Traces(\tau)$.

Proof. We show this result by induction over the shape of an adt.

If $\tau = \epsilon$, then there is no lift of ϵ . If τ is a non-empty leaf γ , the result holds since the lift relation preserves the last letter.

Now, we assume that the result holds for two adds τ_1 and τ_2 and we need to show the result for $OR(\tau_1, \tau_2)$, $SAND(\tau_1, \tau_2)$ and $AND(\tau_1, \tau_2)$.

The three cases are direct and similar, we only write the case $\tau = \text{SAND}(\tau_1, \tau_2)$: if $t \in Traces(\tau)$ and $t \sqsubseteq t'$ for a certain trace t', then we can write $t = t_1 \cdot t_2$ with $t_1 \in Traces(\tau_1)$ and $t_2 \in Traces(\tau_2)$. From the structure of a lift, we can also write $t' = t'_1 \cdot t'_2$ such that $t_1 \sqsubseteq t'_1$ and $t_2 \sqsubseteq t'_2$, we can then conclude with our induction hypothesis.

From Lemma 4, it is immediate that any non-empty semantics of an adt in ADT_0 is necessarily infinite. We now turn to the notion of *generators* to establish the Small Model Property.

Definition 6. Let $L \subseteq \Sigma^*$. A set of generators of L is a finite subset G of L with the following property: for each non-empty trace $t_L \in L$ there exists $t_{gen} = a_1...a_n \in G$ such that we can write $t_L = w_1a_1...w_na_n$ and for each $t \in \Sigma^*$, if we can write $t = w'_1a_1...w'_1a_n$ with $w'_1a_1 \subseteq w_1a_1$, ..., $w'_na_n \subseteq w_na_n$, then $t \in L$.

Example 6. For the singleton set of propositions $\{p\}$, we consider the set of traces $L = p^+ \setminus \{pp\}$. The set $\{p, ppp\}$ is a set of generators for T. However, the set $\{p\}$ is not because we have $ppp \in T$ and $p \sqsubseteq ppp$ but we also have $p \sqsubseteq pp \sqsubseteq ppp$ and $pp \notin T$.

Example 7. For the singleton set of propositions $\{p\}$, we let a be the valuation that makes p true and b the valuation that makes p false. Thus each trace is a word over the alphabet $\{a, b\}$. Now, remark that the set $(ab)^+$ has no set of generators. Indeed, assume that $G \subseteq (ab)^+$ is a set of generators. We write $(ab)^N$ for the largest element of G, thus $(ab)^N ab \in (ab)^+ \setminus G$. However, for each $t_{gen} \in G$, we have $t_{gen} \sqsubseteq (ab)^N b \sqsubseteq (ab)^N ab$ and $(ab)^N b \notin (ab)^+$, which contradicts Definition 6.

We now define the set $gen(\tau)$ for each adt τ that will be our candidate to show that every of adts in ADT_1 has a set of generators.

Definition 7. Let $\tau \in ADT_1$. The set $gen(\tau)$ is inductively defined as follows:

- $-gen(\epsilon) = \emptyset \text{ and } gen(\gamma) = \{a \in 2^{Prop} : a \models \gamma\},\$
- $gen(OR(\tau_1, \tau_2)) = gen(\tau_1) \cup gen(\tau_2),$
- $gen(SAND(\tau_1, \tau_2)) = gen(\tau_1) \cdot gen(\tau_2),$
- $gen(CO(\tau_1, \tau_2)) = gen(\tau_1) \setminus Traces(\tau_2),$
- $gen(AND(\tau_1, \tau_2)) = (gen(\tau_1) \bowtie gen(\tau_2)) \cap Traces(AND(\tau_1, \tau_2)), where \bowtie is the classic shuffle operator inductively defined as: \varepsilon \bowtie t = t \bowtie \varepsilon = \{t\}, a_1t_1 \bowtie a_2t_2 = a_1 \cdot (t_1 \bowtie a_2t_2) \cup a_2 \cdot (a_1t_1 \bowtie t_2) \text{ and } at_1 \bowtie at_2 = a \cdot (t_1 \bowtie at_2) \cup a \cdot (a_1 \bowtie t_2) \cup a \cdot (t_1 \bowtie t_2).$

We can prove the following Lemma 5 by induction over adds.

Lemma 5. For each trace $t \in gen(\tau)$, its size |t| is bounded by the number of leaves of τ .

Proof. We show this result by induction over the shape of an adt.

If the adt is the empty leaf ϵ , then there is no element in $gen(\tau)$. If adt is a non-empty leaf γ then all elements of $gen(\tau)$ are of size 1.

Now, we assume that the result holds for two adds τ_1 and τ_2 and we show the result for $gen(OR(\tau_1, \tau_2))$, $gen(SAND(\tau_1, \tau_2))$, $gen(AND(\tau_1, \tau_2))$ and $gen(CO(\tau_1, \tau_2))$. For the rest of the proof, we write n_1 the number of leaves of τ_1 and n_2 the number of leaves of τ_2 .

By induction, if $t \in gen(OR(\tau_1, \tau_2))$, then $|t| \leq MAX\{n_1, n_2\}$. If $t \in gen(SAND(\tau_1, \tau_2))$ or $t \in gen(AND(\tau_1, \tau_2))$, then $|t| \leq n_1 + n_2$. Finally, if $t \in gen(CO(\tau_1, \tau_2))$, then $|t| \leq n_1$. In each case, $|t| \leq n_1 + n_2$ and $n_1 + n_2$ is the number of leaves of the four studied adts, which concludes.

If τ has no nested countermeasures yields the following.

Proposition 3. When $\tau \in ADT_1$, the set $gen(\tau)$ is a set of generators for $Traces(\tau)$.

Proof. We show this result by induction over the shape of an adt.

If $\tau = \epsilon$, then its semantics contains only the empty trace. Thus \emptyset is a set of generators. If $\tau = \gamma$, then $Traces(\gamma) = \{a_1...a_n \in \Sigma^* : a_n \models \gamma\} = \Sigma^* \cdot gen(\gamma)$, thus $gen(\gamma) \subseteq Traces(\gamma)$ and is a set of generators.

Now, for two adds τ_1 and τ_2 , we assume that $gen(\tau_1)$ and $gen(\tau_2)$ are sets of generators and we show that $gen(OR(\tau_1, \tau_2))$, $gen(SAND(\tau_1, \tau_2))$ and $gen(AND(\tau_1, \tau_2))$ are sets of generators for their corresponding add.

First of all, for $\tau \in \{OR(\tau_1, \tau_2), SAND(\tau_1, \tau_2), AND(\tau_1, \tau_2), CO(\tau_1, \tau_2)\}$, we have $gen(\tau) \subseteq Traces(\tau)$. Indeed, for $gen(OR(\tau_1, \tau_2))$ and $gen(SAND(\tau_1, \tau_2))$, the composition rule is the same as the one used for $Traces(\tau)$. Moreover, for $gen(AND(\tau_1, \tau_2))$ and $gen(CO(\tau_1, \tau_2))$, we intersect with $Traces(\tau)$. Furthermore, from lemma 5, each element of $gen(\tau)$ is bounded, thus $gen(\tau)$ is a finite set.

It remains to show that for each $t_{\tau} \in Traces(\tau)$, we have that there exists $t_{gen} = a_1...a_n \in gen(\tau)$ such that, $t_{\tau} = t_1a_1...t_na_n$ and for each trace $t = t'_1a_1...t'_na_n$ with $t'_1a_1 \sqsubseteq t_1a_1,...,t'_na_n \sqsubseteq t_na_n$, we have $t \in Traces(\tau)$. We distinguish the four adds operators:

- $-\tau = OR(\tau_1, \tau_2)$. If $t_{\tau} \in Traces(\tau)$, then either $t_{\tau} \in Traces(\tau_1)$ or $t_{\tau} \in Traces(\tau_2)$. With no loss of generality, we assume $t_{\tau} \in Traces(\tau_1)$. Since $gen(\tau_1)$ is a set of generators for τ_1 , there exists $t_{gen} = a_1...a_n \in gen(\tau_1) \subseteq gen(\tau)$ such that $t_{\tau} = t_1a_1...t_na_n$ and for each trace $t = t'_1a_1...t'_na_n$ with $t'_1a_1 \sqsubseteq t_1a_1, ..., t'_na_n \sqsubseteq t_na_n$, we have $t \in Traces(\tau_1) \subseteq gen(\tau)$.
- $\begin{aligned} &-\tau = \operatorname{SAND}(\tau_1,\tau_2). \text{ If } t_{\tau} \in Traces(\tau), \text{ there exists } t_{gen} = a_1...a_n \in gen(\tau) \\ &\text{ such that } t_{\tau} = t_1a_1...t_na_n. \text{ Indeed, we can write } t_{\tau} = t_1 \cdot t_2 \text{ with } t_1 \in Traces(\tau_1) \text{ and } t_2 \in Traces(\tau_2), \text{ thus there exists } t_{gen_1} = a_1...a_i \text{ and } t_{gen_2} = a_{i+1}...a_n \text{ with } t_{gen_1} \in gen(\tau_1) \text{ and } t_{gen_2} \in gen(\tau_2). \text{ Therefore we can consider } \\ &t_{gen} = t_{gen_1} \cdot t_{gen_2}. \text{ Moreover, the following holds: property for each trace } \\ &t = t_1'a_1...t_n'a_n \text{ with } t_1'a_1 \sqsubseteq t_1a_1, ..., t_n'a_n \sqsubseteq t_na_n, \text{ we have } t \in Traces(\tau_1) \subseteq gen(\tau). \text{ Indeed we can simply write } t = t_a \cdot t_b \text{ with } t_a = t_1'a_1...t_i'a_i \text{ and } t_b = t_{i+1}'a_{i+1}...t_na_n. \text{ So, } t_a \in Traces(\tau_1) \text{ and } t_b \in Traces(\tau_2) \text{ and we conclude } \\ &t \in Traces(\tau). \end{aligned}$

- $\begin{aligned} &-\tau = \operatorname{AND}(\tau_1,\tau_2). \text{ First, let's remark that, for a trace } t_\tau \in Traces(\tau), \text{ there} \\ & \text{exists } t_1 \in Traces(\tau_1) \text{ and } t_2 \in Traces(\tau_2) \text{ and } t_{gen} = a_1...a_n \in (t_1 \bowtie t_2) \\ & \text{such that } t_\tau = t_1a_1...t_na_n. \text{ Indeed, we can write } t_\tau = t_a \cdot t_b \text{ with either } t_a \in \\ & Traces(\tau_1) \text{ and } t_\tau \in Traces(\tau_2) \text{ or } t_a \in Traces(\tau_2) \text{ and } t_\tau \in Traces(\tau_1). \\ & \text{If we assume the former case (the latter is symmetrical), then there exists } \\ & t_{gen_1} = b_1...b_m \in gen(\tau_1) \text{ such that } t_a = u_1b_1...u_mb_m \text{ and there exists } t_{gen_2} = \\ & c_1...c_p \in gen(\tau_2) \text{ such that } t_\tau = v_1c_1...v_pc_p. \\ & \text{If we consider the trace } t_{gen} \text{ as the trace obtained from } t_\tau \text{ by removing all valuations appearing neither in } \\ & t_{gen_1} \text{ nor in } t_{gen_2}. \\ & \text{ Then, by construction: } t_{gen} = a_1...a_n \in (t_{gen_1} \bowtie t_{gen_2}) \\ & \text{ and } t_\tau = t_1a_1...t_na_n. \\ & \text{ The rest of the proof is similar to the SAND as we show } \\ & \text{ that a trace } t = t_1'a_1...t_na_n \text{ with } t_1'a_1 \sqsubseteq t_1a_1, ..., t_n'a_n \sqsubseteq t_na_n \text{ always can be } \\ & \text{ written as } t = t_1 \cdot t_2 \text{ with } t_1 \in Traces(\tau_1) \text{ and } t \in Traces(\tau_2). \end{aligned}$
- $-\tau = \operatorname{CO}(\tau_1, \tau_2)$. For a trace $t_{\tau} \in Traces(\tau)$, there exists $t_{gen_1} = a_1...a_n \in gen(\tau_1)$ such that, $t = t'_1a_1...t'_na_n$ with $t'_1a_1 \sqsubseteq t_1a_1, ..., t'_na_n \sqsubseteq t_na_n$, we have $t \in Traces(\tau_1)$. Moreover, if $t_{\tau} \in Traces(\operatorname{CO}(\tau_1, \tau_2))$, then $t_{\tau} \notin Traces(\tau_2)$. Furthermore, if $\operatorname{CO}(\tau_1, \tau_2)$ is in ADT_1 , then $\tau_2 \in ADT_0$, thus from Lemma 4, for each t' such that $t' \sqsubseteq t_{\tau}$ we have $t' \notin Traces(\tau_2)$. This shows us that $t_{gen_1} \in Traces(\operatorname{CO}(\tau_1, \tau_2))$ and by taking $t_{gen} = t_{gen_1}$, the property we need to prove holds.

Remark 3. An adt $\tau \in ADT_0$ with a non-empty semantics always can be written as an OR over SAND over leaves. Indeed, for an adt $\tau \in ADT_0$ and its set of generator $gen(\tau) = \{t_1, ..., t_n\}$, by combining Proposition 3 and Lemma 4, we have: $Traces(\tau) = \bigcup_{a_1...a_m \in gen(\tau)} \Sigma^* a_1...\Sigma^* a_m$. Therefore, for we can write $\tau = OR(\tau_{t_1}, ..., \tau_{t_n})$, Where $\tau_{a_1...a_m} = SAND(\gamma_1, ..., \gamma_m)$ with γ_j is the formula only satisfied by a_j .

By Proposition 3, for $\tau \in ADT_1$, if some non-empty trace $t_{\tau} \in Traces(\tau)$, there is some $t_{gen} \in gen(\tau)$ with $t_{gen} \sqsubseteq t_{\tau}$ that is a witness. Together with Lemma 5, we conclude the proof of Theorem 1. Remark that, since $ADT_0 \subseteq$

Observe that our proof technique heavily relies on the notion of set of generators, which calls for the existence of a set of generators for adts above ADT_1 . There is an adt in ADT_2 that shows it is hopeless: this adt specifies the language $(ab)^+$ (see Section 5) but has no set of generators (see Example 7).

C Complements of Section 4

 ADT_1 , the result also holds for ADT_0 .

Lemma 1. – For any $t \in \Sigma^*$, $t \models \psi_{\tau}$ iff $t \in Traces(\tau)$. – For any $adt \tau$, formula ψ_{τ} is of size exponential in $|\tau|$.

Proof. We will show this result by induction over the adt.

If $\tau = \epsilon$, then we have clearly that $\psi_{\epsilon} = \{\varepsilon\}$. Similarly, for a Boolean formula γ , we have that $Words(\psi_{\gamma})$ is all traces finishing with a valuation a such that $a \models \gamma$. Thus, the property holds for leaves attack-defense trees.

Now, if we assume that the property is true for τ_1 and τ_2 , then, since semantics for the OR operator is a union of sets and semantics of C operator is a difference of sets, the property for $\psi_{\text{OR}(\tau_1,\tau_2)}$ and $\psi_{\text{CO}(\tau_1,\tau_2)}$ trivially holds.

The formulas for AND and SAND operators are a little more difficult to understand since we need to distinguish whether an empty trace is in semantics of τ_1 or τ_2 . Indeed, a t is in semantics of $\text{SAND}(\tau_1, \tau_2)$ if and only if we can find t_1 and t_2 such that $t_1 \in Traces(\tau_1)$ and $t_2 \in Traces(\tau_2)$ if t_1 is nonempty, then this is equivalent to say that there exists a position in the trace where we can cut as done in the first parenthesis of $\psi_{\text{SAND}(\tau_1, \tau_2)}$. Otherwise, if t_1 is empty, then we handle this case with the use of $\psi_{\tau_1}^{\leq 0}$ in the second parenthesis of $\psi_{\text{SAND}(\tau_1, \tau_2)}$. The idea for the AND operator is completely symmetrical except that we also need to consider the case where τ_2 has the empty trace in its semantics.

Finally, it is clear that ψ_{τ} is of size exponential as subformulas for subtrees may be duplicated (for SAND and AND).

For the rest of this section, we often switch from an adt τ to its corresponding FO[<]-formulas ψ_{τ} as defined in Section 4, always while assuming Lemma 1.

Let us recall some basic results on Σ_k and Π_k hierarchies.

Remark 4 ([22], Lemma 2.4).

- (a) The negation of a Σ_k -formula is equivalent to a Π_k -formula.
- (b) A disjunction or conjunction of Σ_k -formulas is equivalent to a Σ_k formula.
- (c) A Boolean combination of Σ_k -formula is equivalent to a Σ_{k+1} -formula.
- (d) The statements (a)-(c) hold in dual form for Π_k -formulas.

Lemma 2. 1. $ADT_0 \subseteq \Sigma_2 \cap \Pi_2$ – with an effective translation. 2. For every k > 0, $ADT_k \subseteq \Sigma_{k+1}$ – with an effective translation.

Proof. Regarding Item 1, for both cases $ADT_0 \subseteq \Sigma_2$ and $ADT_0 \subseteq \Pi_2$, we construct for each $\tau \in ADT_0$ an FO[<]-formula ψ_{τ} by induction over τ such that $Traces(\tau) = \{t \in \Sigma^* : t \models \psi_{\tau}\}.$

To prove that $ADT_0 \subseteq \Sigma_2$, one can refer to the translation from ADT into FO[<], and easily verify that formula $\psi_{\tau} \in \Sigma_2$.

On the contrary, proving that $ADT_0 \subseteq \Pi_2$ requires some work. First remark that $\psi_{\epsilon} \in \Pi_1 \subseteq \Pi_2$. Moreover, recall that for a Boolean formula γ ,

$$\psi_{\gamma} \equiv \forall y \exists x (\neg (x < y) \land \hat{\gamma}(x))$$

where $\widehat{\gamma}(x) := \bigvee_{a \models \gamma} a(x)$.

Indeed, both formulae hold for a trace $a_1...a_n$ if, and only if, $a_n \models \gamma$.

Recall (Remark 3) that an arbitrary adt $\tau \in ADT_0$, can be equivalently written $OR(\tau_1, ..., \tau_n)$ where $\tau_1, ..., \tau_n$ are either the empty leaf ϵ or of the form SAND $(\gamma_1, ..., \gamma_m)$. Consider the following Π_2 -formula for $SAND(\gamma_1, ..., \gamma_m)$:

$$\theta_{\text{SAND}(\gamma_1, \gamma_2, \dots, \gamma_m)} := \forall y \; \exists x_1 \; \exists x_2 \; \dots \; \exists x_m \quad x_1 < x_2 < \dots < x_m \; \land \; y \le x_m \; \land \\ \hat{\gamma_1}(x_1) \land \hat{\gamma_2}(x_2) \land \dots \land \hat{\gamma_m}(x_m)$$

Clearly, for a word $w \in \Sigma^*$, we have $w \in Traces(SAND(\gamma_1, ..., \gamma_m))$ if, and only if, $w \in \Sigma^* a_1 \cdot ... \cdot \Sigma^* a_m$ with $a_1 \models \gamma_1, ..., a_m \models \gamma_m$, which is exactly described by $\theta_{\text{SAND}(\gamma_1, \gamma_2, ..., \gamma_m)} \in \Pi_2$. We now get back to the construction of the Π_2 -formula for τ , that is a disjunction of either ψ_{ϵ} , or formulae of the form $\theta_{\text{SAND}(\gamma_1,\gamma_2,\ldots,\gamma_m)}$). Since they all belong to Π_2 , so does their disjunction.

This concludes the proof of Item 1.

For Item 2 of Lemma 2, we start with some notations: let k > 0, for each $d \in \mathbb{N}$, we define $ADT_k(d) \subseteq ADT_k$ as follows.

$$\begin{cases} ADT_k(0) := ADT_{k-1} \cup \{ \operatorname{CO}(\tau_1, \tau_2) : \tau_1, \tau_2 \in ADT_{k-1} \} \\ ADT_k(d+1) := ADT_k(d) \cup \{ \operatorname{OP}(\tau_1, \tau_2) : \operatorname{OP} \text{ arbitrary, and } \tau_1, \tau_2 \in ADT_k(d) \} \end{cases}$$

We clearly have $\bigcup_{d\in\mathbb{N}} ADT_k(d) = ADT_k$. We establish by a double induction over k > 0 and $d \ge 0$ that for each $d \in \mathbb{N}$, $ADT_k(d) \subseteq \Sigma_{k+1}$, which clearly entails $ADT_k \subseteq \Sigma_{k+1}$.

k = 1: that is to show $ADT_1(d) \subseteq \Sigma_2$, for every $d \in \mathbb{N}$.

d = 0: Let adt $\tau \in ADT_1(0)$. If $\tau \in ADT_0$, it is immediate by Item 1 that $Traces(\tau)$ is definable by a Σ_2 formula.

Otherwise $\tau = co(\tau_1, \tau_2)$ where τ_1 and $\tau_2 \in ADT_0$. Again, by Item 1 there exists formulas $\psi_1 \in \Sigma_2$ and $\psi_2 \in \Pi_2$ that characterize $Traces(\tau_1)$ and $Traces(\tau_2)$ respectively, so that, by the very definition of operator co, formula $\psi_1 \wedge \neg \psi_2$ characterizes $Traces(\tau)$ and belongs to Σ_2 (see Remark 4).

d > 0: Let $\tau \in ADT_1(d)$; note that $\delta(\tau) \leq 1$. If $\tau \in ADT_1(d-1)$, we use the induction hypothesis over d-1 and we are done. Otherwise, $\tau = \mathsf{OP}(\tau_1, \tau_2).$

Suppose $OP \in \{OR, SAND, AND\}$, and because $\tau_1, \tau_2 \in ADT_1(d-1)$, we know by induction over d-1 that there exist $\psi_1, \psi_2 \in \Sigma_2$ that characterize $Traces(\tau_1)$ and $Traces(\tau_2)$ respectively. For OP = OR (resp. = SAND, AND), formula $\psi = \psi_1 \lor \psi_2$ (resp. = $\exists x [\psi_1 \leq x \land \psi_2 > x] \lor (\psi_1 \leq 0 \land \psi_2)$, = $\exists x [(\psi_1 \leq x \land \psi_2) \lor (\psi_2 \leq x \land \psi_1)] \lor (\psi_1 \leq 0 \land \psi_2) \lor (\psi_2 \leq 0 \land \psi_1))$ characterises $Traces(OP(\tau_1, \tau_2))$ and belongs to Σ_2 (see Remark 2).

Now, suppose OP = CO. Therefore, $\tau_2 \in ADT_0(d-1)$ otherwise $\delta(\tau) = 2$. By Item 1, we can assume that $\psi_2 \in \Pi_2$ Now formula $\psi = \psi_1 \wedge \neg \psi_2$ characterises $Traces(\tau)$ and is indeed in Σ_2 (see Remark 4).

- k > 1: that is to show $ADT_k(d) \subseteq \Sigma_{k+1}$, for every $d \in \mathbb{N}$.
 - d = 0: Let $\tau \in ADT_k(0)$. If $\tau \in ADT_{k-1}$, we resort to the induction hypothesis over k to conclude since $\Sigma_k \subseteq \Sigma_{k+1}$. Otherwise, $\tau = \operatorname{CO}(\tau_1, \tau_2)$ with τ_1 and $\tau_2 \in ADT_{k-1}$. By induction over k-1, $Traces(\tau_1)$ and $Traces(\tau_2)$ can be equivalently characterized by Σ_k -formulas, say ψ_1 and ψ_2 respectively. Now the formula $\psi := \psi_1 \wedge \neg \psi_2$ characterizes $Traces(\tau)$ and, by Remark 4 is clearly in Σ_{k+1} .
 - d > 0: Let $\tau \in ADT_k(d)$. If $\tau \in ADT_k(d-1)$, we use the induction hypothesis on d-1. Otherwise, $\tau = \mathsf{OP}(\tau_1, \tau_2)$ with $\tau_1, \tau_2 \in ADT_k(d-1)$.

We can proceed in a way similar to what we did for the previous case k = 1, d > 0, by noticing that for the case where OP = CO, namely $\tau = CO(\tau_1, \tau_2)$, the tree $\tau_2 \in ADT_{k-1}$ otherwise τ would not belong to ADT_k .

Lemma 3. 1. $B_{\ell} \subseteq ADT_{2\ell+2}$, and therefore $\Sigma_{\ell} \subseteq ADT_{2\ell+2}$ 2. $\Sigma_1 \subseteq ADT_0$ – with an effective translation.

Proof. For Item 1, the proof is conducted by induction over *ell*. We start by showing that $B_0 \subseteq ADT_2$. We recall that a language L is in B_0 if it is finite or co-finite. We distinguish the two cases.

If L is finite, we have that $L = \{t_1, ..., t_n\} = \{t_1\} \cup ... \cup \{t_n\}$ where $t_1, ..., t_n \in \Sigma^*$. For a trace $t = a_1, ..., a_n$, we use the notation τ_t to express the adt $\text{SAND}(\underline{a_1}, ..., \underline{a_n})$. Clearly, $Traces(\tau_t) = \{t\}$ and $\tau_t \in ADT_1$. Then if we consider the adt $\overline{\tau_L} = OR(\tau_{t_1}, ..., \tau_{t_n})$, we have by construction that $Traces(\tau_L) = L$. Moreover $\tau_L \in ADT_1$ since an adt τ_t only uses non-nested CO.

If L is co-finite, then it can be written as $\Sigma \setminus L'$ with L' a finite language. Thus, by definition of CO, if we consider $\tau_L = \operatorname{CO}(\top, \tau_{L'})$, we have $Traces(\tau_L) = L$. We also have $\tau_L \in ADT_2$. Since $\tau_{L'} \in ADT_1$. We concludes that $B_0 \subseteq ADT_2$.

Now, for k > 0, we assume $B_{\ell-1} \subseteq ADT_{2\ell}$ and we show that $B_{\ell} \subseteq ADT_{2\ell+2}$.

Let $L \in B_{\ell}$, so L is a Boolean combination of languages of the form $L_1 \cdot \ldots \cdot L_n$ with $L_1, ..., L_n \in B_{\ell-1}$. With no loss of generality, we can assume the Boolean combination written in disjunctive normal form where each conjunct is of the form $M_1 \cap ... \cap M_m$, where the sets M_i can be written either as $L_1 \cdot ... \cdot L_n$ or as $(L_1 \cdot \ldots \cdot L_n)^c$ with $L_1, \ldots, L_n \in B_{\ell-1}$. Moreover, We can rewrite each conjunct by using the equality $M_1 \cap ... \cap M_m = (M_1^c \cup ... \cup M_m^c)^c$ to suppress all intersections of the Boolean expression, it results an expression only using union operators and complement operators. Furthermore, the maximal tower of complementary operator is 2. By induction hypothesis, we know that an expression $L_1 \cdot \ldots \cdot L_n$ with $L_1, ..., L_n \in B_{\ell-1}$ is definable in $ADT_{2\ell}$, indeed we consider the SAND of the adds associated with the languages $L_1, \ldots L_n$. Then, we can use the Boolean expression to construct the adt for L, where the union is replaced by the OR and the complementation is replaced by NOT(.). Since the maximal tower of complementary operator of the Boolean expression is 2, we need at most two more nested CO to express the full Boolean expression. Therefore, the final adt is in $ADT_{2\ell+2}$, which concludes.

Item 1 of Lemma 3 is an immediate corollary of the fact that $\Sigma_k \subseteq B_\ell$.

Regarding Item 2 of Lemma 3, let $\psi \in \Sigma_1$ a first-order formula of the form $\exists x_1, ..., \exists x_n \psi'(x_1, ..., x_n)$ with ψ' quantifier-free. With no loss of generality, we consider $\psi' = C_1 \lor ... \lor C_m$ in disjunctive normal form, in other words $C_1, ..., C_m$ only use conjunctions of literals of the form $x_i < x_j$, or a(x), or their negations. As expected, the adt corresponding to ψ is of the form $OR(\tau_{C_1}, ..., \tau_{C_m})$, and we now explain how to build τ_C .

If clause C is not satisfiable, we associate adt \perp , otherwise we proceed as follows.

First, we show that with no loss of generality clause C of ψ' specifies a linear order over $x_1, ..., x_n$ together with literals based on predicates $a(x_i)$. To do so, we

decompose the clause C, according to $C^{<} \wedge C^{a}$, where the first conjunct gathers all <-based literals and the second gathers the rest. Now, the sub-clause $C^{<}$ naturally induces a partial order between all free variables $x_{1}, ..., x_{n}$ of ψ' . It is easy to see that clause $C^{<}$ is equivalent to a disjunctive formula $\psi_{C^{<}}$ where each disjunct describes a possible linearization of this partial order. For example, if $C^{<} = x < y \wedge \neg y < z$ then $\psi_{C^{<}} = (x < y \wedge y = z) \lor (x < z \wedge z < y) \lor (z = x \wedge x < y)$, so that C is equivalent to $(x < y \wedge y = z \wedge C^{a}) \lor (x < z \wedge z < y) \lor (z < x \wedge x < y) \land (z = x \wedge x < y \wedge C^{a}) \lor (z < x \wedge x < y \wedge C^{a})$, which concludes.

In clause C (which now specifies a linear order of the variables), if $x_i = x_j$ with $i \neq j$, we replace all occurrences of x_j by x_i in C and obtain a clause equivalent to C of the form $x_{i_1} < ... < x_{i_\ell} \land P_1(x_{i_1}) \land ... \land P_\ell(x_{i_\ell})$ where each $P_k(x_{i_k})$ is the conjunction of literals of the form $a(x_{i_k})$ and $\neg a(x_{i_k})$. For convenience, we still write C this clause.

We associate with clause C the adt $\tau_C = \text{SAND}(\gamma_1, ..., \gamma_\ell) [$ where γ_i is made of the conjunction of the propositional formulas stemming from the valuations, or negation of valuations, that occur in $P_k(x_{i_k})$. It can be shown that a trace $t \in Traces(\tau_C)$ if, and only, if $t \models \exists x_{i_1} ... \exists x_{i_\ell} x_{i_1} < ... < x_{i_\ell} \land P_1(x_{i_1}) \land ... \land P_\ell(x_{i_\ell})$.

D Complements of Section 5

We start this section with some results and definitions to help us to prove Proposition 2. We then conduct the proof of Proposition 2, and we finish with the proof of Lemma 8, a useful result to prove Proposition 1.

Lemma 6. Let $t_1, t_2, t \in \{a, b\}^*$. The following assertions hold:

- 1. $||t_1 \cdot t_2|| = ||t_1|| + ||t_2||$
- 2. If $||t|| = k \ge 0$, then, for each $i \in \{0, 1, ..., k\}$ there exists $t' \le t$ such that ||t'|| = i.
- 3. If ||t|| = k > 0, then there exists t_1, t_2 such that $t = t_1 \cdot a \cdot t_2$ with $||t_2|| = 0$ and for each $t'_2 \leq t_2$, we have $||t'_2|| \geq 0$

Item 1 is trivial from the definition of $\|.\|$, Item 2 stems from the classic "Intermediate Value Theorem" in mathematics – that is applicable since extending a trace t with a letter only increments or decrements $\|t\|$ by 1. Item 3 is obtained by an application of Item 2.

We now start the proof of Proposition 2.

Proposition 2. For each k > 0, $W_k \in ADT_{k+1} \setminus ADT_{k-2}$.

Proof. Recall $W_k \notin ADT_{k-2}$ (page 11). We here focus on proving $W_k \notin ADT_{k+1}$.

We introduce companion languages of W_k , that were introduced in [23] namely W_k^+ and W_k^- :

 $- W_k^+ := \{ w \in \Sigma^* \text{ s.t. } \|w\| = k \text{ and for each } w' \preceq w, 0 \le \|w'\| \le k \}, \\ - W_k^- := \{ w \in \Sigma^* \text{ s.t. } \|w\| = -k \text{ and for each } w' \preceq w, -k \le \|w'\| \le 0 \},$

In [23], alternative definitions of W_k , W_k^+ and W_k^- are provided:

- $\begin{array}{l} W_0 = W_0^+ = W_0^- := \{\varepsilon\}; \\ W_{k+1} := (W_k^+ a \varSigma^* \cap \varSigma^* b W_k^-) \setminus (\varSigma^* a W_k^+ a \varSigma^* \cup \varSigma^* b W_k^- b \varSigma^*); \\ W_{k+1}^+ := (W_k^+ a \varSigma^* \cap \varSigma^* a W_k^+) \setminus (\varSigma^* a W_k^+ a \varSigma^* \cup \varSigma^* b W_k^- b \varSigma^*); \\ W_{k+1}^- := (W_k^- b \varSigma^* \cap \varSigma^* b W_k^-) \setminus (\varSigma^* a W_k^+ a \varSigma^* \cup \varSigma^* b W_k^- b \varSigma^*). \end{array}$

In the following, we may use the most convenient characterisation of W_k , W_k^+ and W_k^- . Our proof relies on a third characterisation. We build languages U_k, U_k^+ and U_k^- where we eliminate the \cap operator occurring in the recursive definitions of W_k , W_k^+ and W_k^- . For $w \in \{a, b\}^*$ we let its swap be the word swap(w) obtained by swapping a and b in w. We lift this operator to languages like usual: $swap(L) = \{swap(w) : w \in L\}$. Note that $W_k^- = swap(W_k^+)$.

We set:

$$- U_{0} = U_{0}^{+} = U_{0}^{-} := \{\varepsilon\};$$

$$- U_{k+1} := (U_{k}^{+}a\Sigma^{*}bU_{k}^{-}) \setminus (\Sigma^{*}aU_{k}^{+}a\Sigma^{*}\cup\Sigma^{*}bU_{k}^{-}b\Sigma^{*});$$

$$- U_{k+1}^{+} := ((U_{k}^{+}a\Sigma^{*}aU_{k}^{+}) \cup (\bigcup_{i \leq k} U_{i}aU_{k}^{+})) \setminus (\Sigma^{*}aU_{k}^{+}a\Sigma^{*}\cup\Sigma^{*}bU_{k}^{-}b\Sigma^{*});$$

$$- U_{k+1}^{-} := swap(U_{k+1}^{+}).$$

Lemma 7. For every $k \ge 0$, we have $W_k = U_k$, $W_k^+ = U_k^+$ and $W_k^- = U_k^-$.

Proof. The proof is conducted by induction over k. By definition, $W_0 = U_0$, $W_0^+ = U_0^+$ and $W_0^- = U_0^-$. For the rest of the proof, we fix k > 0.

- We start to show $W_{k+1} = U_{k+1}$. Namely, by replacing U_k (respectively U_k^+ , U_k^{-}) by W_k (respectively W_k^{+}, W_k^{-}) that:

$$(W_k^+ a \Sigma^* \cap \Sigma^* b W_k^-) \setminus (\Sigma^* a W_k^+ a \Sigma^* \cup \Sigma^* b W_k^- b \Sigma^*)$$
$$= (W_k^+ a \Sigma^* b W_k^-) \setminus (\Sigma^* a W_k^+ a \Sigma^* \cup \Sigma^* b W_k^- b \Sigma^*)$$

It is enough to show that $(W_k^+ a \Sigma^* \cap \Sigma^* b W_k^-) = (W_k^+ a \Sigma^* b W_k^-)$ for the equality to hold. Because $(W_k^+ a \Sigma^* \cap \Sigma^* b W_k^-) \supseteq (W_k^+ a \Sigma^* b W_k^-)$ is clear, we focus on showing $(W_k^+ a \Sigma^* \cap \Sigma^* b W_k^-) \subseteq (W_k^+ a \Sigma^* b W_k^-)$. Let w be in $W_k^+ a \Sigma^* \cap \Sigma^* b W_k^-$. As $w \in W_k^+ a \Sigma^*$, we can write $w = u_1 \cdot a \cdot u_2$ with $u_1 \in W_k^+$ and let n be the position of the distinguished a occurrence in w. As we also have $w \in \Sigma^* b W_k^-$, we can write $w = v_2 \cdot b \cdot v_1$ with $v_1 \in W_k^$ and let m be the position of the distinguished b occurrence in w. Clearly $n \neq m$. Moreover, we cannot have m < n. Indeed, if m < n we can write $w = v_2 \cdot b \cdot w' \cdot a \cdot u_2$ with $v_2 \cdot b \cdot w' = u_1 \in W_k^+$ and $w' \cdot a \cdot u_2 = v_1 \in W_k^-$. Since v_2 is a prefix of $u_1 \in W_k^+$, we have $||v_2|| \leq k$, and therefore $||v_2 \cdot b|| \leq k-1$ (by Item 1 of Lemma 6). Moreover, since w' is a prefix of $v_1 \in W_k^-$, we have $||w'|| \leq 0$. We conclude that $||v_2 \cdot b \cdot w'|| \leq k-1$ (by Item 1 of Lemma 6), which contradicts $v_2 \cdot b \cdot w' \in W_k^+$. Therefore n < m, and $w \in (W_k^+ a \Sigma^* b W_k^-)$, which concludes.

- We now prove that $W_{k+1}^+ = U_{k+1}^+$. Because the inclusion $W_{k+1}^+ \supseteq U_{k+1}^+$ is clear, we focus on $W_{k+1}^+ \subseteq U_{k+1}^+$. Namely, by replacing U_k (respectively U_k^+, U_k^-) by W_k (respectively W_k^+, W_k^-) that:

$$(W_k^+ a \Sigma^* \cap \Sigma^* a W_k^+) \setminus (\Sigma^* a W_k^+ a \Sigma^* \cup \Sigma^* b W_k^- b \Sigma^*) \subseteq (W_k^+ a \Sigma^* a W_k^+) \cup (\bigcup_{i \le k} W_i a W_k^+)) \setminus (\Sigma^* a W_k^+ a \Sigma^* \cup \Sigma^* b W_k^- b \Sigma^*)$$

Let $w \in W_{k+1}^+$. As, on the one hand, $w \in W_k^+ a \Sigma^*$, we can write $w = u_1 \cdot a \cdot u_2$ with $u_1 \in W_k^+$ and let n be the position of the distinguished a occurrence in w. As, on the other hand, we also have $w \in \Sigma^* a W_k^+$, we can write $w = v_2 \cdot a \cdot v_1$ with $v_1 \in W_k^+$ and let m be the position of the distinguished a occurrence in w.

First of all, $n \neq m$: indeed, if n = m, we can write $w = u_1 \cdot a \cdot v_1$ with $u_1, v_1 \in W_k^+$. Since $||u_1 \cdot a|| = k + 1$, we can write $u_1 \cdot a = q_1 \cdot a \cdot q_2$ with $q_2 \in W_k^+$. Since $v_1 \in W_k^+$, we can write $v_1 = a \cdot v'_1$. So that $w = q_1 \cdot a \cdot q_2 \cdot a \cdot v'_1 \in (\Sigma^* a W_k^+ a \Sigma^*)$ which entails $w \notin W_{k+1}$, leading to a contradiction.

We distinguish the two remaining cases.

- If n < m, then we have $w \in (W_k^+ a \Sigma^* a W_k^+)$, which entails $w \in U_{k+1}^+$.
- If m < n, we can write $u_1 = p_1 \cdot a \cdot p_2 \in W_k^+$ and $v_1 = p_2 \cdot a \cdot p_3 \in W_k^+$, where $w = p_1 \cdot a \cdot p_2 \cdot a \cdot p_3$. Notice that $||p_1|| \ge 0$, as $u_1 \in W_k^+$, but we show that $||p_1|| = 0$: if $||p_1|| > 0$, we can rewrite $p_1 = q_1 \cdot a \cdot q_2$ with $||q_2|| = 0$ and for each q'_2 prefix of q_2 , we have $||q'_2|| \ge 0$ (by Item 3 of Lemma 6). Moreover, $||a \cdot p_2 \cdot a \cdot p_3|| = k + 1$ (by Item 1 of Lemma 6 since $||p_2 \cdot a \cdot p_3|| = k$), thus we can write $a \cdot p_2 \cdot a \cdot p_3 = r_1 \cdot a \cdot r_2$ with $r_1 \in W_k^+$ (by Item 2 of Lemma 6). Therefore, $w = q_1 \cdot a \cdot q_2 \cdot r_1 \cdot a \cdot r_2$ with $q_2 \cdot r_1 \in W_k^+$ (by Item 1 of Lemma 6), so that $w \in (\Sigma^* a W_k^+ a \Sigma^*)$, which contradicts $w \in W_{k+1}$.

Since $||p_1|| = 0$, we have $p_1 \in \bigcup_{i \leq k} W_i$, and we conclude $w = p_1 \cdot a \cdot v_1 \in \bigcup_{i \leq k} W_i a W_k^+$. Also because $w \in W_{k+1}^+$, $w \notin (\Sigma^* a W_k^+ a \Sigma^* \cup U_k^+)$

- $\Sigma^* b W_k^{-} b \Sigma^*$) and we obtain $w \in U_{k+1}^+$, which concludes.
- Finally, $W_{k+1}^{-} = U_{k+1}^{-}$ because $W_{k+1}^{-} = swap(W_{k+1}^{+})$ and $U_{k+1}^{-} = swap(U_{k+1}^{+})$ and we have already shown $W_{k+1}^{+} = U_{k+1}^{+}$.

This conclude the proof of Lemma 7.

We use Lemma 7 to construct three adts μ_k, μ_k^+ and μ_k^- such that $Traces(\mu_k) = W_k$, $Traces(\mu_k^+) = W_k^+$ and $Traces(\mu_k^-) = W_k^-$, $\delta(\mu_k) = \delta(\mu_k^+) = \delta(\mu_k^-) = k + 1$ which achieves the proof of *Proposition* 2. The proof is conducted by induction over $k \ge 1$.

To capture W_1 , we propose the following add depicted in fig. 2a:

$$\mu_1 := \operatorname{CO}(b, \operatorname{OR}(\underline{b}\lceil,]\operatorname{CO}(|\geq 2|, \operatorname{AND}(a\lceil, b\lceil))\rceil)),$$

We prove $Traces(\mu_1) = (ab)^+ = W_1$: we point to Example 2 for the semantics for ALL, NOT (τ) and $]\tau[$. First of all, AND(a[,b[)) defines the set of all traces with at least one occurrence of a and one occurrence of b. Thus $Traces(CO(|\geq 2|, AND(a[,b[))) = aa^+ \cup bb^+$. Write $\tau := CO(|\geq 2|, AND(a[,b[)))$. So, $Traces(]\tau[) = \Sigma^*.(aa^+ \cup bb^+).\Sigma^* = \Sigma^*.aa.\Sigma^* \cup \Sigma^*.bb.\Sigma^*$. On this basis, $Traces(\mu_1) = Traces(CO(b, OR(b[,]\tau[))) = \Sigma^*b \setminus (b\Sigma^* \cup \Sigma^*.aa.\Sigma^* \cup \Sigma^*.bb.\Sigma^*) = (ab)^+ = W_1$.

We now compute $\delta(\mu_1)$ since $\delta(\tau) = 1$, we have $\delta(\mu_1) = \delta(\operatorname{CO}(b, \operatorname{OR}(\underline{b}[,]\tau[))) = \max\{0, \max\{1, 1\} + 1\} = 2$.

Similarly, we define:

$$- \mu_1^+ := \operatorname{CO}(a, \operatorname{OR}(\underline{b}\lceil,]\operatorname{CO}(|\geq 2|, \operatorname{AND}(a\lceil, b\lceil))\lceil));) \\ - \mu_1^- := \operatorname{CO}(b, \operatorname{OR}(\underline{a}\lceil,]\operatorname{CO}(|\geq 2|, \operatorname{AND}(a\lceil, b\lceil))\rceil)).$$

As done for μ_1 , one can verify that $Traces(\mu_1^+) = (ab)^*a = W_1^+$ and $Traces(\mu_1^-) = b(ab)^* = W_1^-$ and that $\delta(\mu_1^+) = \delta(\mu_1^-) = 2$.

We make use of U_k , U_k^+ and U_k^- to inductively define μ_k , μ_k^+ and μ_k^- . For readability, we introduce for $k \ge 2$ the subtrees ξ_k and ζ_k and we extend the definition of swap(.) to add by applying the swap to leaves.

$$\xi_k := \operatorname{OR}(\operatorname{SAND}(a, \mu_k^+, \underline{a}[), \operatorname{SAND}(b, \mu_k^-, \underline{b}[)))$$

 $\zeta_k := \operatorname{OR}(\operatorname{SAND}(\mu_1, \underline{a}, \mu_k^+), \operatorname{SAND}(\mu_2, \underline{a}, \mu_k^+), \dots, \operatorname{SAND}(\mu_k, \underline{a}, \mu_k^+))$

And μ_k , μ_k^+ and μ_k^- are defined by:

$$-\mu_{k} := \operatorname{CO}(\operatorname{SAND}(\mu_{k-1}^{+}, \underline{a}, \operatorname{ALL}, \underline{b}, \mu_{k-1}^{-}), \xi_{k-1}) -\mu_{k}^{+} := \operatorname{CO}(\operatorname{OR}(\operatorname{SAND}(\mu_{k-1}^{+}, \underline{a}, \operatorname{ALL}, \underline{a}, \mu_{k-1}^{+}), \zeta_{k-1}), \xi_{k-1}) -\mu_{k}^{-} := swap(\mu_{k}^{+})$$

The adds μ_k , μ_k^+ and μ_k^- are built in such a way that a direct application of add semantics yields $W_k = Traces(\mu_k)$, $W_k^+ = Traces(\mu_k^+)$ and $W_k^- = Traces(\mu_k^-)$.

It remains to show that $\delta(\mu_k) = \delta(\mu_k^+) = \delta(\mu_k^-) = k+1$ assuming $\delta(\mu_{k-1}) = \delta(\mu_{k-1}^-) = k$. By definition, $\delta(\mu_k) = \max\{\delta(\tau_1), \delta(\xi_{k-1})+1\}$ with $\tau_1 = \text{SAND}(\mu_{k-1}^+, \underline{a}, \text{ALL}, \underline{b}, \mu_{k-1}^-)$. By using Example 5, we have $\delta(\tau_1) = \max\{\delta(\mu_{k-1}^+), 1, 0, 1, \delta(\mu_{k-1}^-)\}$ and $\delta(\xi_{k-1}) = \max\{0, \delta(\mu_{k-1}^+), 1, 0, \delta(\mu_{k-1}^-), 1\}$. Applying the induction hypothesis over $\delta(\mu_{k-1}^+) = \delta(\mu_{k-1}^-) = k$, we obtain: $\delta(\tau_1) = \max\{k, 1, 0, 1, k\} = k$ and $\delta(\tau_2) = \max\{0, k, 1, 0, k, 1\} = k$, thus $\delta(\mu_k) = \max\{k, k+1\} = k+1$, which concludes. Similarly, we can establish $\delta(\mu_k^+) = \delta(\mu_k^-) = k + 1$.

We have shown $W_k = Traces(\mu_k)$ with $\delta(\mu_k) = k + 1$ which concludes the proof of Proposition 2.

Lemma 8. If $ADT_{k_0} = ADT_{k_0+1}$ for some $k_0 > 0$, then all ADT_k collapse from k_0 .

Proof. Let k > 0 be such that for all $\tau \in ADT_{k+1}$, we have that τ is ADT_k -definable. Given $\tau_{k+2} \in ADT_{k+2}$, we know that for each subtree of the form $\operatorname{CO}(\tau_1, \tau_2)$ in τ_{k+2} , we have that $\tau_2 \in ADT_{k+1}$. Therefore τ_2 is ADT_k -definable. Hence there exists $\tau'_2 \in ADT_k$ such that $\tau_2 \equiv \tau'_2$. By replacing τ_2 by τ'_2 in τ_{k+2} , we do not change its semantics. By applying this procedure over all CO operator of minimal depth (ie. CO operator having no CO in their ancestors), we obtain $\tau'_{k+1} \in ADT_{k+1}$ such that $\tau_{k+1} \equiv \tau'_{k+2}$. By hypothesis, we know that τ'_{k+1} is ADT_k -definable. This implies that τ_{k+1} is also ADT_k -definable. We can then extend this result for each l > k by induction.

E Complements of Section 6

Regarding the upper-bound complexity of ADT-MEMB (Table 1), we present here an alternating algorithm using the following logarithmic space (hence a PTIME complexity for ADT-MEMB) Algorithm 1. Remark that our algorithm can be extended to allow arbitrary ERES (with the Kleene star) as inputs, but this is out of the scope of the paper.

Algorithm 1 Memb (τ, t)

Input: τ an adt and t a trace **Output:** True if $t \in Traces(\tau)$, False otherwise. 1: switch (τ) 20: case $\tau = AND(\tau_1, ..., \tau_n)$: 2: case ϵ : 21:(\exists) guess $(i, a) \in \{1, ..., |t|\} \times$ $\{1, \dots, n\} \quad (\forall)$ guess test 3: return $(t = \varepsilon)$ \in $\{first, others\}$ 4: case $\tau = \gamma$: 5:**return** $(last(t) \models \gamma)$ 22:if test = first then 6: case $\tau = OP(\tau')$: 23:return Memb $(\tau_a, t_1...t_i)$ 7: return Memb (τ', t) 24:else 8: case $\tau = OR(\tau_1, ..., \tau_n)$: 25:return (\exists) guess $\tau' \in \{\tau_1, ..., \tau_n\}$ 9: 26: $Memb(AND(\tau_1, ..., \tau_{i-1}, \tau_{i+1}, ..., \tau_n), t)$ 10:return $Memb(\tau', t)$ 27:end if 28: case $\tau = co(\tau_1, \tau_2)$: 11: case $\tau = \text{SAND}(\tau_1, ..., \tau_n)$: 29: (\forall) guess $test \in \{first, second\}$ 12:(\exists) guess $i \in \{1, ..., |t|\}$ 13: (\forall) guess $test \in \{first, others\}$ 30: if test = first then if test = first then 31: return Memb (τ_1, t) 14: 15:return Memb $(\tau_1, t_1...t_i)$ 32: else 16:else 33: return $\neg Memb(\tau_2, t)$ $34 \cdot$ end if 17:return 18: $Memb(SAND(\tau_2, ..., \tau_n), t_{i+1}...t_{|t|})$ 35: end switch 19:end if

Proposition 4. Algorithm 1 solves ADT-MEMB in logarithmic space.

Proof. We start by showing that Algorithm 1 runs in logarithmic space, then we prove its correctness.

Since at each recursive call we only need to recall over which factor of t we are computing (in constant space) and over which part of τ (in constant space) we are pursuing the computation, Algorithm 1 runs in logarithmic space.

Regarding the correctness of Algorithm 1, we conduct a proof by induction over τ .

The cases where τ is a leaf (lines 1 to 7) are correct by definition. For the case where $\tau = OP(\tau')$ (line 9), then, Algorithm 1 is correct too since $Traces(\tau) = Traces(\tau')$.

If $\tau = OR(\tau_1, ..., \tau_n)$, then the case lines 13 to 16 are correct too since by definition, $t \in Traces(\tau)$ if and only if one can find $i \in \{1, ..., n\}$ such that $t \in Traces(att_i)$.

If $\tau = \text{SAND}(\tau_1, ..., \tau_n)$, then, by associativity, $\tau \equiv \text{SAND}(\tau_1, \text{SAND}(\tau_2, ..., \tau_n))$. Moreover, we have that $t = a_1 ... a_m \in Traces(\tau)$ if and only if one can find $i \in \{1, ..., m\}$ such that $a_1...a_i \in \tau_1$ and $a_{i+1}...a_m \in \text{SAND}(\tau_2, ..., \tau_n)$, which is what is done in lines 18-26.

If $\tau = \text{AND}(\tau_1, ..., \tau_n)$, then, by commutativity and associativity for each $1 \leq j \leq n$ we have $\tau \equiv \text{AND}(\tau_j, \text{AND}(\tau_1, ..., \tau_{j-1}, \tau_{j+1}, ..., \tau_n))$. Moreover, if a trace $t \in \text{AND}(\tau_1, ..., \tau_n)$, we can always choose j such that there is $1 \leq k \leq n$, with $k \neq j$ and $t \in Traces(\tau_k)$. Therefore t is in $Traces(\tau)$ if and only if t has a prefix in $Traces(\tau_j)$ and t is in semantics of $\text{AND}(\tau_1, ..., \tau_{j-1}, \tau_{j+1}, ..., \tau_n)$. Thus procedure describes from line 28 to 36 is correct.

Finally, if $\tau = co(\tau_1, \tau_2)n$ then trace $t \in Traces(\tau)$ if, and only if, $t \in Traces(\tau_1)$ and $t \notin Traces(\tau_2)$, which is what is done in lines 38-45.

Proposition 5. ADT_k-NE with $k \ge 6$ is not solvable in NSPACE $(g(k-5, c\sqrt{\frac{n-1}{3}}))$.

Proof. From [21, Theorem 4.29], for an ERE E of size n and of \sim -depth d, there exists a constant c such that the non-emptiness of $\sim E$ is not solvable in NSPACE $(g(d-3, c\sqrt{n}))$. Moreover, from the proof of Theorem 2, it can be shown that the non-emptiness of $\sim E$ reduces to answering ADT-NE for adt $\operatorname{CO}(\top, \tau_E) \in ADT_{d+2}$ of size at most 3n + 1, which concludes.

Proposition 6. For $k \ge 1$, ADT_k -NE is in (k+1)-EXPSPACE.

Proof. For an adt $\tau \in ADT_k$, we have a formula $\psi_{\tau} \in \Sigma_{k+1}$ with $|\psi_{\tau}| \in O(2^{|\tau|})$ (Lemma 1) that is equivalent to τ (see Lemma 1). Now, the non-emptiness of $Traces(\tau)$ is equivalent to the satisfiability of $\psi_{\tau} \in \Sigma_{k+1}$ which, by [12] take k-exponential time in the size of ψ_{τ} and therefore (k + 1)-exponential time in $|\tau|$.