Squeezing State Spaces of (Attack-Defence) Trees

Extended Abstract

Laure Petrucci*
LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord
Villetaneuse, France
laure.petrucci@lipn.univ-paris13.fr

Michał Knapik, Wojciech Penczek,
Teofil Sidoruk
ICS PAS, Warsaw, Poland
m.knapik,w.penczek,t.sidoruk@ipipan.waw.pl

ABSTRACT

In our earlier work, we presented a translation of attack-defence trees (ADTrees) to extended asynchronous multi-agent systems. We now introduce a general reduction scheme applicable to tree topologies, and in particular to ADTrees. It exploits the layered structure of a tree by avoiding unnecessary interleavings between nodes at different depths. We prove the soundness of this new method and show that it can be effectively used alongside existing techniques. **NOTE:** the paper previously published in Proc. of ICECCS'19 [29].

KEYWORDS

Attack-defence trees; multi-agent systems; state space reduction

1 INTRODUCTION

While the translation of ADTrees to multi-agent systems [4] already embedded reduced patterns resembling partial order reduction [14, 28, 32], they are not sufficient to combat state space explosion. In this paper, we define *layered reduction*, fully compatible with the previous scheme. We begin by recalling the formalism of ADTrees, then introduce Guarded Update Systems and identify key properties of tree synchronisation topologies. The layer-based reduction is defined next, followed by experimental evalutation and conclusions.

2 ATTACK-DEFENCE TREES

Attack-defence trees (ADTrees, [6, 19, 20]) extend the well-known formalism of attack trees [10, 18, 21, 25, 27, 30] to allow for representing security scenarios as an interplay between attacking and defending parties. The root nodes of ADTrees correspond to the main goal (e.g., steal treasure), their children to sub-goals (e.g., enter the vault undetected), and the leaves to specific actions (e.g., bribe guards). For ADTrees, *attributes* [6, 8, 10, 25] are numeric properties of nodes that allow for quantitative analyses (e.g., cost, time, or probability), and *conditions* are Boolean functions over attributes serving as constraints for node operations (e.g., the heist succeeds only if thieves get away within given time).

ADTrees remain extensively studied [9, 12, 22] and have been implemented in analysis frameworks based on, among others, Timed Automata [13, 26], Petri Nets [11], I/O-IMCs [5, 23], Bayesian Networks [15], and stochastic games [7, 16]. The parametric analysis of [3] is adapted to ADTrees using extended asynchronous semantics [4] from [17], allowing to *synthesise* values of attributes that yield the effectiveness of an attack/defence in a multi-agent setting.

3 GUARDED UPDATE SYSTEMS

We recall a compositional, executable semantics for ADTrees [4], in terms of their translation into networks of Guarded Update Systems ($\mathcal{GUS}$s), i.e. automata with variables and guarded transitions.

Definition 3.1 ($\mathcal{GUS}$). Let *Vars* be a finite set of integer variables. A $\mathcal{GUS}$ is a 4-tuple $\mathcal{M} = \langle \mathcal{S}, s^0, \rightarrow, Acts \rangle$, where:

(1) $\mathcal{S}$ is a finite set of states and $s^0 \in \mathcal{S}$ the initial state;
(2) $\rightarrow \subseteq \mathcal{S} \times Acts \times G \times U \times \mathcal{S}$ is a transition relation, where:
  (a) G is a set of *guards*, i.e. boolean formulae over atoms $t \sim 0$, s.t. t is a linear term over *Vars* and $\sim \in \{\leq, =, \geq\}$;
  (b) U is a set of *updates*, i.e. sets of assignments of type $v_j := f(v_0, \ldots, v_k)$, where $\forall_{0 \leq i \leq k}\, v_i \in Vars$, $v_j \in Vars$ and f is a function whose domain and codomain are compatible with the domains of its arguments and target; it is assumed that each variable is assigned at most once per update;
  (c) *Acts* is a finite set of action names;

Vals denotes the set of all functions $\omega\colon Vars \rightarrow \mathbb{N}$, i.e. valuations of *Vars*. By $u(\omega) \in Vals$ we denote the valuation s.t. for $v_j \in Vars$ we have $u(\omega)(v_j) = f(\omega(v_0), \ldots, \omega(v_k))$ if $f(v_0, \ldots, v_k) \in u$ and $u(\omega)(v_j) = \omega(v_j)$ otherwise. By $g(\omega)$ we mean the boolean value of the expression obtained after valuating the variables in g with ω. We denote $(s, act, g, u, s') \in \rightarrow$ by $s \xrightarrow[u]{g, act} s'$ and $acts(\mathcal{M})$ by *Acts*.

Definition 3.2 (Concrete Semantics of $\mathcal{GUS}$). Let $\mathcal{M}$ be a $\mathcal{GUS}$ and $\omega^0 \in Vals$ an initial valuation of *Vars*. By the *concrete semantics* of $\mathcal{M}$ over ω^0, we mean a tuple $\mathcal{CS}(\mathcal{M}, \omega^0) = \langle CS, w^0, \rightarrow \rangle$, where:

(1) $CS = \mathcal{S} \times Vals$ is the set of *concrete states*;
(2) $w^0 = (s^0, \omega^0)$;
(3) $\rightarrow \subseteq CS \times Acts \times CS$ is the transition relation s.t. $(s, \omega) \xrightarrow{act} (s', \omega')$ iff $s \xrightarrow[u]{g, act} s'$ where $g(\omega)$ is true and $\omega' = u(\omega)$, for some guard g and update u.

A *run* $\rho = t_0 act_0 t_1 act_1 \ldots$ is an infinite sequence of alternating concrete states and transitions s.t. for all $i \in \mathbb{N}$ we have $t_i \xrightarrow{act_i} t_{i+1}$. By $Runs(\mathcal{M}, t)$ we denote the set of all runs starting from $t \in CS$. We write $Runs(\mathcal{M})$ when the starting state is assumed to be initial.

Definition 3.3 (Asynchronous Product). For $i \in \{1..k\}$ let $\mathcal{M}_i = \langle \mathcal{S}_i, s_i^0, \rightarrow_i, Acts_i \rangle$ be a $\mathcal{GUS}$. The *asynchronous product* of $\mathcal{M}_i$ is the $\mathcal{GUS}$ $\mathcal{M}_1 || \ldots || \mathcal{M}_k = \langle \mathcal{S}_1 \times \ldots \times \mathcal{S}_k, (s_1^0, \ldots, s_k^0), \rightarrow, \bigcup Acts_i \rangle$

*L. Petrucci acknowledges the support the PICS CNRS/PAN project PARTIES.
T. Sidoruk is also affiliated with Faculty of Mathematics and Information Science, Warsaw University of Technology.

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 2020, Auckland, New Zealand

https://doi.org/doi

with the transition rule defined in the usual way, i.e., the component transitions labeled with the same action are synchronized while the component transitions labeled with actions that are not in the alphabets of the other components occur on their own [29].

Definition 3.4 (Synchronisation Topology). The *synchronisation topology* induced by a $\mathcal{GUS}$ $\mathcal{G} = ||_{i=0}^{n}\mathcal{M}_i$ is the undirected graph $\mathcal{SG}(\mathcal{G}) = \langle\{\mathcal{M}_i \mid i = 0 \ldots n\}, \mathcal{E}\rangle$, where $(\mathcal{M}_i, \mathcal{M}_j) \in \mathcal{E}$ iff $i \neq j$ and $Acts_i \cap Acts_j \neq \emptyset$.

Given an ADTree $\mathcal{T}$ with a set of nodes $\{A_i \mid i = 0 \ldots n\}$, a $\mathcal{GUS}$ M_i is associated with each A_i. The associated topology $\mathcal{SG}(\mathcal{T})$ is defined by replacing each node A_i with M_i. The attributes and conditions of A_i are modelled by variables and guards in $\mathcal{M}_i$. Variables are updated during the synchronisation between a child and its parent node, mimicking how the values of the attributes are actually used. Note that the topology induced by $\mathcal{T}$ is a tree.

4 PROPERTIES OF TREE TOPOLOGIES

We now consider $\mathcal{GUS}$ $\mathcal{G} = ||_{i=0}^{n}\mathcal{M}_i$ with a tree synchronisation topology. Under certain assumptions, it can be exploited to obtain a reachability-preserving reduction of the system. We now give the notions necessary to define these assumptions.

Let $\mathcal{M}_N, \mathcal{M}_C$ be a node and one of its children, respectively. If along each run $\rho \in Runs(\mathcal{G})$ after executing an action $act \in Acts_N \setminus Acts_C$ no action from $Acts_N \cap Acts_C$ appears, then $\mathcal{M}_C$ *precedes* $\mathcal{M}_N$ (denoted by $\mathcal{M}_C \hookrightarrow \mathcal{M}_N$).

A synchronisation tree $\mathcal{SG}(\mathcal{G})$ is *root-directed* if for each node $\mathcal{M}_N$ and any of its children $\mathcal{M}_C$ we have $\mathcal{M}_C \hookrightarrow \mathcal{M}_N$.

A root-directed synchronisation tree $\mathcal{SG}(\mathcal{G})$ is *update-separable* if for each $v \in Vars$:

(1) v is updated in at most one component $\mathcal{M}_v$;
(2) v is tested only in guards of the ancestors of $\mathcal{M}_v$ in $\mathcal{SG}(\mathcal{G})$.

In what follows, we will manipulate subtrees of tree topologies.

The subtree of $\mathcal{SG}(\mathcal{G})$ rooted in $\mathcal{M}_i$ is the tree $\Downarrow\mathcal{M}_i$ containing $\mathcal{M}_i$ and all its descendants. Let $\rho = t_0 act_0 t_1 act_1 \ldots$ be a run in $Runs(\mathcal{G})$ and $\mathcal{M}_i$ be a node of $\mathcal{SG}(\mathcal{G})$. The projection of ρ on $\Downarrow\mathcal{M}_i$, denoted by $\rho_{\Downarrow\mathcal{M}_i}$, is obtained by:

(1) retaining in each concrete state $t_j, j \in \mathbb{N}$ only its projection (states and variables) on $\Downarrow\mathcal{M}_i$;
(2) keeping only the transitions in the nodes of $\Downarrow\mathcal{M}_i$.

Any actions not in the subtree are safely removed from the projected run, as their projected source and target states are identical.

Lemma 4.1. *Let $\mathcal{SG}(\mathcal{G})$ be a root-directed, update-separable tree. Let $\mathcal{M}_i$ be a node and $\rho \in Runs(\mathcal{G})$. Then, $\rho_{\Downarrow\mathcal{M}_i}$ is a prefix of some run $\rho' \in Runs(\Downarrow\mathcal{M}_i)$.*

Lemma 4.2. *Synchronisation topologies of ADTrees are root-directed and update-separable.*

5 LAYERED REDUCTION FOR TREES

In the layered reduction, we consider specifically the last synchronisation of node $\mathcal{M}_N$ with one of its children $\mathcal{M}_C$, before any other action in $\mathcal{M}_N$. Let $\#_{child}(\mathcal{M}_N)$ be the number of children of $\mathcal{M}_N$.

Definition 5.1 (Last synchronisations with children). By the last synchronisations of M_N with its children we mean the transitions (denoted by *lst*), that are synchronising transitions of M_N and one of its children $\mathcal{M}_{C_j}$, such that there are states s_i, s_{i+1}, s_{i+2} and another transition t of M_N which does not synchronise with any transition of its children $\mathcal{M}_{C_j}$, with $s_i \ lst \ s_{i+1} \ t \ s_{i+2}$. The set of these transitions is denoted by $Lst_C(\mathcal{M}_N)$.

Dealing with a single depth Let us fix a depth $d > 0$ of the tree $\mathcal{SG}(\mathcal{G})$. We add a fresh variable v_d, initialised with 0, that counts the total number of synchronisations between the nodes at depth d and the nodes at depth $d + 1$. We modify each node $\mathcal{M}_N$ at depth d by adding to the update u of any transition in $Lst_C(\mathcal{M}_N)$ a new element $v_d := v_d + \#_{child}(\mathcal{M}_N)$. It is a way for node $\mathcal{M}_N$ to notify it has performed all synchronisations with its children. The total number of the children of the nodes at depth d is $\#_{child}(d) = \#_{child}(\mathcal{M}_{N_1}) + \ldots + \#_{child}(\mathcal{M}_{N_k})$ where $\{\mathcal{M}_{N_1}, \ldots, \mathcal{M}_{N_k}\}$ are all the nodes at depth d in $\mathcal{SG}(\mathcal{G})$.

In the next step of the construction we also modify each node $\mathcal{M}_N$ at depth d by extending the guard g of each transition t of M_N which does not synchronise with any transition of the children of M_N to $g \wedge (v_d = \#_{child}(d))$. This prevents any action at depth d to occur before all synchronisations with the children are finished.

Dealing with the entire tree In order to obtain the layered reduction of $\mathcal{SG}(\mathcal{G})$, denoted by $\mathcal{SG}^{lr}(\mathcal{G})$, the above transformation is performed for each depth $0 < d <$ height of $\mathcal{SG}(\mathcal{G})$.

Proposition 5.2. *Let $\mathcal{SG}(\mathcal{G})$ be a root-directed, update-separable tree. A good (bad) final state s is reachable in $\mathcal{SG}(\mathcal{G})$ iff it is reachable in $\mathcal{SG}^{lr}(\mathcal{G})$.*

6 EXPERIMENTS

Experimental evaluation, performed on a 2.7 GHz Intel Core i7, with 16 GB of memory, using IMITATOR (www.imitator.fr, [2]), a model-checker for Parametric Timed Automata [1], was based on case studies modelling real-world security scenarios [4, 10, 24, 25, 31, 33] as well as a scalable, generated ADTree[1]. The effectiveness of layered reduction, especially when applied in conjunction with our previous pattern-based scheme, is clear. It significantly improves the results achieved by the latter alone, leading to state space reduction of well over 90% in many cases. We refer the reader to [29] for extensive experimental results and discussion.

7 CONCLUSION

We defined Guarded Update Systems, which are automata manipulating variables and proposed to exploit the characteristics of those that exhibit a tree structure to further contain state space explosion by avoiding some unnecessary interleavings. This approach directly applies to ADTrees, can be combined with our previous pattern-based reduction, and extensive experiments proved its efficiency. We believe that the approach applies not only to tree topologies but also to directed acyclic graphs (DAGs), and thus to models from other application domains such as workflows.

This layered reduction opens several avenues of further research e.g., compositional analysis and parallel model checking of systems with tree topologies, as well as reasoning about the assignment of agents to ADTree nodes to analyse possible coalitions in attacks and/or defences.

[1] See http://lipn.univ-paris13.fr/~petrucci/ICECCS19_imitator_files.zip.

REFERENCES

[1] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. 1993. Parametric real-time reasoning. In *ACM Symposium on Theory of Computing, 1993*. ACM, 592–601. https://doi.org/10.1145/167088.167242

[2] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. 2012. IMITATOR 2.5: A Tool for Analyzing Robustness in Scheduling Problems. In *FM 2012 (LNCS)*, Vol. 7436. Springer, 33–36. https://doi.org/10.1007/978-3-642-32759-9_6

[3] Étienne André, Didier Lime, Mathias Ramparison, and Mariëlle Stoelinga. 2019. Parametric analyses of attack-fault trees. In *Proc. 19th Int. Conf. on Application of Concurrency to System Design (ACSD'19), Aachen, Germany*. IEEE. To appear.

[4] Jaime Arias, Carlos Budde, Wojciech Penczek, Laure Petrucci, and Mariëlle Stoelinga. 2019. Hackers vs. Security: Attack-Defence Trees as Asynchronous Multi-Agent Systems. https://arxiv.org/abs/1906.05283.

[5] Florian Arnold, Dennis Guck, Rajesh Kumar, and Mariëlle Stoelinga. 2015. Sequential and Parallel Attack Tree Modelling. In *Computer Safety, Reliability, and Security*. Springer International Publishing, 291–299. https://doi.org/10.1007/978-3-319-24249-1_25

[6] Zaruhi Aslanyan and Flemming Nielson. 2015. Pareto Efficient Solutions of Attack-Defence Trees. In *Principles of Security and Trust*. Vol. 9036. Springer Berlin Heidelberg, 95–114. https://doi.org/10.1007/978-3-662-46666-7_6

[7] Zaruhi Aslanyan, Flemming Nielson, and David Parker. 2016. Quantitative Verification and Synthesis of Attack-Defence Scenarios. In *CSF 2016*. IEEE, 105–119. https://doi.org/10.1109/CSF.2016.15

[8] Alessandra Bagnato, Barbara Kordy, Per Håkon Meland, and Patrick Schweitzer. 2012. Attribute Decoration of Attack-Defense Trees. *IJSSE* 3, 2 (2012), 1–35. https://doi.org/10.4018/jsse.2012040101

[9] Angèle Bossuat and Barbara Kordy. 2017. Evil Twins: Handling Repetitions in Attack-Defense Trees - A Survival Guide. In *Graphical Models for Security - 4th International Workshop, GraMSec 2017, Santa Barbara, CA, USA, August 21, 2017, Revised Selected Papers (Lecture Notes in Computer Science)*, Peng Liu, Sjouke Mauw, and Ketil Stølen (Eds.), Vol. 10744. Springer, 17–37. https://doi.org/10.1007/978-3-319-74860-3_2

[10] Ahto Buldas, Peeter Laud, Jaan Priisalu, Märt Saarepera, and Jan Willemson. 2006. Rational Choice of Security Measures Via Multi-parameter Attack Trees. In *Critical Information Infrastructures Security*. Springer, 235–248.

[11] Dalton, Mills, Colombi, and Raines. 2006. Analyzing Attack Trees using Generalized Stochastic Petri Nets. In *2006 IEEE Information Assurance Workshop*. 116–123. https://doi.org/10.1109/IAW.2006.1652085

[12] Barbara Fila and Wojciech Widel. 2019. Efficient Attack-Defense Tree Analysis using Pareto Attribute Domains. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 200–215. https://doi.org/10.1109/CSF.2019.00021

[13] Olga Gadyatskaya, René Rydhof Hansen, Kim Guldstrand Larsen, Axel Legay, Mads Chr. Olesen, and Danny Bøgsted Poulsen. 2016. Modelling Attack-defense Trees Using Timed Automata. In *Formal Modeling and Analysis of Timed Systems*, Martin Fränzle and Nicolas Markey (Eds.). Vol. 9884. Springer International Publishing, 35–50. https://doi.org/10.1007/978-3-319-44878-7_3

[14] Patrice Godefroid. 1996. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem.* Springer-Verlag, Berlin, Heidelberg.

[15] Marco Gribaudo, Mauro Iacono, and Stefano Marrone. 2015. Exploiting Bayesian Networks for the Analysis of Combined Attack Trees. *Electronic Notes in Theoretical Computer Science* 310 (2015), 91–111. https://doi.org/10.1016/j.entcs.2014.12.014

[16] Holger Hermanns, Julia Krämer, Jan Krčál, and Mariëlle Stoelinga. 2016. The Value of Attack-Defence Diagrams. In *POST 2016 (LNCS)*, Vol. 9635. Springer, 163–185.

[17] Wojciech Jamroga, Wojciech Penczek, Piotr Dembinski, and Antoni W. Mazurkiewicz. 2018. Towards Partial Order Reductions for Strategic Ability. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar (Eds.). International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 156–165. http://dl.acm.org/citation.cfm?id=3237413

[18] Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Rolando Trujillo-Rasua. 2015. Attack Trees with Sequential Conjunction. In *ICT Systems Security and Privacy Protection*. Springer, 339–353. https://doi.org/10.1007/978-3-319-18467-8_23

[19] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patric Schweitzer. 2011. Foundations of Attack-Defense Trees. In *FAST 2010 (LNCS)*, Vol. 6561. Springer, 80–95.

[20] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. 2014. Attack–defense trees. *Journal of Logic and Computation* 24, 1 (2014), 55–87. https://doi.org/10.1093/logcom/exs029

[21] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. 2014. DAG-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer Science Review* 13-14 (2014), 1–38. https://doi.org/10.1016/j.cosrev.2014.07.001

[22] Barbara Kordy and Wojciech Widel. 2018. On Quantitative Analysis of Attack-Defense Trees with Repeated Labels. In *Principles of Security and Trust - 7th International Conference, POST 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings (Lecture Notes in Computer Science)*, Lujo Bauer and Ralf Küsters (Eds.), Vol. 10804. Springer, 325–346. https://doi.org/10.1007/978-3-319-89722-6_14

[23] Rajesh Kumar, Dennis Guck, and Mariëlle Stoelinga. 2015. Time Dependent Analysis with Dynamic Counter Measure Trees. *CoRR* abs/1510.00050 (2015). arXiv:1510.00050 http://arxiv.org/abs/1510.00050

[24] Rajseh Kumar, Enno Ruijters, and Mariëlle Stoelinga. 2015. Quantitative Attack Tree Analysis via Priced Timed Automata. In *FORMATS 2015 (LNCS)*, Vol. 9268. Springer, 156–171.

[25] Rajesh Kumar, Stefano Schivo, Enno Ruijters, Buğra Mehmet Yildiz, David Huistra, Jacco Brandt, Arend Rensink, and Mariëlle Stoelinga. 2018. Effective Analysis of Attack Trees: A Model-Driven Approach. In *Fundamental Approaches to Software Engineering*. Springer, 56–73.

[26] Rajesh Kumar and Mariëlle Stoelinga. 2017. Quantitative Security and Safety Analysis with Attack-Fault Trees. In *18th IEEE International Symposium on High Assurance Systems Engineering, HASE 2017, Singapore, January 12-14, 2017*. IEEE Computer Society, 25–32. https://doi.org/10.1109/HASE.2017.12

[27] Sjouke Mauw and Martijn Oostdijk. 2006. Foundations of Attack Trees. In *ICISC 2005*. Springer, 186–198.

[28] Doron Peled. 1993. All from one, one for all: on model checking using representatives. In *Computer Aided Verification*, Costas Courcoubetis (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 409–423.

[29] Laure Petrucci, Michal Knapik, Wojciech Penczek, and Teofil Sidoruk. 2019. Squeezing State Spaces of (Attack-Defence) Trees. In *24th International Conference on Engineering of Complex Computer Systems, ICECCS 2019, Guangzhou, China, November 10-13, 2019*, Jun Pang and Jing Sun (Eds.). IEEE, 71–80. https://doi.org/10.1109/ICECCS.2019.00015

[30] C. Salter, O.S. Saydjari, B. Schneier, and J. Wallner. 1998. Toward a Secure System Engineering Methodolgy. In *NSPW'98*. ACM, 2–10. https://doi.org/10.1145/310889.310900

[31] Max Steiner and Peter Liggesmeyer. 2015. Qualitative and Quantitative Analysis of CFTs Taking Security Causes into Account. In *Computer Safety, Reliability, and Security*. Springer, 109–120.

[32] Antti Valmari. 1991. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990*, Grzegorz Rozenberg (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 491–515.

[33] J. D. Weiss. 1991. A System Security Engineering Process. In *Proceedings of the 14th National Computer Security Conference*. 572–581.