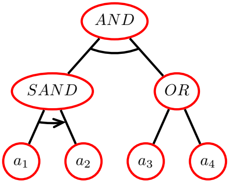
Deciding the Non-Emptiness of Attack trees

Maxime Audinot, Sophie Pinchinat, François Schwarzentruber, Florence Wacheux

Univ. Rennes, IRISA, CNRS

GraMSec 2018

Non-Emptiness of attack trees: Relevance



Attack trees always seem non-empty:

$a_1 a_3 a_2$ is an attack

But is the attack $a_1 a_3 a_2$ realizable?

Maybe executing action a_1 consumes the resource required to execute a_3.

We need a model $\mathcal{S}$ of the system.

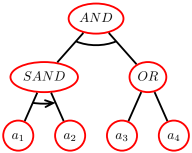
Non-emptiness of an attack tree w.r.t. a system $\mathcal{S}$

Is there an attack described by the tree that is realizable in $\mathcal{S}$?

Outline

Attack trees

Action-based trees (classical)

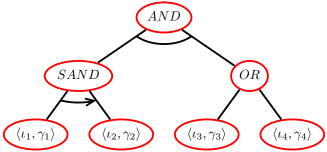


State-based trees

- more recent
- homogeneous formalism
- a system-based semantics [APK17]

Definition (Attack tree)

$\tau ::= \langle \iota, \gamma \rangle \mid \mathtt{OR}(\tau,\tau) \mid \mathtt{SAND}(\tau,\tau) \mid \mathtt{AND}(\tau,\tau) \mid \mathtt{AND}(\tau,\tau,\tau) \mid \mathtt{AND}(\tau,\tau,\tau,\tau) \mid \ldots$
where $\iota, \gamma \in$ *Prop*.



Model of the system

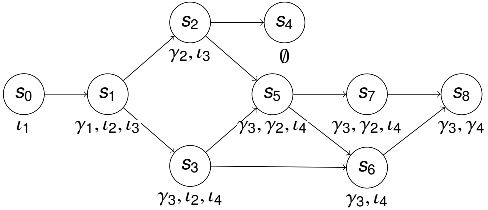
Definition (Labeled transition system)

A *labeled transition system* over *Prop* is a tuple $\mathcal{S} = (S, \rightarrow, \lambda)$, where

- S finite set of *states*
- $\rightarrow \subseteq S \times S$ *transition relation*
- $\lambda : S \rightarrow 2^{Prop}$ *labeling* function

Write $s \models p$ whenever $p \in \lambda(s)$.

$Prop = \{\iota_1, \iota_2, \iota_3, \iota_4, \gamma_1, \gamma_2, \gamma_3, \gamma_4\}$



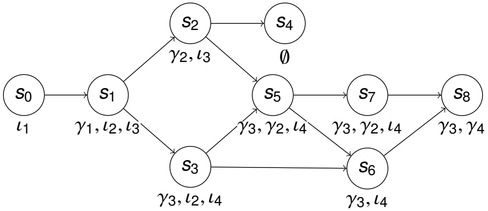
$\lambda(s_0) = \{\iota_1\}$, $\lambda(s_1) = \{\gamma_1, \iota_2, \iota_3\}$, $\lambda(s_2) = \{\gamma_2, \iota_3\}$ ...

Paths in a label transition system

Definition (Paths)

A *path* in $\mathcal{S}$ is a sequence $\pi = s_0 \dots s_n$ of consecutive states in $\mathcal{S}$.
$\pi.first := s_0$ and $\pi.last := s_n$

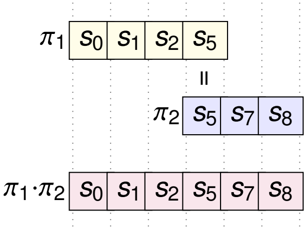
Write Paths($\mathcal{S}$) for the set of paths in $\mathcal{S}$.



$\pi = s_0 s_1 s_3 s_5 s_6 s_8$
$\pi.first = s_0$
$\pi.last = s_8$

Paths concatenation

Concatenation of paths π_1 and π_2 possible whenever $\pi_1.\mathit{last} = \pi_2.\mathit{first}$

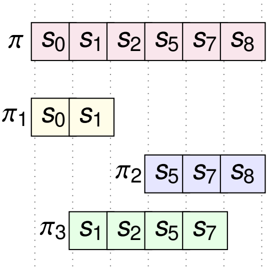


Notation

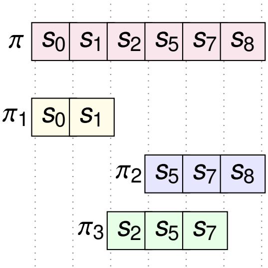
For $\Pi_1, \Pi_2 \subseteq \mathrm{Paths}(\mathcal{S})$, define $\Pi_1 \cdot \Pi_2 := \{\pi_1 \cdot \pi_2 \mid \pi_1 \in \Pi_1 \text{ and } \pi_2 \in \Pi_2\}$.

Path parallel composition

Informal



(a) A parallel composition.



(b) Not a parallel composition.

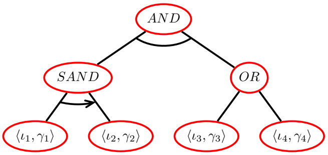
Notation

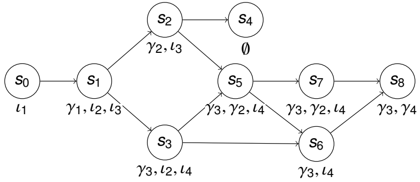
For $\Pi_1, \ldots, \Pi_n \subseteq \text{Paths}(\mathcal{S})$, define $\otimes_n(\Pi_1, \Pi_2 \ldots, \Pi_n)$ by $\{\pi \mid \pi \text{ is a parallel composition of some } \pi_1 \in \Pi_1, \ldots, \pi_n \in \Pi_n\}$.

Path semantics of an attack tree [APK17]

Definition ($[\![\tau]\!]^{\mathcal{S}} \subseteq \mathrm{Paths}(\mathcal{S})$)

- $[\![\langle\iota,\gamma\rangle]\!]^{\mathcal{S}} = \{\pi \in \mathrm{Paths}(\mathcal{S}) \mid \pi.\mathit{first} \models \iota \text{ and } \pi.\mathit{last} \models \gamma\}$
- $[\![\mathtt{OR}(\tau_1,\tau_2)]\!]^{\mathcal{S}} = [\![\tau_1]\!]^{\mathcal{S}} \cup [\![\tau_2]\!]^{\mathcal{S}}$
- $[\![\mathtt{SAND}(\tau_1,\tau_2)]\!]^{\mathcal{S}} = [\![\tau_1]\!]^{\mathcal{S}} . [\![\tau_2]\!]^{\mathcal{S}}$
- $[\![\mathtt{AND}(\tau_1,\ldots,\tau_n)]\!]^{\mathcal{S}} = \otimes_n([\![\tau_1]\!]^{\mathcal{S}},\ldots,[\![\tau_n]\!]^{\mathcal{S}})$





$$[\![\tau]\!]^{\mathcal{S}} = \{s_0 s_1 s_2 s_5,\ s_0 s_1 s_2 s_5,\ s_0 s_1 s_3 s_6 s_8, \ldots\}$$

Outline

The decision problem NON-EMPTINESS

Definition (NON-EMPTINESS)

Input: a system $\mathcal{S}$ and an attack tree τ.
Output: $[\![\tau]\!]^{\mathcal{S}} \neq \emptyset$?

Theorem

NON-EMPTINESS *is* NP*-complete.*

Proof:

- NON-EMPTINESS is NP-easy:
  - compute an abstraction of the path semantics
  - guess and check a paths corresponding with the abstract semantics
- NON-EMPTINESS is NP-hard: Reduction of SAT (NP-complete by [Coo71])

Non-emptiness is NP-easy

Abstracting the path semantics

Recall the path semantics:

$$[\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}} = \{\pi \in \mathsf{Paths}(\mathcal{S}) \mid \pi.\textit{first} \models \iota \text{ and } \pi.\textit{last} \models \gamma\}$$

The path semantics is not adequate for “computation”, as any cycle in $\mathcal{S}$ yields infinitely many paths.

The *abstract semantics* retains only end-states of paths in $[\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}}$.

$$[\![\langle \iota, \gamma \rangle]\!]^{\mathcal{S}}_{\mathsf{abs}} = \{s_1 s_2 \mid s_1 \models \iota \text{ and } s_2 \models \gamma\}$$

NON-EMPTINESS is NP-easy

Abstract semantics

Definition ($[\![\tau]\!]^{\mathcal{S}}_{\text{abs}} \subseteq S^*$)

- $[\![\langle\iota, \gamma\rangle]\!]^{\mathcal{S}}_{\text{abs}} = \{s_1 s_2 \mid s_1 \models \iota \text{ and } s_2 \models \gamma\}$
- $[\![\mathtt{OR}(\tau_1, \tau_2)]\!]^{\mathcal{S}}_{\text{abs}} = [\![\tau_1]\!]^{\mathcal{S}}_{\text{abs}} \cup [\![\tau_2]\!]^{\mathcal{S}}_{\text{abs}}$
- $[\![\mathtt{SAND}(\tau_1, \tau_2)]\!]^{\mathcal{S}}_{\text{abs}} = [\![\tau_1]\!]^{\mathcal{S}}_{\text{abs}} \cdot [\![\tau_2]\!]^{\mathcal{S}}_{\text{abs}}$
- How about AND?

Vocabulary

Call *word* any sequence $u \in S^*$ (that may not be a path).

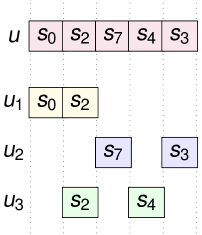
NON-EMPTINESS is NP-easy

Abstract semantics for AND

Definition (Shuffle)

u shuffles $u_1, \ldots, u_n$ whenever:

- (Linearization) u is composed of all the states occurring in $u_1, \ldots, u_n$ with preserved precedence.
- (Covering) Every sequence of two consecutive states in u is between the occurrence of $u_j.first$ and $u_j.last$ for some j.



Definition

$$[\![\mathtt{AND}(\tau_1, \ldots, \tau_n)]\!]^{\mathcal{S}}_{\text{abs}} = \{u \mid u \text{ shuffles some } u_1 \in [\![\tau_1]\!]^{\mathcal{S}}_{\text{abs}}, \ldots, u_n \in [\![\tau_n]\!]^{\mathcal{S}}_{\text{abs}}\}.$$

Algorithm for Non-emptiness

Guess $u \in [\![\tau]\!]^{\mathcal{S}}_{\text{abs}}$

```
Input: An attack tree τ and a transition system S
Output: A word u ∈ [[τ]]^S_abs
switch τ do
    case ⟨ι, γ⟩ do
        guess s_1, s_2 ∈ S;
        check ι ∈ λ(s_1) and γ ∈ λ(s_2);
        return s_1 s_2;
    end
    case OR(τ_1, τ_2) do
        guess i ∈ {1, 2};
        return guessAbstractPath(τ_i, S);
    end
    case SAND(τ_1, τ_2) do
        u_1 := guessAbstractPath(τ_1, S);
        u_2 := guessAbstractPath(τ_2, S);
        check u_1.last = u_2.first;
        return u_1·u_2
    end
    case AND(τ_1, ..., τ_n) do
        u_i := guessAbstractPath(τ_i, S) for each 1 ≤ i ≤ n;
        guess u, a linearization of u_1, ..., u_n;
        forall letters s of u except u.first and u.last do
            check there exists j, k ∈ [1, n] such that either s is strictly between u_j.first
              and u_j.last in u, or s equals both u_j.first and u_k.last
        end
        return u;
    end
end
```

Algorithm 1: guessAbstractPath($\tau, \mathcal{S}$).

Algorithm for Non-emptiness

Check that *u* can instanciate some path

Input: An attack tree τ and a transition system $\mathcal{S}$
Output: Accept whenever $[\![\tau]\!]^{\mathcal{S}} \neq \emptyset$.

```
u := guessAbstractPath(τ, S);
foreach s1, s2 successive in u do
  | check reach_S(s1, s2)
end
accept
```

Algorithm 2: emptiness($\tau, \mathcal{S}$).

Theorem

Non-emptiness *is* NP*-complete.*

Non-emptiness is NP-hard

NP-hardness arise already for very simple attack trees of the form $\texttt{AND}(\langle \iota_1, \gamma_1 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)$:

Proposition (From [APK17])

Given a system $\mathcal{S}$ and $\iota_1, \gamma_1, \ldots \iota_n, \gamma_n \in Prop$, it is NP*-hard to decide* $[\![AND(\langle \iota_1, \gamma_1 \rangle, \ldots, \langle \iota_n, \gamma_n \rangle)]\!]^{\mathcal{S}} \neq \emptyset$.

Proof: By reduction of SAT (NP-complete by [Coo71]).

SAT problem

Input: $C_1, \ldots, C_k$ clauses over Boolean variables $p, q, r, \ldots$
Output: is there a valuation of $p, q, r, \ldots$ that satisfies $C_1, \ldots, C_k$?

We define a polynomial translation from SAT inputs to Non-emptiness inputs.

NP-hardness of $[\![\texttt{AND}(\langle\iota_1,\gamma_1\rangle,\ldots\langle\iota_n,\gamma_n\rangle)]\!]^{\mathcal{S}} \neq \emptyset$?

SAT problem

Input: $C_1,\ldots,C_k$ clauses over Boolean variables $p,q,r,\ldots$
Output: is there a valuation of $p,q,r,\ldots$ that satisfies $C_1,\ldots,C_k$?

Reduction of SAT: From input $C_1,\ldots,C_k$ over $p,q,r,\ldots$ of SAT, define system $\mathcal{S}$ (of polynomial size) over $Prop = \{\iota, C_1,\ldots,C_k\}$ s.t.

$$C_1,\ldots,C_k \in \texttt{SAT} \text{ iff } [\![\texttt{AND}(\langle\iota,C_1\rangle,\ldots\langle\iota,C_k\rangle)]\!]^{\mathcal{S}} \neq \emptyset$$

$C_1 = p \vee \bar{r}$
$C_2 = \bar{q} \vee r$

$[\![\texttt{AND}(\langle\iota,C_1\rangle,\langle\iota,C_2\rangle)]\!]^{\mathcal{S}} = \{pqr, p\bar{q},\ldots\}$

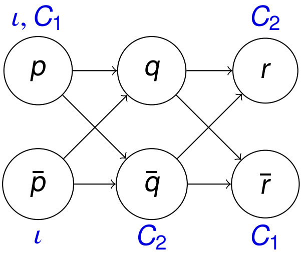
NP-hardness of $[\![\mathtt{AND}(\langle \iota_1, \gamma_1 \rangle, \ldots \langle \iota_n, \gamma_n \rangle)]\!]^{\mathcal{S}} \neq \emptyset$?

SAT problem

Input: $C_1, \ldots, C_k$ clauses over Boolean variables $p, q, r, \ldots$
Output: is there a valuation of $p, q, r, \ldots$ that satisfies $C_1, \ldots, C_k$?

Reduction of SAT: From input $C_1, \ldots, C_k$ over $p, q, r, \ldots$ of SAT, define system $\mathcal{S}$ (of polynomial size) over $Prop = \{\iota, C_1, \ldots, C_k\}$ s.t.

$$C_1, \ldots, C_k \in \mathtt{SAT} \text{ iff } [\![\mathtt{AND}(\langle \iota, C_1 \rangle, \ldots \langle \iota, C_k \rangle)]\!]^{\mathcal{S}} \neq \emptyset$$



$C_1 = p \vee \bar{r}$
$C_2 = \bar{q} \vee r$

$[\![\mathtt{AND}(\langle \iota, C_1 \rangle, \langle \iota, C_2 \rangle)]\!]^{\mathcal{S}} = \{pqr, p\bar{q}, \ldots\}$

Outline

The sub-polynomial AND-free case

Definition (NON-EMPTINESS$_{Af}$)

Input: a system $\mathcal{S}$ and an AND-free attack tree τ
Output: $[\![\tau]\!]^{\mathcal{S}} \neq \emptyset$?

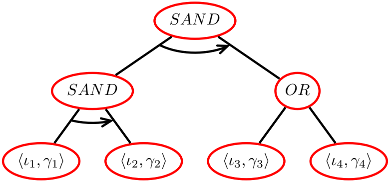
Theorem

NON-EMPTINESS$_{Af}$ *is* NLOGSPACE-*complete.*

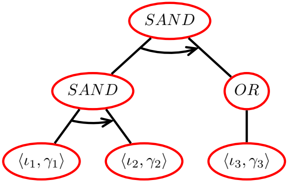
Proof:

- NON-EMPTINESS$_{Af}$ is NLOGSPACE-hard.
  Trivial logspace reduction from the s–t-connectivity in a graph (NLOGSPACE-complete by [Jon75]) to the non-emptiness of a leaf attack tree $\langle \iota, \gamma \rangle$.
- NON-EMPTINESS$_{Af}$ is NLOGSPACE-easy

NON-EMPTINESS$_{Af}$ is NLOGSPACE-easy



τ



An OR-resolution of τ

Proposition

$\pi \in [\![\tau]\!]^{\mathcal{S}}$ iff there exists an OR-resolution τ' of τ s.t. $\pi \in [\![\tau']\!]^{\mathcal{S}}$

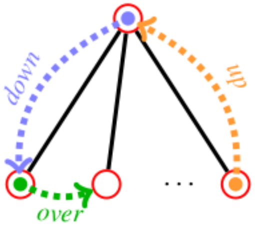
Guess simultaneously τ' and $\pi \in [\![\tau']\!]^{\mathcal{S}}$ during a depth-first search of τ.
Require to store (logarithmic memory):

- 1 node of τ
- 2 states of $\mathcal{S}$

Algorithm for NON-EMPTINESS$_{Af}$

Input: An AND-free attack tree τ and a transition system $\mathcal{S}$
Output: Accept whenever $[\![\tau]\!]^{\mathcal{S}} \neq \emptyset$.
guess $s \in S$;
node := root of τ;
lastOp := *down*;
repeat
  if *node* = $\langle \iota, \gamma \rangle$ **then**
    check $s \models \iota$;
    loop
      guess whether we break the loop or not; if yes, **break** the loop;
      guess $s' \in S$ with $s \to s'$;
      $s := s'$
    endLoop
    check $s \models \gamma$;
  end
  if *(lastOp* = down*) or (lastOp* = over*)* **then**
    Try to perform and update *node* with operation *down*, *over*, *up* in priority;
    Store in *lastOp* the last performed operation
  else
    Try to perform and update *node* with operation *over*, *up* in priority;
    Store in *lastOp* the last performed operation
  end
until *(node = root of* τ*) and (lastOp* = up*)*;
accept

Algorithm 5: emptinessNL$_{ANDfree}$(τ,$\mathcal{S}$).



Outline

Conclusion

- Achievements
  - Deciding $[\![\tau]\!]^{\mathcal{S}} \neq \emptyset$ is NP-complete.
  - Deciding $[\![\tau]\!]^{\mathcal{S}} \neq \emptyset$ for an `AND`-free attack tree is NLOGSPACE-complete.
  - `AND` is a really complex operator!
- Future work
  - Non-emptiness of action-based attack trees.
    Our results should still hold.
  - Other decision problems, e.g.

$$[\![\tau_1]\!]^{\mathcal{S}} = [\![\tau_2]\!]^{\mathcal{S}}?$$

Maxime Audinot, Sophie Pinchinat, and Barbara Kordy.
Is my attack tree correct?
In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security – ESORICS 2017*, pages 83–102, Cham, 2017. Springer International Publishing.

Stephen A Cook.
The complexity of theorem-proving procedures.
In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

Neil D Jones.
Space-bounded reducibility among combinatorial problems.
Journal of Computer and System Sciences, 11(1):68–85, 1975.