Attack-Defense Trees

Kordy, Mauw, Radomirovic, Schweitzer - University Luxembourg

Benjamin Çoban

Wilhelm-Schickard-Institut für Informatik
Universität Tübingen, Germany

Introduction

- Security of a system is not static

Introduction

- Security of a system is not static
- It is undecidable whether or not a system is "secure"(Rice's Theorem)

Introduction

- Security of a system is not static
- It is undecidable whether or not a system is “secure“(Rice’s Theorem)

25.11.2019 | 21:25 Uhr | Update

Mit der Axt ins Grüne Gewölbe - Soko "Epaulette" aufgestockt

Nach dem Einbruch im Grünen Gewölbe in Dresden laufen die Ermittlungen auf Hochtouren. Von den Tätern fehlt bisher jede Spur. Jetzt wurde ein Ausschnitt eines Überwachungsvideos veröffentlicht. Eine Sonderkommission ermittelt. Der Schock ist groß. Über den Schaden gibt es noch keine genauen Informationen.

https://www.mdr.de/sachsen/dresden/dresden-radebeul/kunstraub-kunstsammlungen-dresden-100.html, 02.01.2020

Introduction

- Security of a system is not static
- It is undecidable whether or not a system is “secure“(Rice’s Theorem)

25.11.2019 | 21:25 Uhr | Update

Mit der Axt ins Grüne Gewölbe - Soko "Epaulette" aufgestockt

Nach dem Einbruch im Grünen Gewölbe in Dresden laufen die Ermittlungen auf Hochtouren. Von den Tätern fehlt bisher jede Spur. Jetzt wurde ein Ausschnitt eines Überwachungsvideos veröffentlicht. Eine Sonderkommission ermittelt. Der Schock ist groß. Über den Schaden gibt es noch keine genauen Informationen.

https://www.mdr.de/sachsen/dresden/dresden-radebeul/kunstraub-kunstsammlungen-dresden-100.html, 02.01.2020

- More complex security measures are followed by more sophisticated attacks - an endless race

Challenges

- What are the best defensive measures to invest in?

Challenges

- What are the best defensive measures to invest in?

- How can it be decided whether a defensive measure from the past is still necessary?

Challenges

- What are the best defensive measures to invest in?
- How can it be decided whether a defensive measure from the past is still necessary?
- How can newly discovered attacks be efficiently and systematically documented?

First approach - Attack Trees (1999)

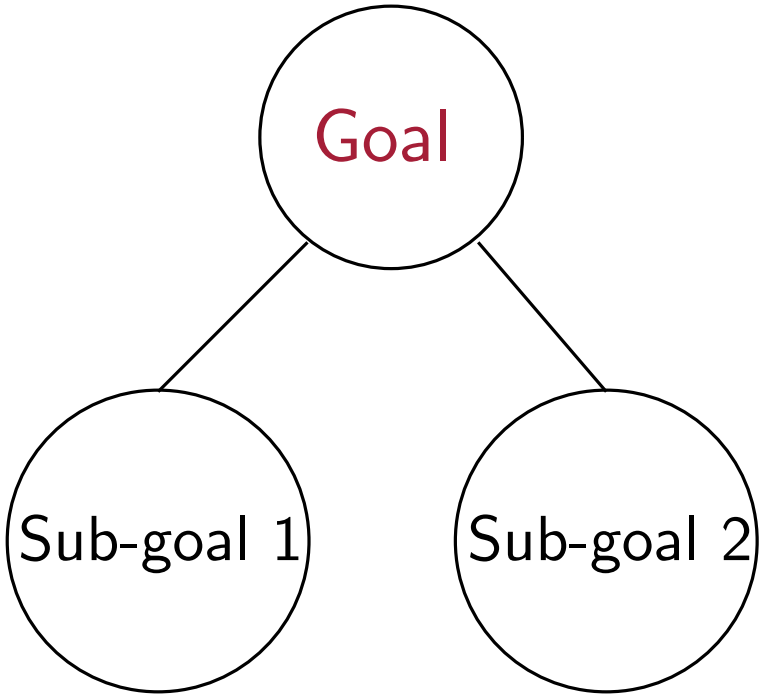
- Tool to evaluate the security of complex systems
- Tree-like representation of an attack scenario

First approach - Attack Trees (1999)

- Tool to evaluate the security of complex systems
- Tree-like representation of an attack scenario
- Root represents attacker's goal

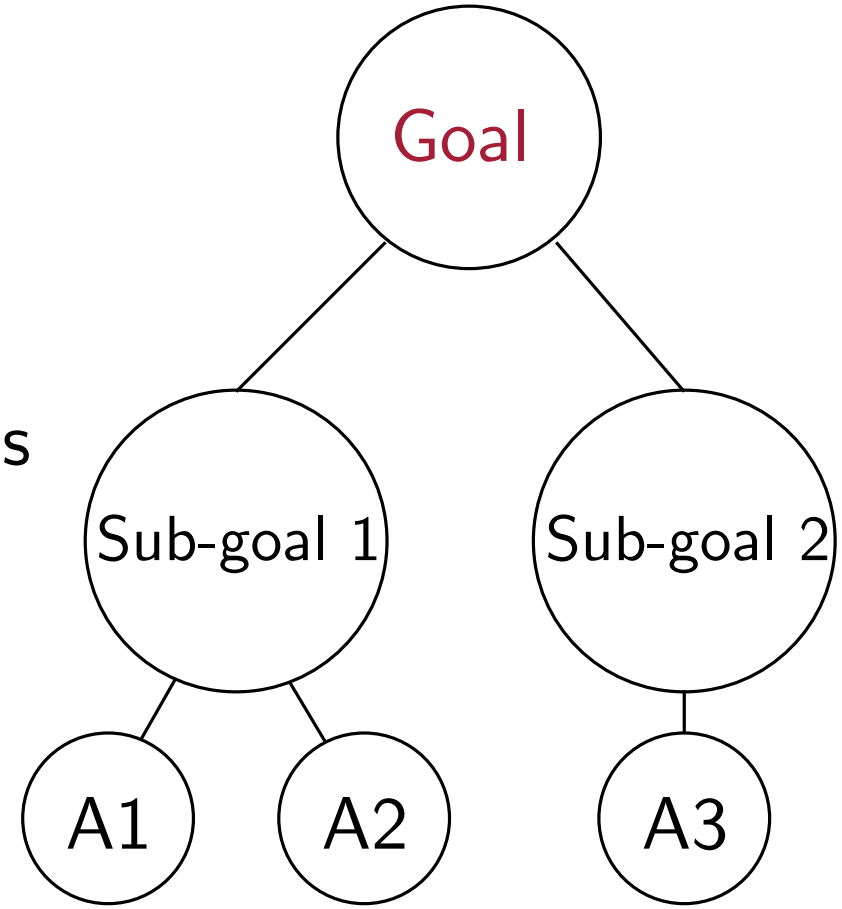
First approach - Attack Trees (1999)

- Tool to evaluate the security of complex systems
- Tree-like representation of an attack scenario
- Root represents attacker's goal
- Children nodes represent sub-goals



First approach - Attack Trees (1999)

- Tool to evaluate the security of complex systems
- Tree-like representation of an attack scenario
- Root represents attacker's goal
- Children nodes represent sub-goals
- Leaves represent the basic attacks executed by the attacker



Limitations of Attack Trees

- No capture of interaction between attacks and defense

Limitations of Attack Trees

- No capture of interaction between attacks and defense
  - Limits the precision of defense analysis

Limitations of Attack Trees

- No capture of interaction between attacks and defense
  - Limits the precision of defense analysis
- Not suitable for the evolution of security
  - Attacks *and* defense mechanisms are both to be considered

Limitations of Attack Trees

- No capture of interaction between attacks and defense
  - Limits the precision of defense analysis
- Not suitable for the evolution of security
  - Attacks *and* defense mechanisms are both to be considered
- An extension of the model is necessary for qualitative analysis

Limitations of Attack Trees

- No capture of interaction between attacks and defense
  - Limits the precision of defense analysis
- Not suitable for the evolution of security
  - Attacks *and* defense mechanisms are both to be considered
- An extension of the model is necessary for qualitative analysis
  - Attack-defense scenario portrayed as game between a proponent p and an opponent o

Attack-Defense trees (ADTrees)

- Extension of Attack Trees
- Node-labelled root tree

Attack-Defense trees (ADTrees)

- Extension of Attack Trees
- Node-labelled root tree
- Two types of nodes:
  - Attack nodes 

Attack-Defense trees (ADTrees)

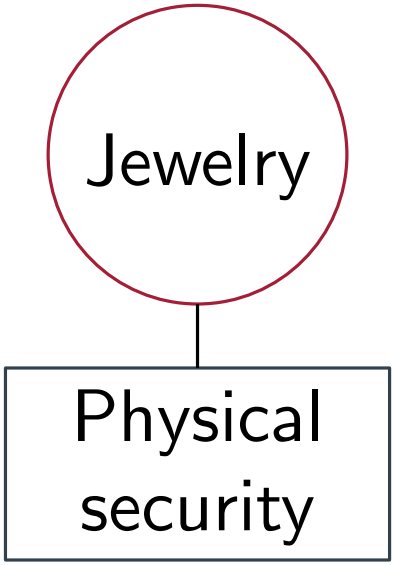
- Extension of Attack Trees
- Node-labelled root tree
- Two types of nodes:
  - Attack nodes
  - Defense nodes

Attack-Defense trees (ADTrees)

- Extension of Attack Trees
- Node-labelled root tree
- Two types of nodes:
  - Attack nodes
  - Defense nodes
- Root: goal of proponent

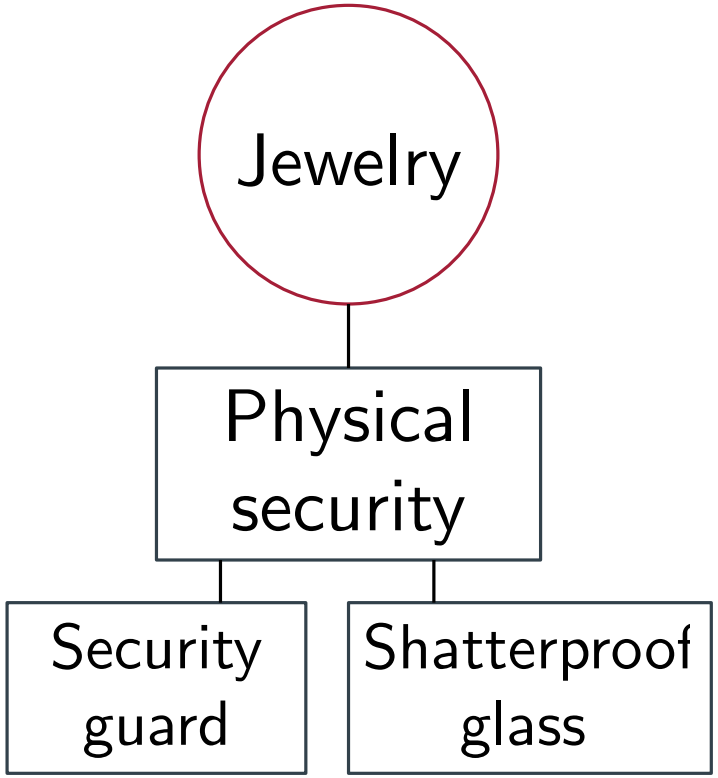
Attack-Defense trees (ADTrees)

- Extension of Attack Trees
- Node-labelled root tree
- Two types of nodes:
  - Attack nodes
  - Defense nodes
- Root: goal of proponent
- Key features:
  - Countermeasure: children of opposite type



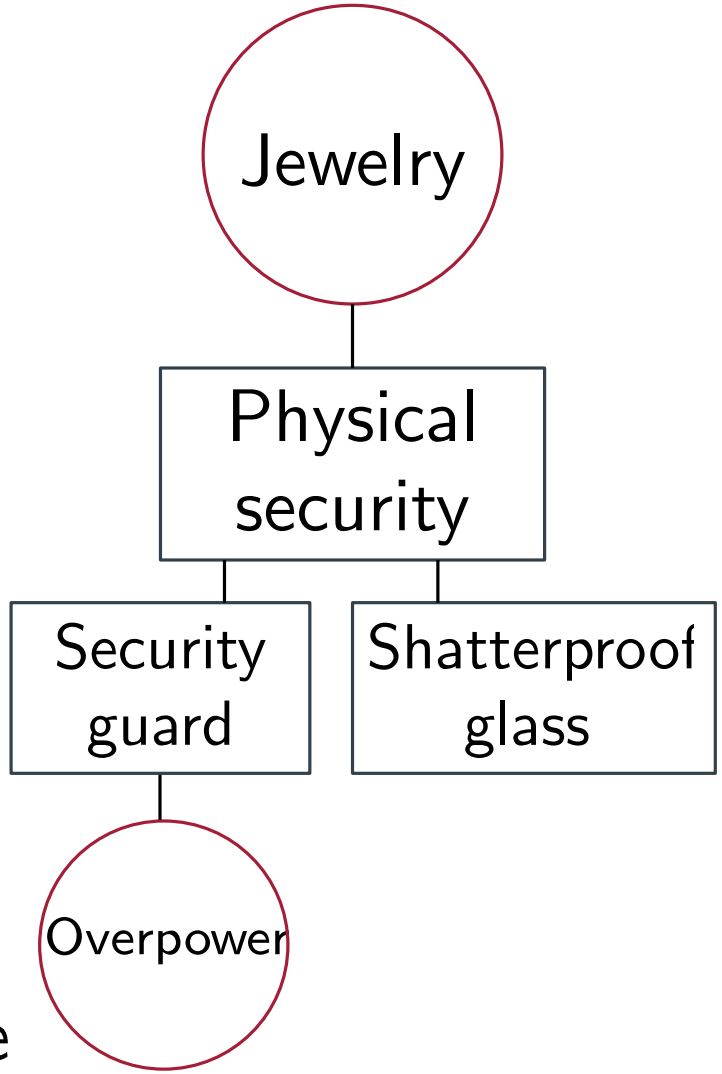
Attack-Defense trees (ADTrees)

- Extension of Attack Trees
- Node-labelled root tree
- Two types of nodes:
  - Attack nodes
  - Defense nodes
- Root: goal of proponent
- Key features:
  - Countermeasure: children of opposite type
  - Refinement: children of the same type



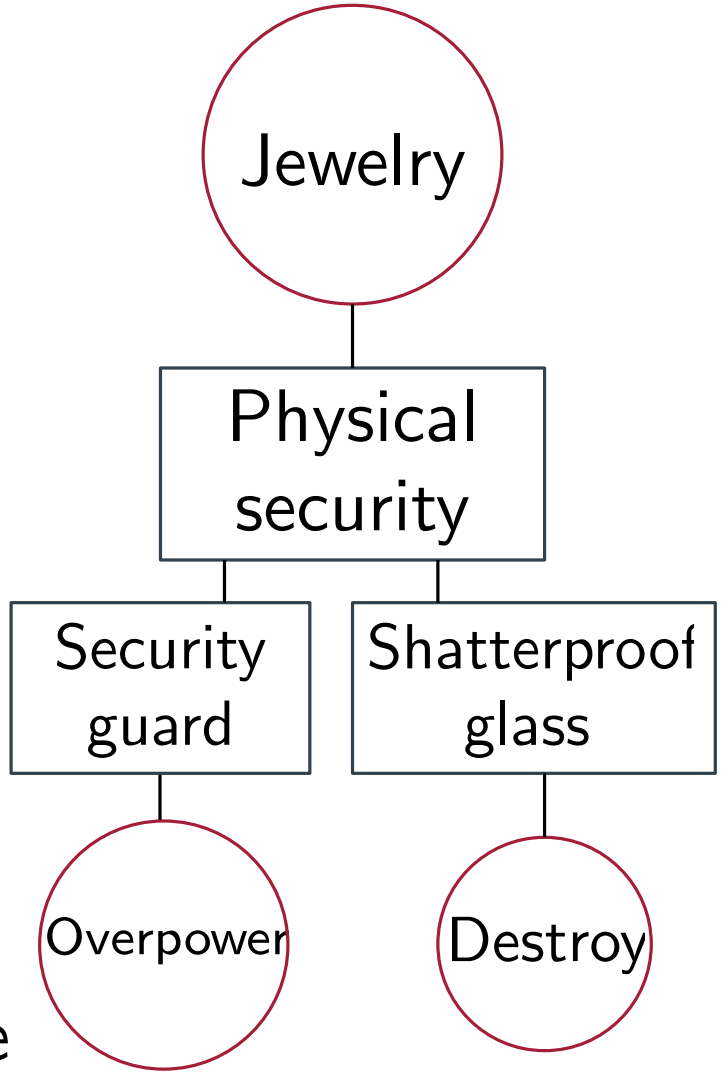
Attack-Defense trees (ADTrees)

- Extension of Attack Trees
- Node-labelled root tree
- Two types of nodes:
  - Attack nodes
  - Defense nodes
- Root: goal of proponent
- Key features:
  - Countermeasure: children of opposite type
  - Refinement: children of the same type



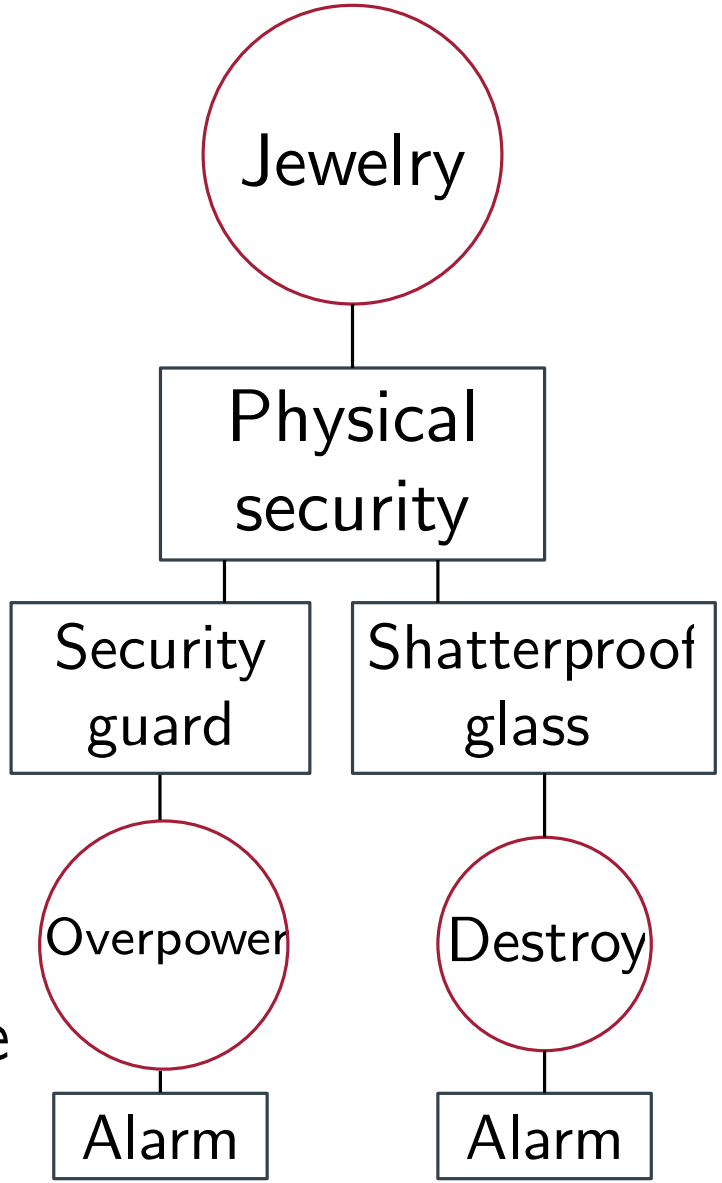
Attack-Defense trees (ADTrees)

- Extension of Attack Trees
- Node-labelled root tree
- Two types of nodes:
  - Attack nodes
  - Defense nodes
- Root: goal of proponent
- Key features:
  - Countermeasure: children of opposite type
  - Refinement: children of the same type



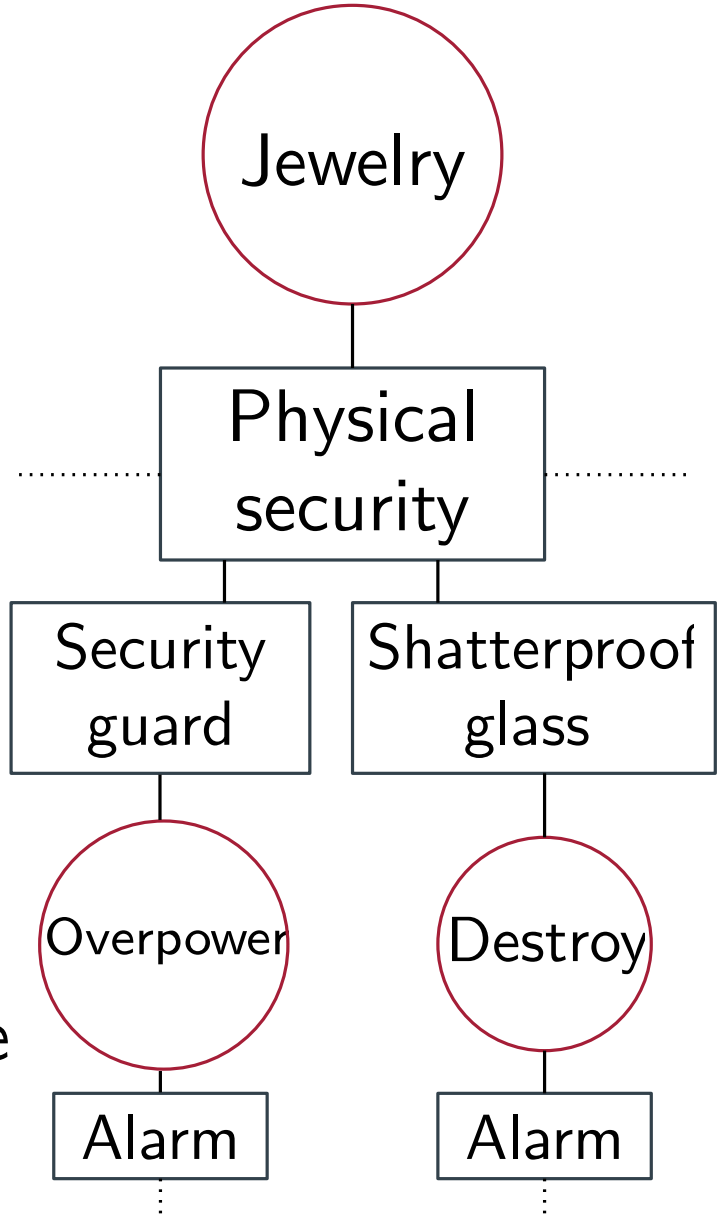
Attack-Defense trees (ADTrees)

- Extension of Attack Trees
- Node-labelled root tree
- Two types of nodes:
  - Attack nodes
  - Defense nodes
- Root: goal of proponent
- Key features:
  - Countermeasure: children of opposite type
  - Refinement: children of the same type



Attack-Defense trees (ADTrees)

- Extension of Attack Trees
- Node-labelled root tree
- Two types of nodes:
  - Attack nodes
  - Defense nodes
- Root: goal of proponent
- Key features:
  - Countermeasure: children of opposite type
  - Refinement: children of the same type

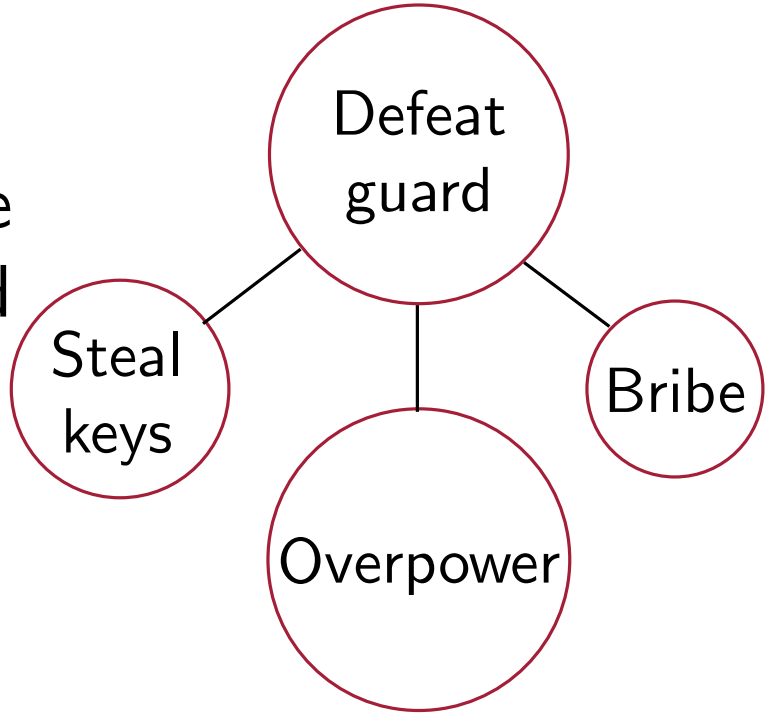


Types of refinement

- Disjunctive refinement
  - A goal is achieved iff at least one of it's children goals are achieved

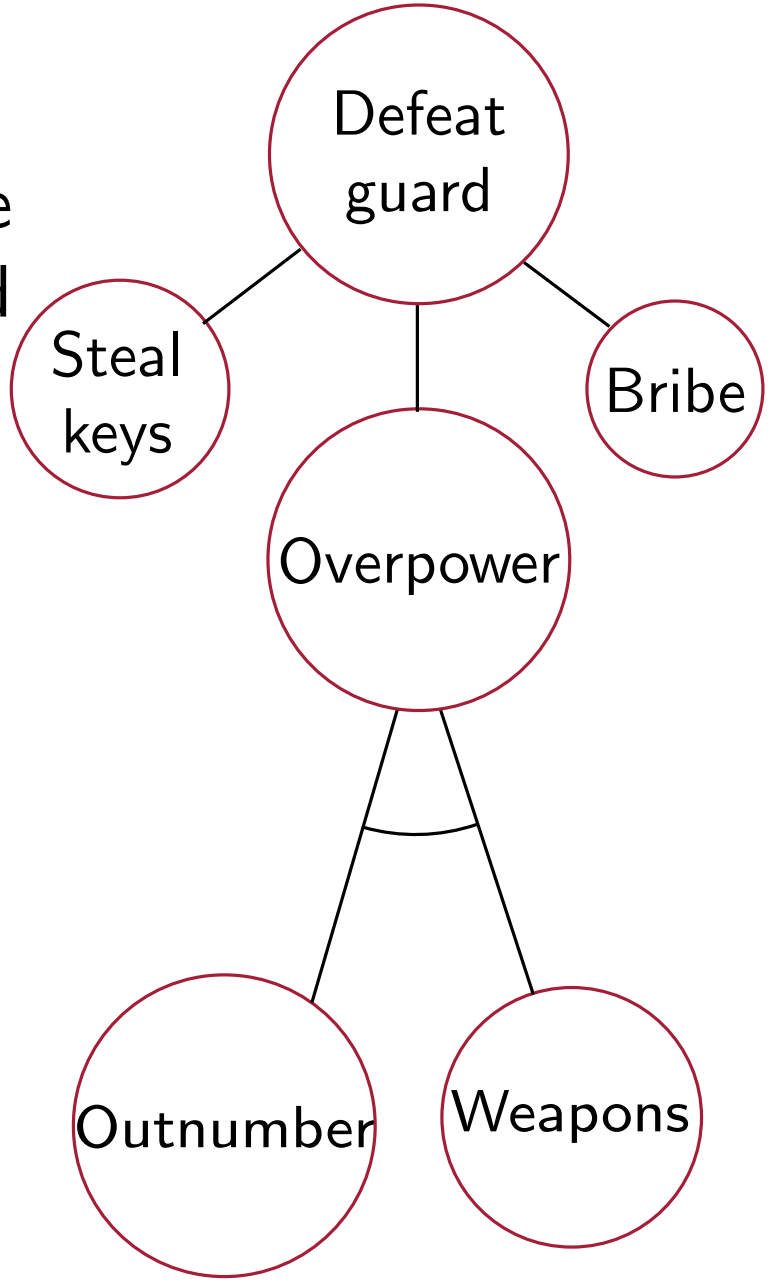
Types of refinement

- Disjunctive refinement
  - A goal is achieved iff at least one of it's children goals are achieved



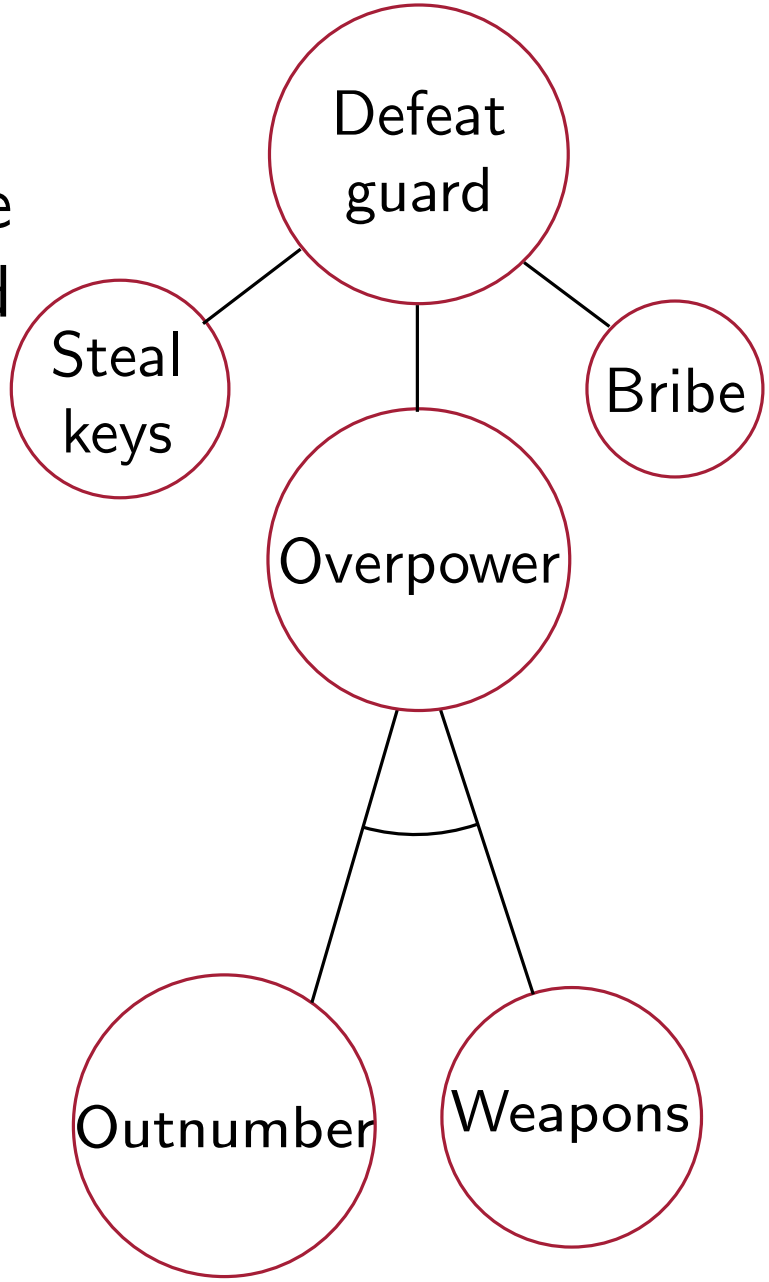
Types of refinement

- Disjunctive refinement
  - A goal is achieved iff at least one of it's children goals are achieved
- Conjunctive refinement
  - A goal is achieved iff all of it's children goals are achieved



Types of refinement

- Disjunctive refinement
  - A goal is achieved iff at least one of it's children goals are achieved
- Conjunctive refinement
  - A goal is achieved iff all of it's children goals are achieved
- Nodes without any refinement will represent *basic actions*



That's nice and all, but...

That's nice and all, but...

- ... we need a method to define semantics of the tree representation (syntax)
  - ADTerms

That's nice and all, but...

- ... we need a method to define semantics of the tree representation (syntax)
  - ADTerms
- ... we then need to define semantics of ADTrees (logic)
  - Propositional semantics for ADTrees

That's nice and all, but...

- ... we need a method to define semantics of the tree representation (syntax)
  - ADTerms
- ... we then need to define semantics of ADTrees (logic)
  - Propositional semantics for ADTrees
- ... then, we finally need an approach to quantitatively analyze attack-defense scenarios (algorithm)
  - Attributes

ADTerms

- Abstract syntax to formally analyse ADTrees

ADTerms

- Abstract syntax to formally analyse ADTrees
- A *signature* $\Sigma = (S, F)$ is a pair of set of types and a set of function symbols, where:

ADTerms

- Abstract syntax to formally analyse ADTrees
- A *signature* $\Sigma = (S, F)$ is a pair of set of types and a set of function symbols, where:
  - $S = \{p, o\}$ for proponent / opponent labelling

ADTerms

- Abstract syntax to formally analyse ADTrees
- A *signature* $\Sigma = (S, F)$ is a pair of set of types and a set of function symbols, where:
  - $S = \{p, o\}$ for proponent / opponent labelling
- F consists of:
  - $\mathbb{B}^s$: typed constants for basic actions of type $s \in S$

ADTerms

- Abstract syntax to formally analyse ADTrees
- A *signature* $\Sigma = (S, F)$ is a pair of set of types and a set of function symbols, where:
  - $S = \{p, o\}$ for proponent / opponent labelling
- F consists of:
  - $\mathbb{B}^s$: typed constants for basic actions of type $s \in S$
  - $\vee^s / \wedge^s$: functions for disjunctive / conjunctive refinement of type $s \in S$

ADTerms

- Abstract syntax to formally analyse ADTrees
- A *signature* $\Sigma = (S, F)$ is a pair of set of types and a set of function symbols, where:
  - $S = \{p, o\}$ for proponent / opponent labelling
- F consists of:
  - $\mathbb{B}^s$: typed constants for basic actions of type $s \in S$
  - $\vee^s / \wedge^s$: functions for disjunctive / conjunctive refinement of type $s \in S$
  - c^s: binary function to connect countermeasures from s to $\overline{s}$

ADTerms

- Abstract syntax to formally analyse ADTrees
- A *signature* $\Sigma = (S, F)$ is a pair of set of types and a set of function symbols, where:
  - $S = \{p, o\}$ for proponent / opponent labelling
- F consists of:
  - $\mathbb{B}^s$: typed constants for basic actions of type $s \in S$
  - $\vee^s / \wedge^s$: functions for disjunctive / conjunctive refinement of type $s \in S$
  - c^s: binary function to connect countermeasures from s to $\overline{s}$
- Terms typed over a signature Σ

ADTerms

- Formally, an ADTree is a finite ordered tree T over the set of labels $\mathbb{B}^s \cup \wedge^s \cup \vee^s, s \in S$

- Moreover, a function $\lambda : Pos(T) \rightarrow \{\square, \bigcirc\}$ will help distinguish between attack and defense nodes (c^s) and determines the proponent and opponent

ADTerms

- Formally, an ADTree is a finite ordered tree T over the set of labels $\mathbb{B}^s \cup \wedge^s \cup \vee^s, s \in S$

- Moreover, a function $\lambda : Pos(T) \rightarrow \{\square, \bigcirc\}$ will help distinguish between attack and defense nodes (c^s) and determines the proponent and opponent

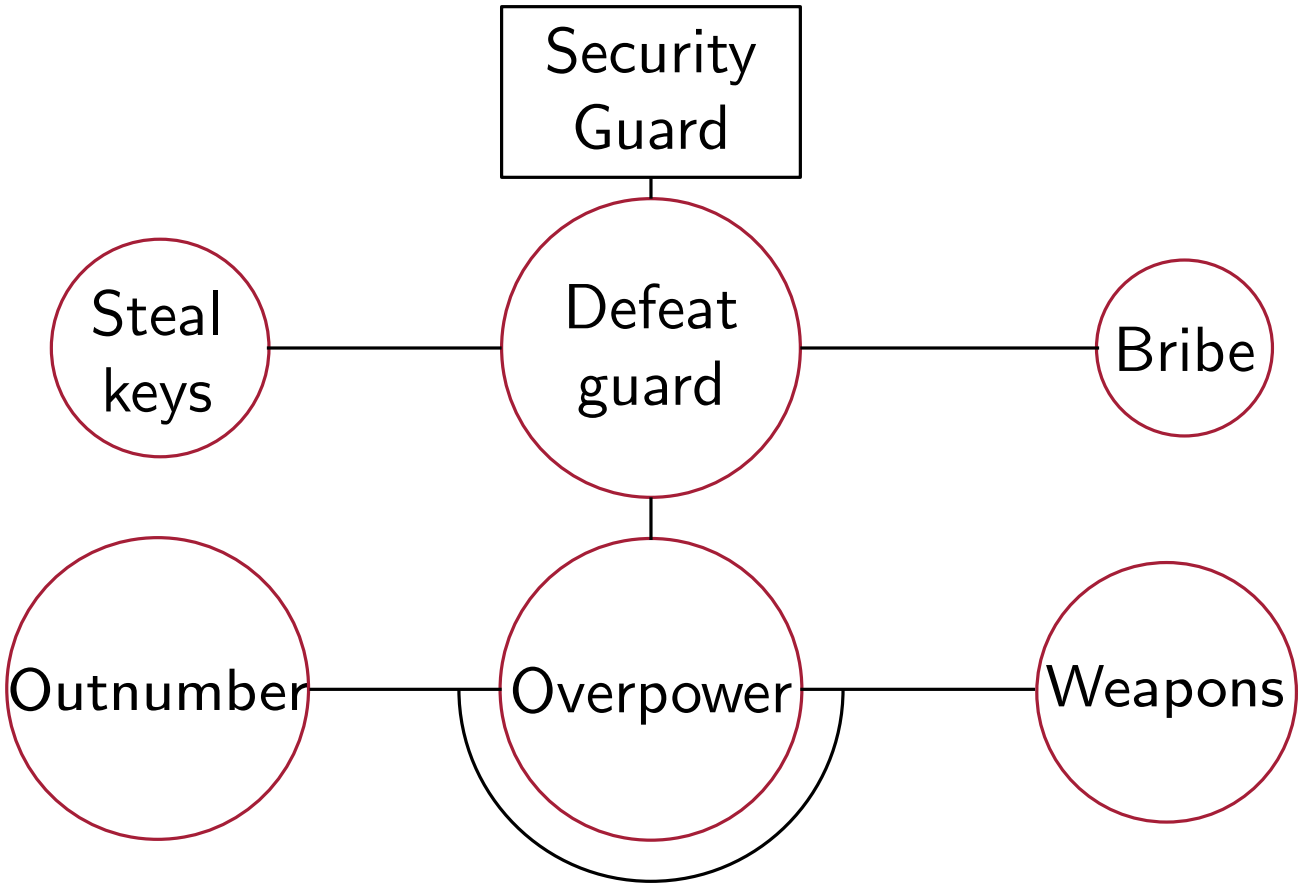
- Ordering is necessary to describe a tree mathematically and illustrate the ADTree

ADTerms

- Formally, an ADTree is a finite ordered tree T over the set of labels $\mathbb{B}^s \cup \wedge^s \cup \vee^s, s \in S$
- Moreover, a function $\lambda : Pos(T) \rightarrow \{\square, \bigcirc\}$ will help distinguish between attack and defense nodes (c^s) and determines the proponent and opponent
- Ordering is necessary to describe a tree mathematically and illustrate the ADTree
- Any ADTree can be transformed into an ADTerm and vice versa

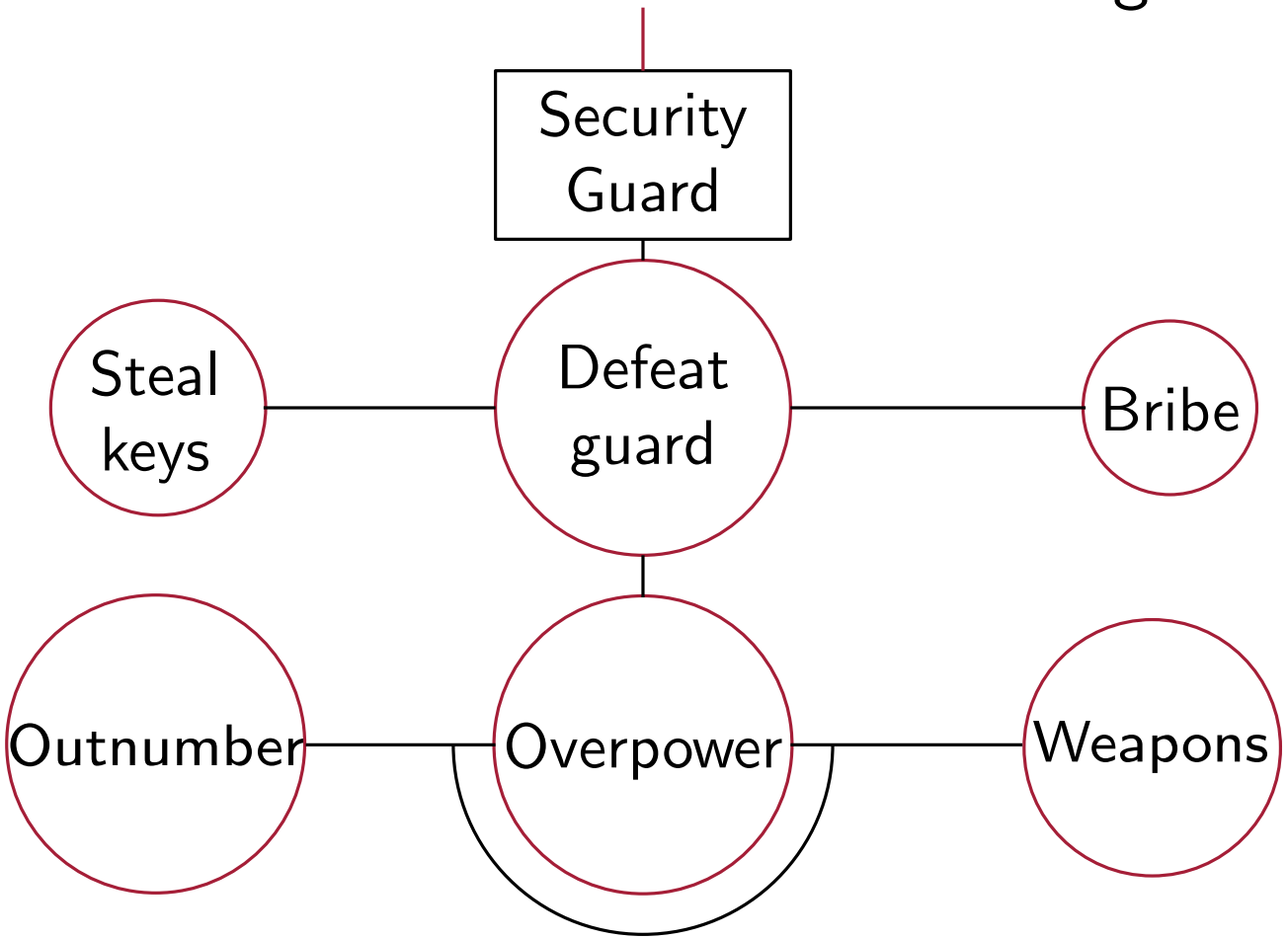
ADTerms

- This sub-tree can be evaluated to following ADTerm:



ADTerms

- This sub-tree can be evaluated to following ADTerm:



$$c^o\,(\text{SecGuard}, c^p\,(\vee^p\,(\text{StealKeys}, \wedge^p\,(\text{Outnum}, \text{Weapons})\,, \text{Bribe})))$$

Propositional semantics

- Most frequently used semantics for attack trees

Propositional semantics

- Most frequently used semantics for attack trees
- Also suitable for ADTerms

Propositional semantics

- Most frequently used semantics for attack trees
- Also suitable for ADTerms
- How it's done:
  - Interpret an ADTerm t as propositional formula t_p

Propositional semantics

- Most frequently used semantics for attack trees
- Also suitable for ADTerms
- How it's done:
  - Interpret an ADTerm t as propositional formula t_p
  - Satisfiability of t_p corresponds to feasibility of scenario represented by t
- Defeat the guard!

Propositional semantics

- Most frequently used semantics for attack trees
- Also suitable for ADTerms
- How it's done:
  - Interpret an ADTerm t as propositional formula t_p
  - Satisfiability of t_p corresponds to feasibility of scenario represented by t
- Defeat the guard!

$$c^o\left(\text{SecGuard}, c^p\left(\vee^p\left(\text{StealKeys}, \wedge^p\left(\text{Outnum}, \text{Weapons}\right), \text{Bribe}\right)\right)\right)$$

$$\Leftrightarrow$$

$$\text{StealKeys} \vee (\text{Outnumber} \wedge \text{Weapons}) \vee \text{Bribe}$$

Propositional semantics

- Most frequently used semantics for attack trees
- Also suitable for ADTerms
- How it's done:
  - Interpret an ADTerm t as propositional formula t_p
  - Satisfiability of t_p corresponds to feasibility of scenario represented by t
- Defeat the guard!

$$c^o\left(\text{SecGuard}, c^p\left(\vee^p\left(\text{StealKeys}, \wedge^p\left(\text{Outnum}, \text{Weapons}\right), \text{Bribe}\right)\right)\right)$$

$$\Leftrightarrow$$

$$\text{StealKeys} \vee (\text{Outnumber} \wedge \text{Weapons}) \vee \text{Bribe}$$

- Finding a feasible attack for a given ADTree is $\mathcal{NP}$-complete

Limitations

- Propositional semantics is limited to binary properties of interest

Limitations

- Propositional semantics is limited to binary properties of interest
  - "Is the system vulnerable to a given attack *a*?"

Limitations

- Propositional semantics is limited to binary properties of interest
  - "Is the system vulnerable to a given attack *a*?"
- Statements of higher quality need a new approach of semantics

Limitations

- Propositional semantics is limited to binary properties of interest
  - "Is the system vulnerable to a given attack *a*?"
- Statements of higher quality need a new approach of semantics
  - "How useful / certain is an attack *a* for a given system?"

Limitations

- Propositional semantics is limited to binary properties of interest
  - "Is the system vulnerable to a given attack *a*?"
- Statements of higher quality need a new approach of semantics
  - "How useful / certain is an attack *a* for a given system?"
- Other approaches of semantics available for more conclusive answers

Attributes

- Enables a quantitative analysis of attack-defense scenarios
- works “bottom-up“ for a given ADTerm

Attributes

- Enables a quantitative analysis of attack-defense scenarios
- works "bottom-up" for a given ADTerm
- A kind of mapping is necessary in order to interpret the operations defined for ADTerms:
  - $\mathbb{B}^s$: typed constants for basic actions of type $s \in S$
  - $\vee^s / \wedge^s$: functions for disjunctive / conjunctive refinement of type $s \in S$
  - c^s: binary function to connect countermeasures from s to $\overline{s}$

Attributes

- Enables a quantitative analysis of attack-defense scenarios
- works "bottom-up" for a given ADTerm
- A kind of mapping is necessary in order to interpret the operations defined for ADTerms:
  - $\mathbb{B}^s$: typed constants for basic actions of type $s \in S$
  - $\vee^s / \wedge^s$: functions for disjunctive / conjunctive refinement of type $s \in S$
  - c^s: binary function to connect countermeasures from s to $\overline{s}$
- This "mapping" defines a *domain* an attribute is defined on

Attribute Domain

- The attribute domain is a tuple

$$A_\alpha = (D_\alpha, \underbrace{\vee_\alpha^p, \wedge_\alpha^p, \vee_\alpha^o, \wedge_\alpha^o, c_\alpha^p, c_\alpha^o}_{\text{functions defined on } D_\alpha})$$

Set of values

Attribute Domain

- The attribute domain is a tuple

$$A_\alpha = (D_\alpha, \underbrace{\vee^p_\alpha, \wedge^p_\alpha, \vee^o_\alpha, \wedge^o_\alpha, c^p_\alpha, c^o_\alpha}_{\text{functions defined on } D_\alpha})$$

Set of values

- Consider the following *attribute domain*

$$A_{\text{sat}} = \{\{0, 1\}, \vee, \wedge, \vee, \wedge, \times\ , \times\ \}$$

$$x \times y = x \wedge \neg y$$

Attribute Domain

- The attribute domain is a tuple

$$A_\alpha = (D_\alpha, \underbrace{\vee^p_\alpha, \wedge^p_\alpha, \vee^o_\alpha, \wedge^o_\alpha, c^p_\alpha, c^o_\alpha}_{\text{functions defined on } D_\alpha})$$

Set of values

- Consider the following *attribute domain*

$$A_{\text{sat}} = \{\{0, 1\}, \vee, \wedge, \vee, \wedge, \times\ , \times\ \}$$

$$x \times y = x \wedge \neg y$$

- A_{sat} is the domain to evaluate the satisfiability of the proponent's goal modelled by the root

Attribute Domain

- The attribute domain is a tuple

$$A_\alpha = (D_\alpha, \underbrace{\vee^p_\alpha, \wedge^p_\alpha, \vee^o_\alpha, \wedge^o_\alpha, c^p_\alpha, c^o_\alpha}_{\text{functions defined on } D_\alpha})$$

Set of values

- Consider the following *attribute domain*

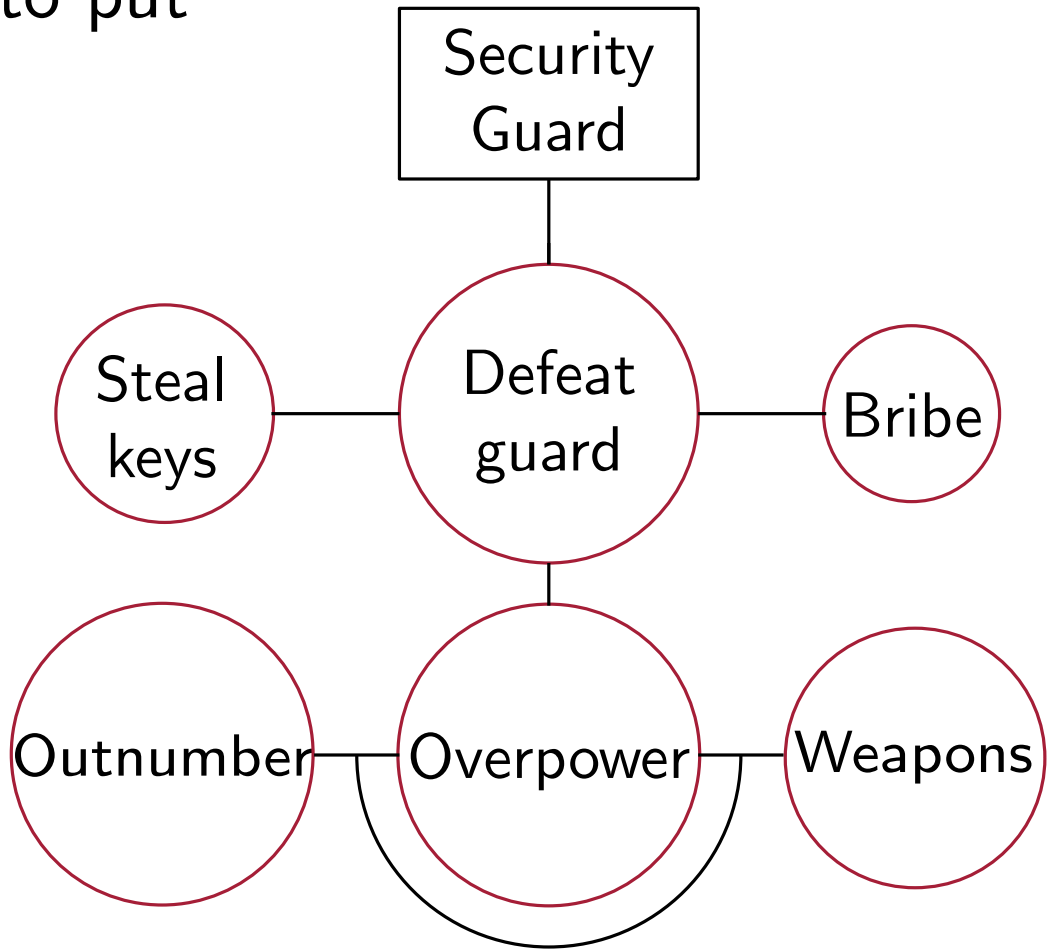
$$A_{\text{sat}} = \{\{0, 1\}, \vee, \wedge, \vee, \wedge, \times\ , \times\ \}$$

$$x \times y = x \wedge \neg y$$

- A_{sat} is the domain to evaluate the satisfiability of the proponent's goal modelled by the root
- Every ADTerm is evaluatable for a given attribute domain

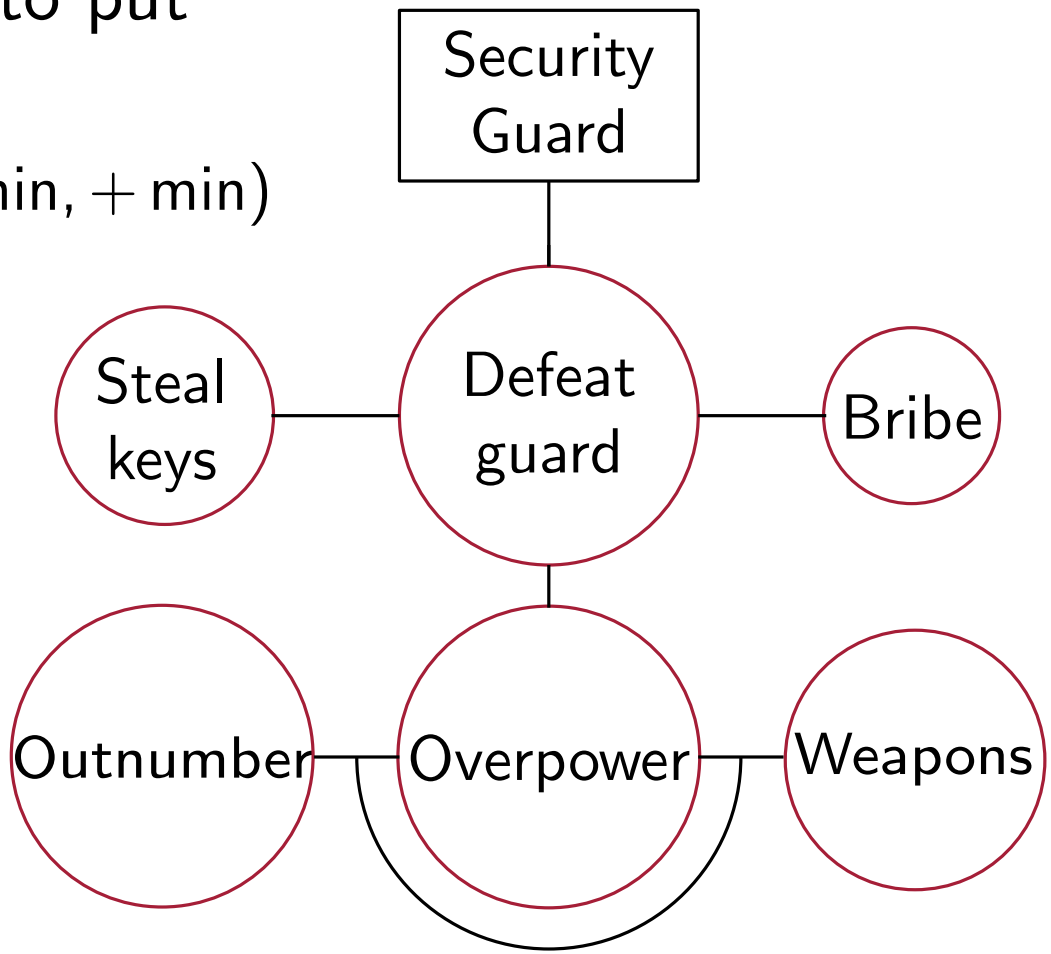
Attribute evaluation

- Evaluate the minimal cost to put down the guard



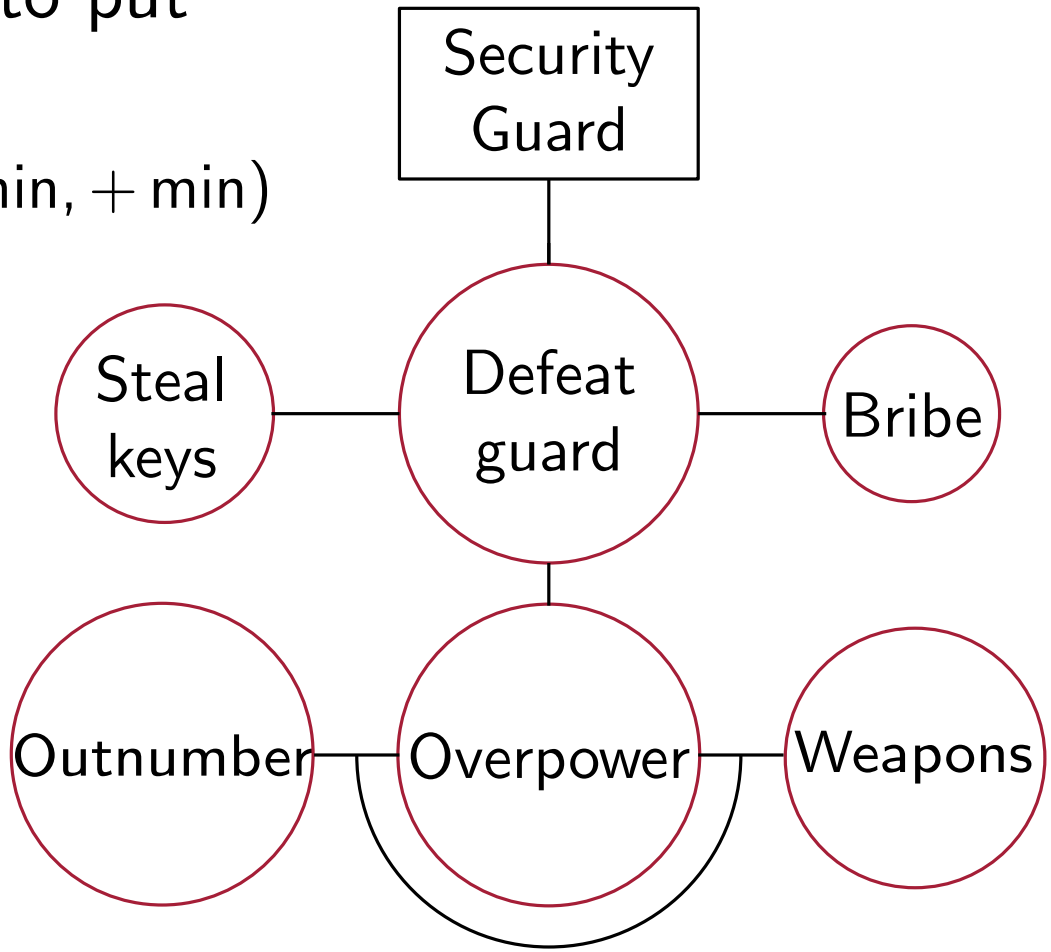
Attribute evaluation

- Evaluate the minimal cost to put down the guard
- $A_{\text{cost}} = (\mathbb{R}^+ \cup \{\infty\}, \min, \cdot, +, \min, + \min)$



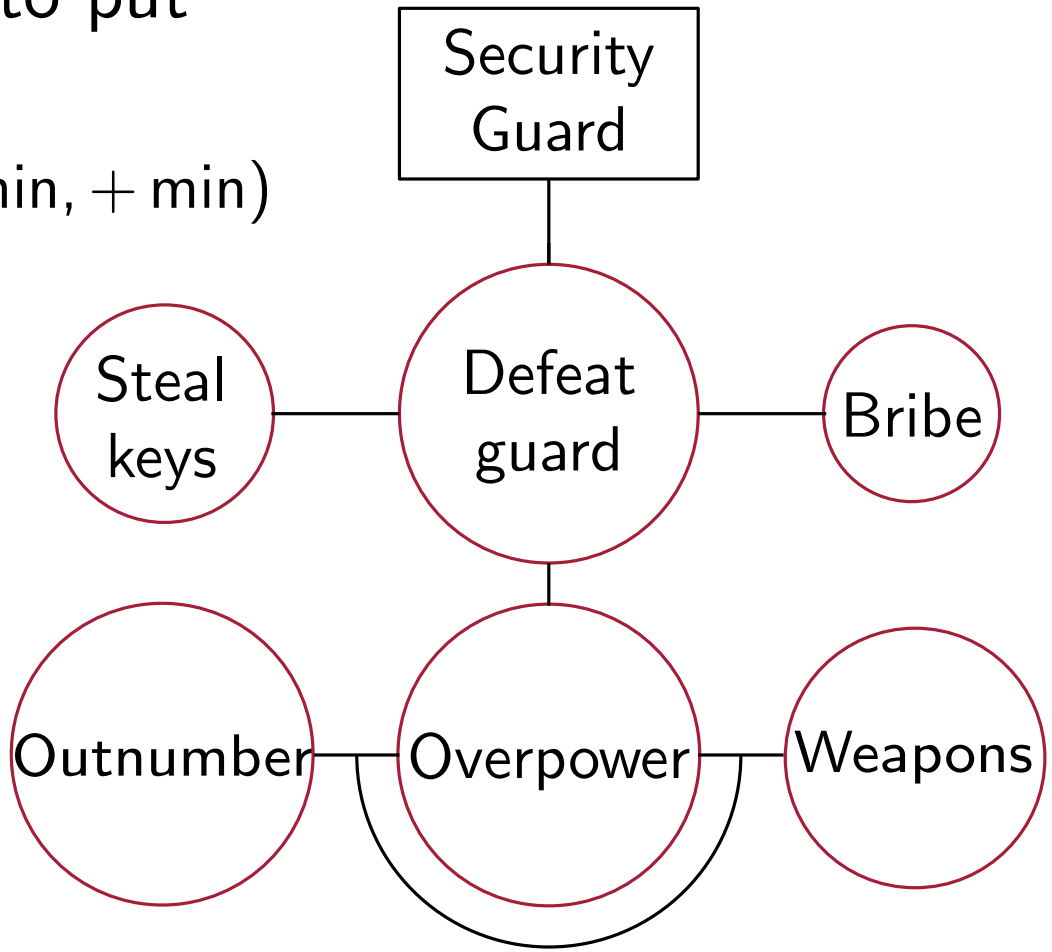
Attribute evaluation

- Evaluate the minimal cost to put down the guard
- $A_{\text{cost}} = (\mathbb{R}^+ \cup \{\infty\}, \min, \cdot, +, \min, + \min)$
- Cost function β
  - $\beta(\text{Weapons}) = 500$



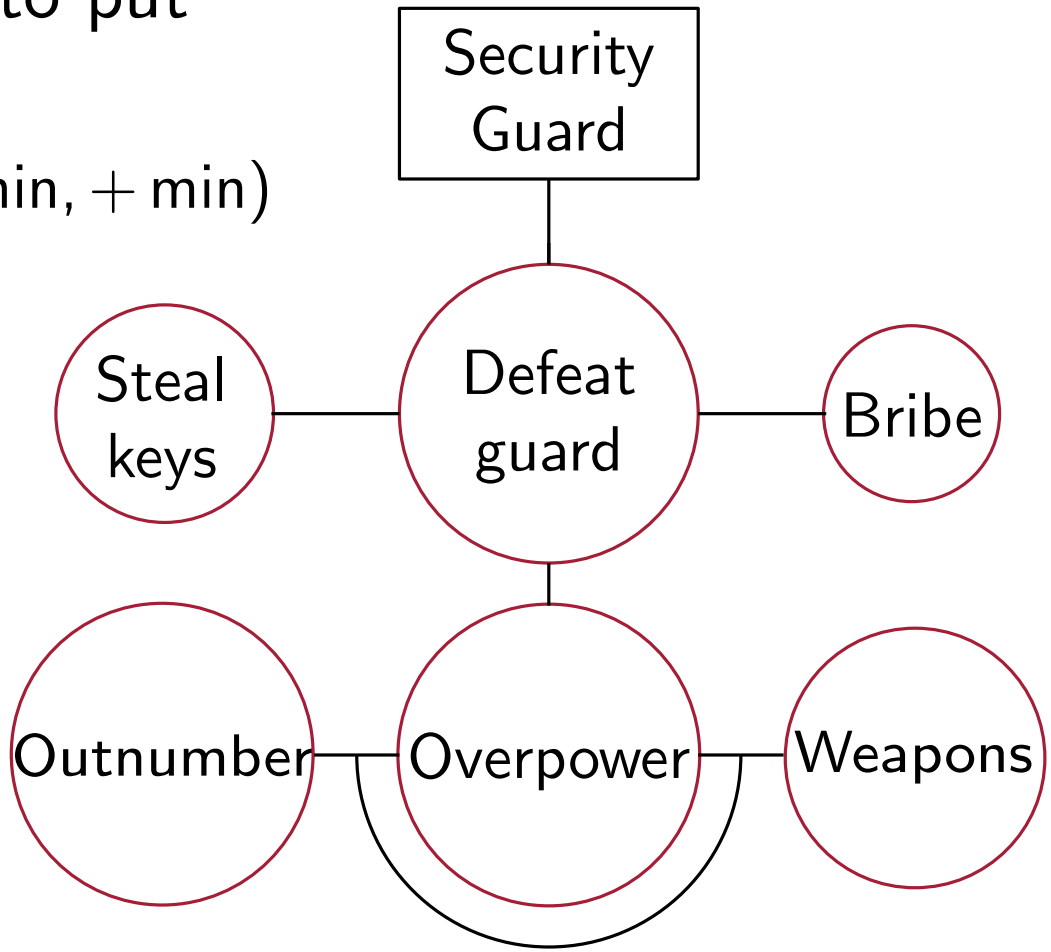
Attribute evaluation

- Evaluate the minimal cost to put down the guard
- $A_{\text{cost}} = (\mathbb{R}^+ \cup \{\infty\}, \min, \cdot, +, \min, + \min)$
- Cost function β
  - $\beta(\text{Weapons}) = 500$
  - $\beta(Outnum) = 4$



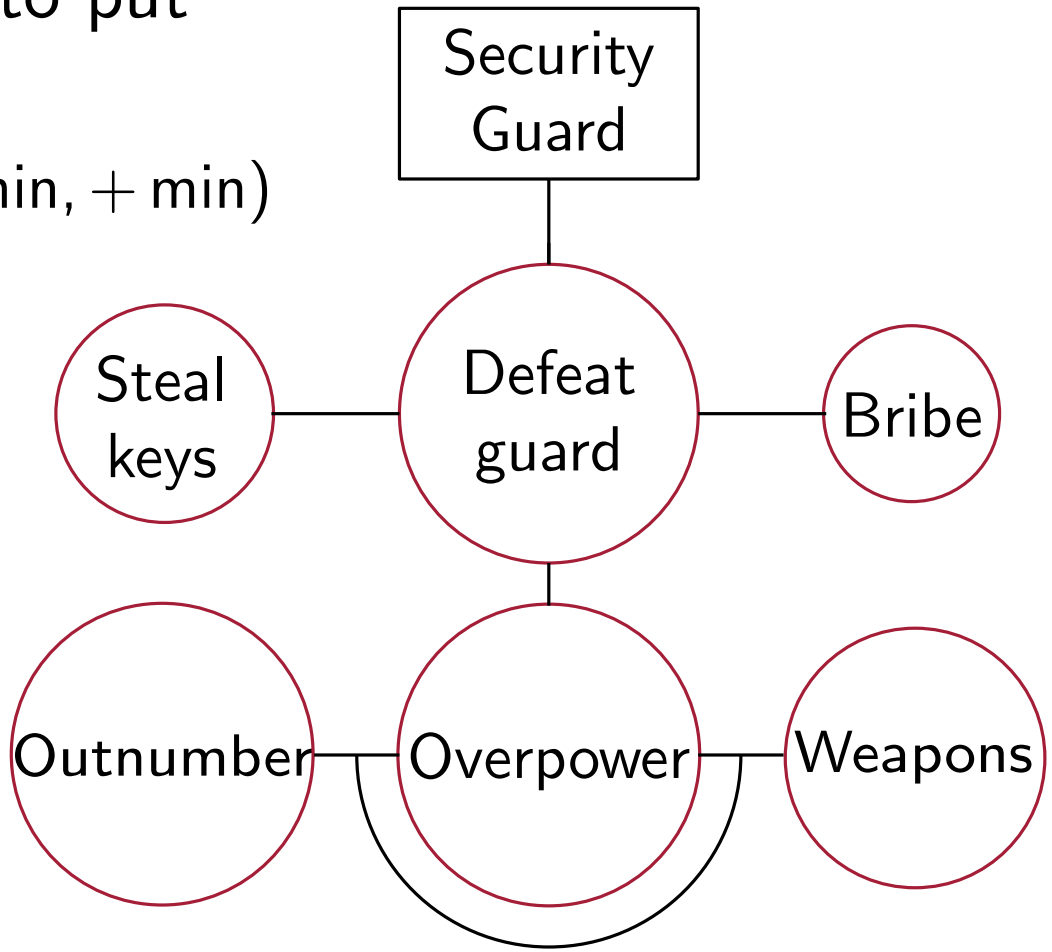
Attribute evaluation

- Evaluate the minimal cost to put down the guard
- $A_{\text{cost}} = (\mathbb{R}^+ \cup \{\infty\}, \min, \cdot, +, \min, + \min)$
- Cost function β
  - $\beta(\text{Weapons}) = 500$
  - $\beta(Outnum) = 4$
  - $\beta(\text{Keys}) = \infty$



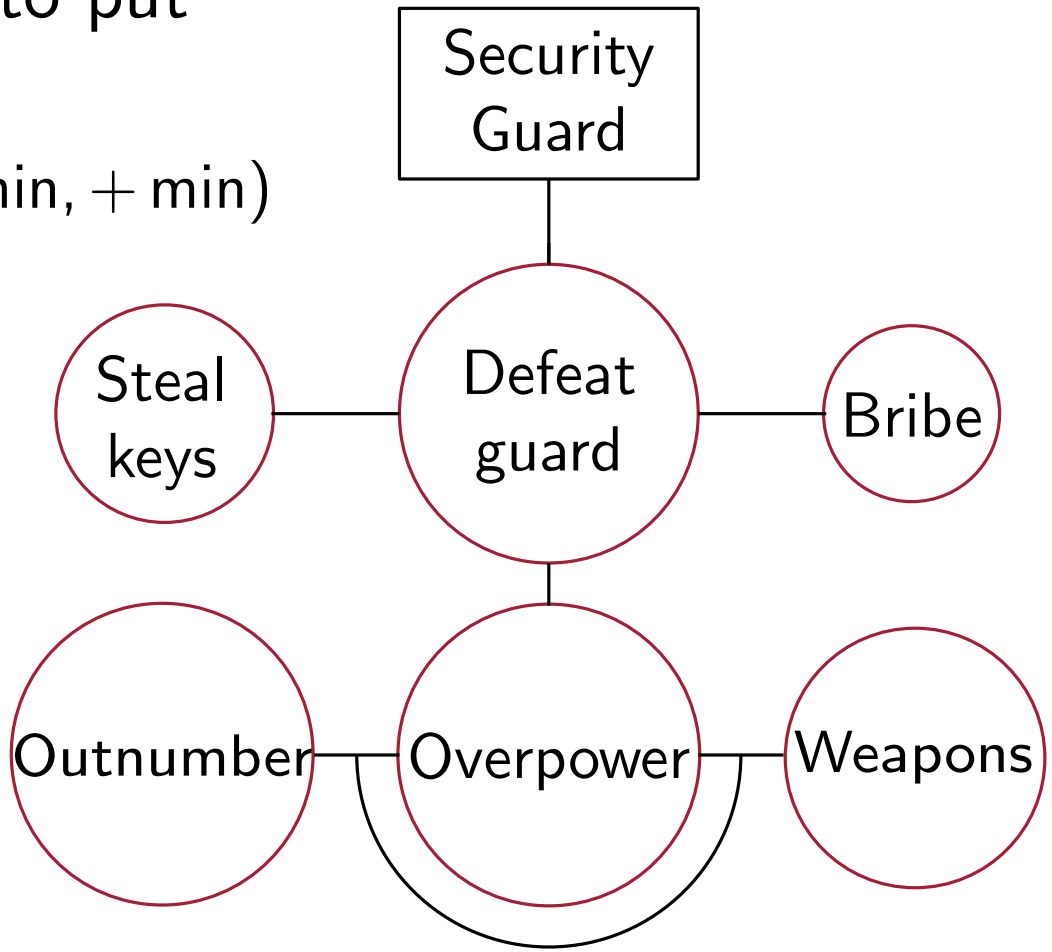
Attribute evaluation

- Evaluate the minimal cost to put down the guard
- $A_{\text{cost}} = (\mathbb{R}^+ \cup \{\infty\}, \min, \cdot, +, \min, + \min)$
- Cost function β
  - $\beta(\text{Weapons}) = 500$
  - $\beta(Outnum) = 4$
  - $\beta(\text{Keys}) = \infty$
  - $\beta(\text{Bribe}) = 10000$



Attribute evaluation

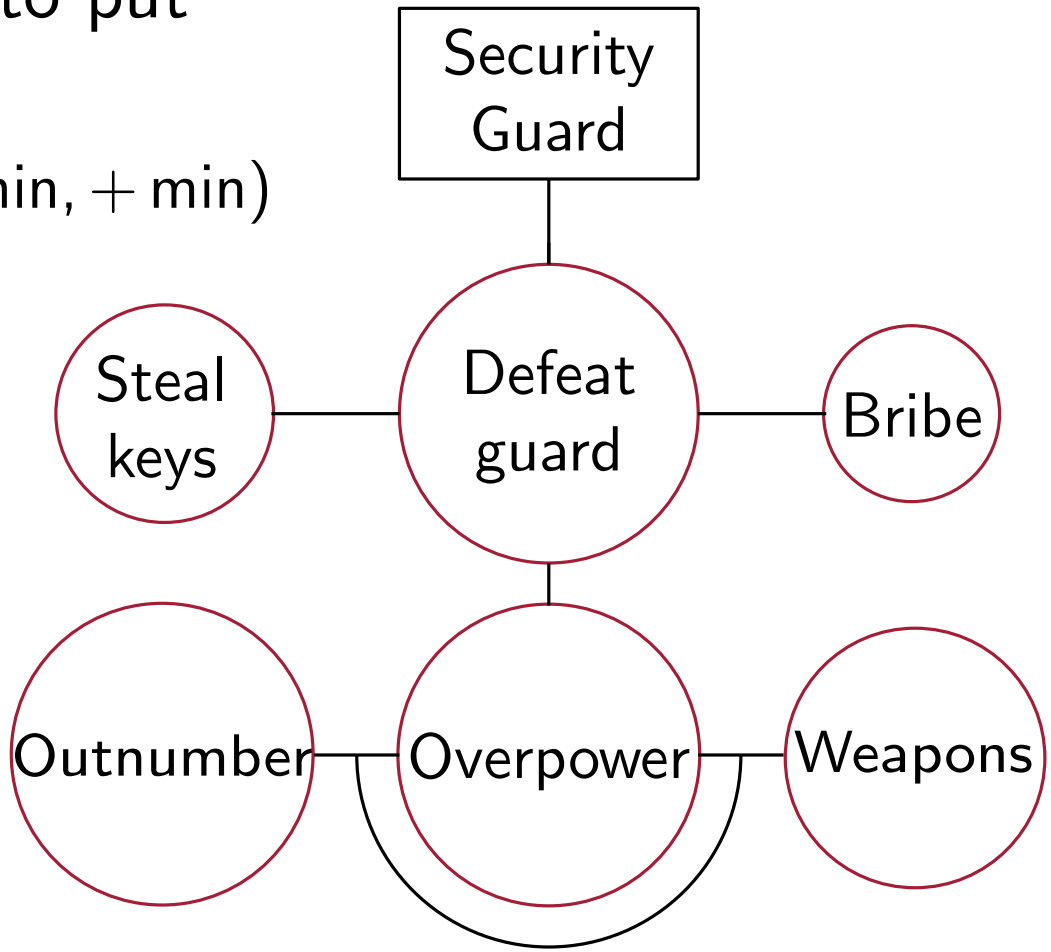
- Evaluate the minimal cost to put down the guard
- $A_{\text{cost}} = (\mathbb{R}^+ \cup \{\infty\}, \min, \cdot, +, \min, + \min)$
- Cost function β
  - $\beta(\text{Weapons}) = 500$
  - $\beta(Outnum) = 4$
  - $\beta(\text{Keys}) = \infty$
  - $\beta(\text{Bribe}) = 10000$



$t = \vee^p (\text{StealKeys}, \wedge^p (\text{Outnum}, \text{Weapons}), \text{Bribe})$

Attribute evaluation

- Evaluate the minimal cost to put down the guard
- $A_{\text{cost}} = (\mathbb{R}^+ \cup \{\infty\}, \min, \cdot, +, \min, + \min)$
- Cost function β
  - $\beta(\text{Weapons}) = 500$
  - $\beta(Outnum) = 4$
  - $\beta(\text{Keys}) = \infty$
  - $\beta(\text{Bribe}) = 10000$



$t = \vee^p (\text{StealKeys}, \wedge^p (\text{Outnum}, \text{Weapons}), \text{Bribe})$

$\Rightarrow \text{cost}(t) = (\min(\infty, \cdot(4, 500), 10000)) = 2000$

Summary

- ADTrees as extension of Attack Trees can represent a security system evolution

Summary

- ADTrees as extension of Attack Trees can represent a security system evolution
- ADTerms deliver an attack-defense language to operate on

Summary

- ADTrees as extension of Attack Trees can represent a security system evolution
- ADTerms deliver an attack-defense language to operate on
- Attributes for ADTerms deliver a base to evaluate a given system, considering a semantics

Summary

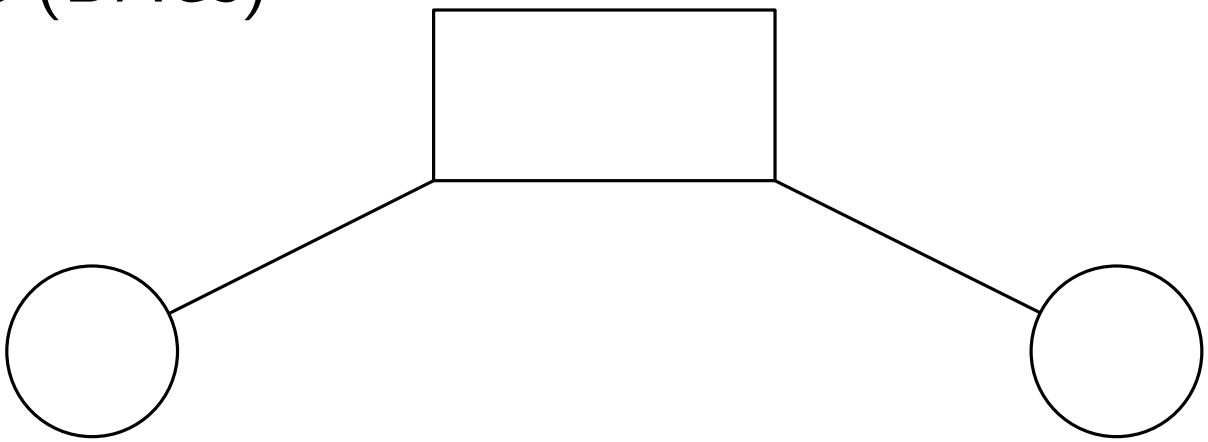
- ADTrees as extension of Attack Trees can represent a security system evolution
- ADTerms deliver an attack-defense language to operate on
- Attributes for ADTerms deliver a base to evaluate a given system, considering a semantics
- Computer tool “ADTool2“ supports large ADTrees with graphical representation and evaluation algorithms

GitHub repository

Homepage of SaToSS Luxembourg

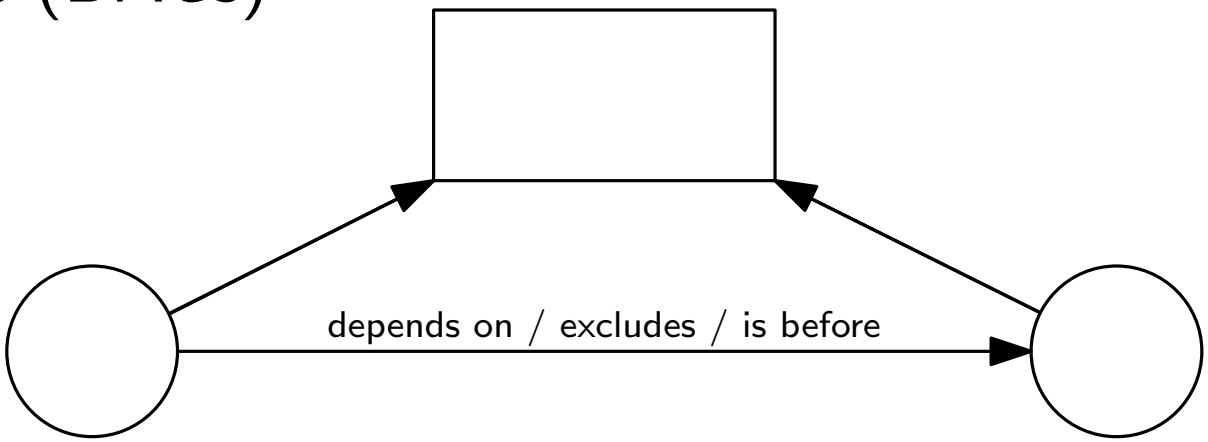
Next (big) step

- Introduction of dependencies in an ADTree
  - Extension of the model to a subclass of Directed Acyclic Graphs (DAGs)



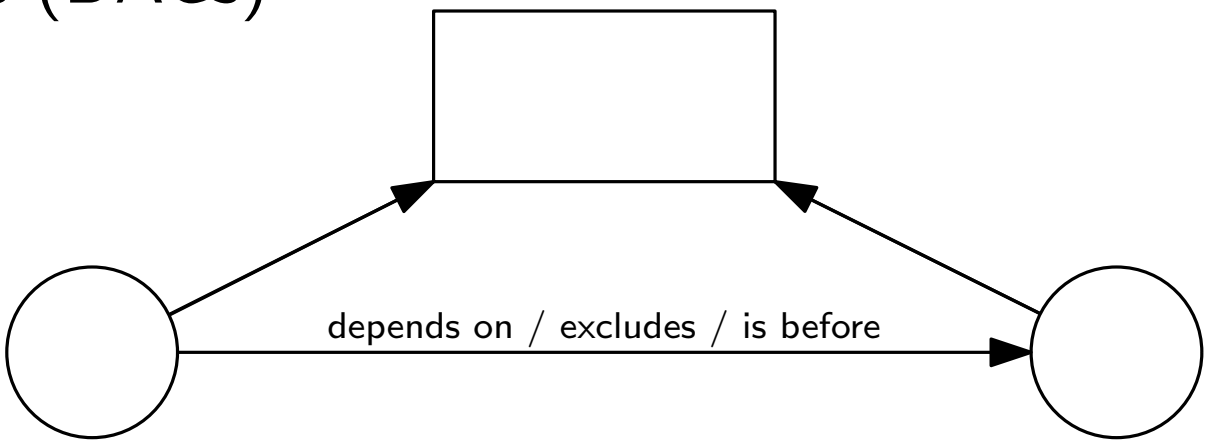
Next (big) step

- Introduction of dependencies in an ADTree
  - Extension of the model to a subclass of Directed Acyclic Graphs (DAGs)



Next (big) step

- Introduction of dependencies in an ADTree
  - Extension of the model to a subclass of Directed Acyclic Graphs (DAGs)



- Then, an execution order can be evaluated
- Also, attack-defense scenario from a propabilistic point of view can be analyzed

Questions?

Thank you!

:)