UCLouvain

École polytechnique de Louvain

Threat Modeling with Attack-Defense Trees

Authors : **Geoffrey VAN PALM**
Supervisors : **Axel LEGAY**
Readers : **Ramin SADRÉ**, **Jean-Michel DRICOT**, **Fabien DUCHÊNE**
Academic year 2020–2021

Acknowledgements

I would like first to thank my thesis advisor Professor Axel Legay for the proposition of this subject and his help to advise me in order to realize the presented solution and this thesis.

I would like then to thank the readers of my thesis Professor Ramin Sadré, Jean-Michel Dricot and Fabien Duchêne for the time that you will be spent on reading this thesis.

Then I would like to thank my friends and my family that helped me to go through this thesis redaction and implementation, and for their support during this important and particular period for me. Without their help and their support, I wouldn't have realized this thesis in time.

Finally, I would like to have a thought for my father who left us exactly 1 month ago on the day of this thesis submission, and who always supported me and helped me when I needed it.

Contents

List Of Figures

List Of Tables

Chapter 1

Introduction

Today, at all levels of society, we use tools that allow us to organize ourselves, visualize our ideas and communicate them. Communication is a crucial aspect, which it is important to improve for the common good, even though most of the population does not realize its importance.

In cybersecurity, the theme of communication is a recurring one because it is important to explain to non-experts' people simply the risks they can incur by not protecting themselves, and by taking security as a light subject. In addition to the communication, we need to have a concrete and realistic visualization of the security in our infrastructure. The security needs to evolve as the threats are evolving, and no solution can provide total protection. That is why we need to consider that even if we are protecting against a specific threat, we are or will still be threatened by another type of attack.

In addition to the fact that we need to communicate, the exercise to explain the threats is hardened by the fact that we often must do it on complex and mature infrastructure. It is not always simple to present visually the security for an infrastructure that can have an increased complexity.

To do this, we need to use a methodology. And that is why we needed to create a tool able to create a graphical representation easily understandable by anyone and representing the interaction between the defenders and the attackers. The representation of the Attack-Defense Trees, already presented by [1] perfectly fit that need, and then is the core of this thesis. To explore the concept of threat modeling thanks to the attack-defense trees, we chose then to present the formalism and to create a solution that will be available on multiple operating systems and easily accessible.

To present our solution, we will first take a look at the origins that led to the formalism we chose to implement with the specificities of this formalism such as the terminology originally presented.

Secondly, we will see the utilities that helped us for the development of our solution and that affects its efficiency.

Thirdly, we will see the concrete solution developed, and the graphical identity we chose to use for the representation of attack-defense trees. We will go across multiple examples to see the potential of the solution and the functionalities accessible.

Finally, we will briefly imagine what were the possible improvements that could be useful for a project like this one, and what are feasible.

Chapter 2

Background

In this section, we will quickly make a remind about some knowledge that will be mentioned in this thesis, and that are important to know to understand the fully potential of the attack-defense trees. In addition to this knowledge, we will present the different tools that helped us to develop the solution.

The Tree Structure

The formalism of the attack-defense trees is based on a data structure called the **trees**.

A tree structure is represented like a tree where the roots are at the top of the graph, and where the leaves are at the bottom of the graph.
The roots and the leaves are represented as Nodes. These are connected by **edge,** that are represented by simple lines and can be visualized as the intermediary branches. There also exists other Nodes present at intermediary levels, that can be connected to one or more branches.

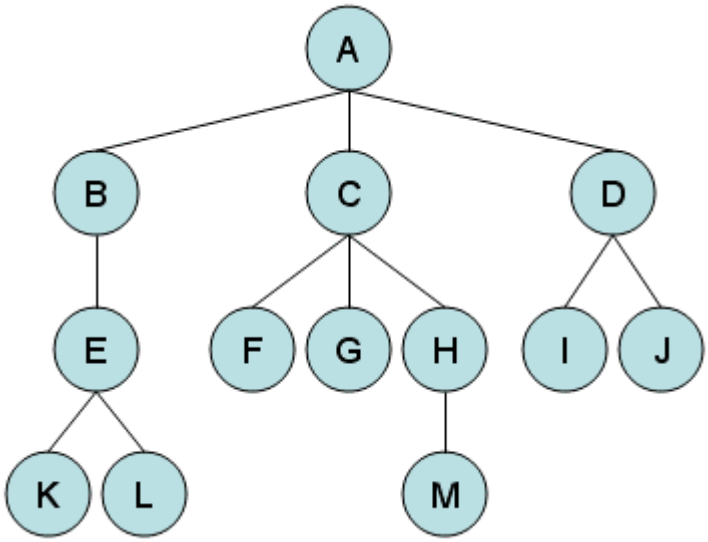


Figure 1 : Example of a Tree structure [1]

Based on the example figure above, here are different terminology used to present a tree.
The Node A is called **the root**. It's the only node that has **no parent** in the tree. No other nodes can pretend to be a root in a tree data structure.
The intermediary nodes directly connected to the root are the **children** of the root. Any child in a tree has one and only **one parent**.
Each link called **edge** is connecting only two nodes, and then if we have a tree of N nodes, we are having exactly N-1 edges.
If Nodes have the same parent, they are **siblings**.

Qt

Qt [2] is a cross-platform application development framework for embedded, desktop and mobile devices. The framework is based on the language C++ that has been extended to include functionalities thanks to signals and slots. A complete GUI interface can be created thanks to this framework.

Figure 2 : Qt Logo

Z3 Solver

Z3 solver is a SMT Solver created by Microsoft Research in 2007. It supports a variety of theories and logics, and also supports 3 types of input format that are "Simplify", "DIMACS" (for the SAT problems) and SMT-LIB.
The SMT-LIB input format will be presented later in this thesis.

Figure 3 : Z3 Solver Logo

The Z3 Solver is the SMT solver used mainly during the development of our solution.

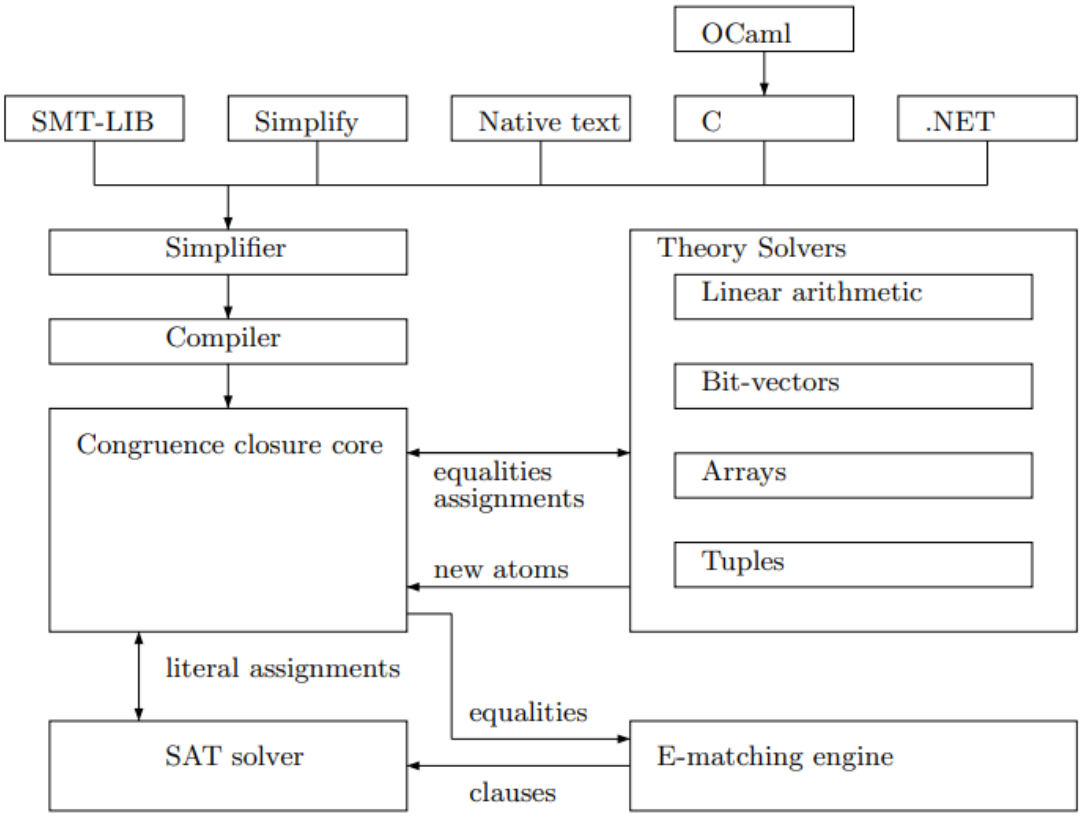


Figure 4 : Z3 Architecture Representation [3]

Chapter 3

Attack-Defense Trees

Terminology

Presented by [4], here is the base terminology that will be widely used also in this thesis to explain and explore the attack-defense trees formalism. Nonetheless, regarding the graphical representation, it won't be the one used by the developed application we built for this thesis. The graphical representation used for our solution will be represented into the Chapter 3 dedicated to the solution.

An **Attack-Defense Tree** is a node-labelled rooted tree populated with attack that can be performed by attackers and countermeasures that can be performed by defenders.

Figure 5 : Nodes Type

Each node can have 2 opposite types**: Attack** or **Defense.**

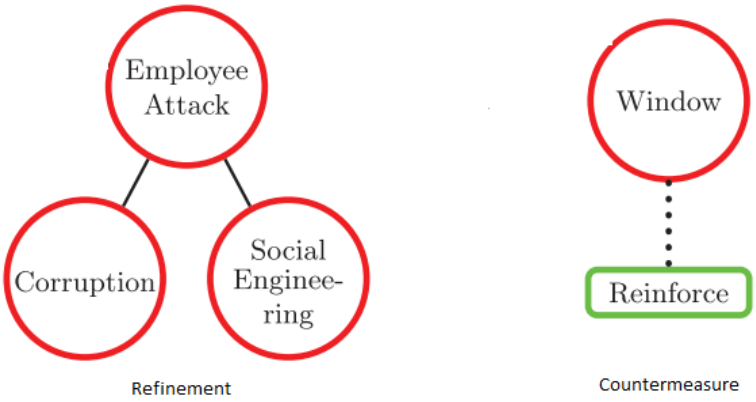


Figure 6 : Node refinement & countermeasure

Each node can be **refined** into sub-goals, that are represented by children of a node of the same type in the tree representation.
2 types of refinements are possible: **Conjunctive**, where all the children are needed to be successful to make the goal succeed and **Disjunctive**, where at least one child needs to be successful.

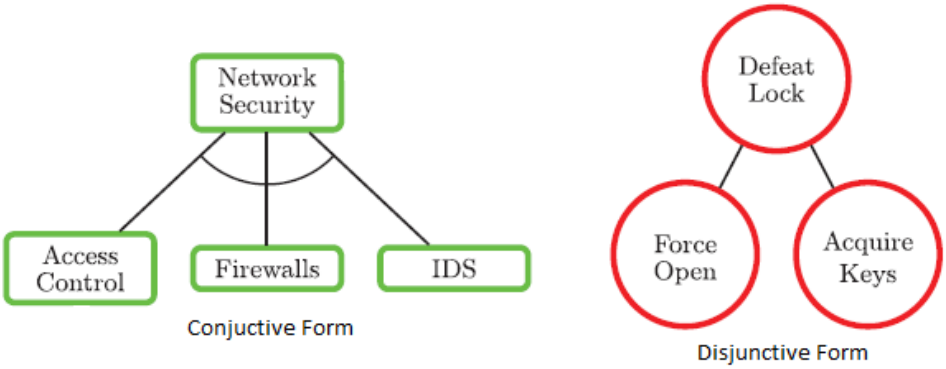


Figure 7 : Conjunctive / Disjunctive Forms

If a node has no children, it represents a basic action.
If the node has a child of the opposite type, it is representing a **countermeasure**.

As we are working with actors that are playing a two-players game, a terminology is also existing for this with the terms **proponent** and **opponent**.

In addition to the previous information, a node can also have **attributes** of multiple types that can add more dimensions to the complexity of a system. These things will be explained more in details in the next sections.

Origins

The origin of the attack-defense trees comes from the formalism of attack trees. It was first presented by Bruce Schneier in 1999 in an article present in the *Dr. Dobb's Journal*.

The idea was to model the threats against computer systems, to understand all the different paths that an attacker could use to affect the system, and to prevent these by creating efficient countermeasures.

The goal of the attack in the Figure 1 is represented as the root node of the tree. Each leaf represents an attack that can lead to the success of the complete attack.
Each intermediate node in the tree represents a sub goal. It is then possible to split a huge scenario in multiple smaller scenarios easier to understand. This formalism also has the possibility to indicate that all its children subgoals are needed to succeed the operation thanks to a small circle located under the subgoal.

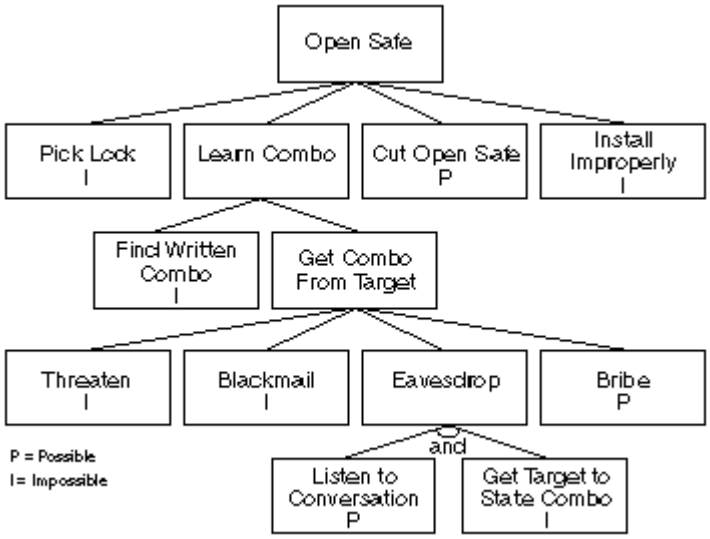


Figure 8 : Bruce Schneier's Attack Tree Representation in [4]

You certainly saw that in each intermediate nodes of the Bruce Schneier example, there are some values indicated I or P for Impossible and Possible respectively. These are some values assigned to each node, that are going to help us to define if an attack is feasible or not. Obviously, we are not limited by this type of values, and that is what he showed in another example tree, where the costs of an attack are represented. In the case the subgoal has multiple success paths, the minimum costs of its children are indicated as the cost of this subgoal.

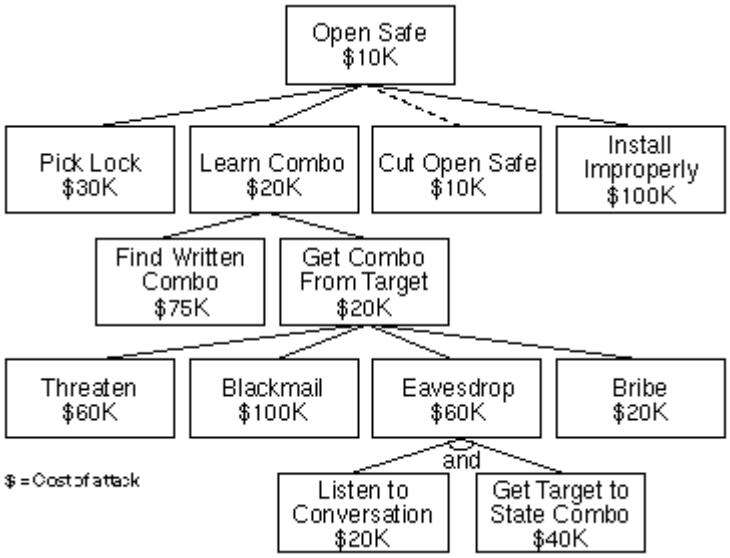


Figure 9 : Bruce Schneier's Attack Tree with Costs based on [4]

Obviously, the number of attributes of a goal or subgoal is not limited to one. The reality would need to have multiple characteristics for each subgoal that could help us to determine which countermeasures are needed to be implemented in priority. We can then imagine for example that each subgoal would have a priority or a percentage of feasibility, and thanks to this percentage and computations, we could compute the total probability percentage of an attack vector.

In addition to this, you certainly noticed that Schneier were only graphically representing the attacks, and not the countermeasures that could have been implemented. Due to the countermeasures, the characteristics of the subgoals would change and then maybe the decision. Moreover, the implementation of a countermeasure will also be the target of a new threat, that can also change the probability of a threat. That is the reason why the Attack-Defense trees formalism has been considered and developed.

From Tree-Graph to Logical representation

Every graph to be interpreted into a computational form needs to be translated into a logical sentence, that will be used later into solvers.
In order to do this translation, we need to formalize the logic behind the trees.
Fortunately, the tree representation does not need a lot of comprehension to be translated, as the tree itself works with the idea of logic operator AND, OR and NOT.

Every Attack-Defense tree has a collection M of **nodes** m, such as $m \in M$.
Every node m has an attribute **child type** $ct \in CT$, where $CT = \{\wedge, \vee\}$ (respectively AND and OR)
Every node m also has an attribute **type** $t \in T$ where $T = \{ATT, DEF\}$.
Each node has a collection of **children** $c \in C$; such as $C \subseteq M$. If a node has no children,

it is called a **leaf**. The leaves will be the components of the formulas we will inspect later. All the intermediary nodes just have context interest, and are not helping logically to find a solution.
If a node m has a type t different than the type of its children c, an operator NOT is added to the formula.

Example of Attack-Defense Tree conversion to Logical Form

Thanks to these, we can transform a complete simple Tree (without the additional attributes we can assign to it such as the cost) into a logical expression.
For example, if we take the following graph representing the attack vectors to perform an intrusion into someone's home:

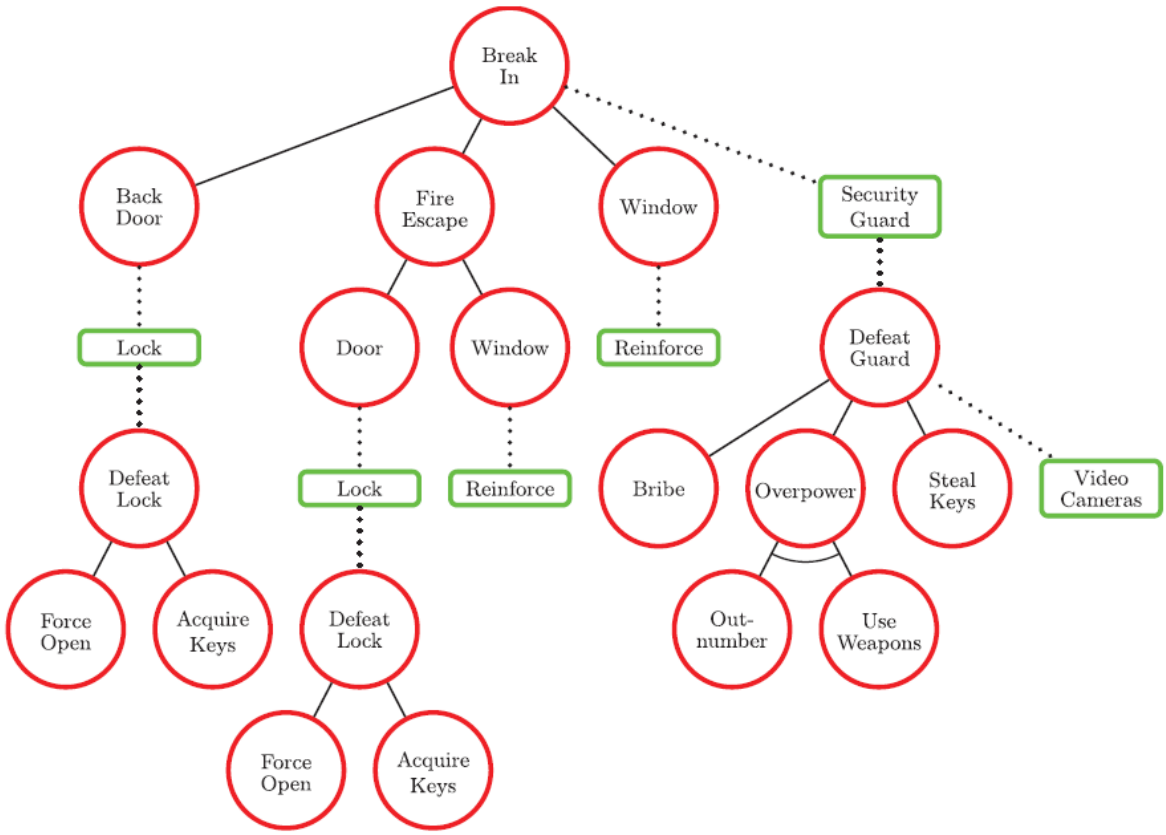


Figure 10 : Attack-Defense tree of home intrusion represented in [4]

We can convert it into a logic formula thanks to the following steps:

Step 1: Start from the root

$$\{Break\ in\}$$

Step 2: Replace the root node with its children, and add the operator mentioned as the child type of your node.

$$(Back\ Door \vee Fire\ Escape \vee Window \vee -(Security\ Guard))$$

You can see that here we are using the operator ∨ (OR) as we are not having a circle arc under the node and the operator – (NOT) due to the fact that the type of our node "Break In" and the type of its children "Security Guard" are not the same. It is the presence of a countermeasure.

Step 3: Redo for all the nodes present in the formula, until there are no more nodes with children

$$(-Lock \vee (Door \vee Window) \vee\ -(Reinforce) \vee\ -\big(-(Defeat\ Guard)\big))$$

$$(-\big(-(Defeat\ Lock)\big) \vee \big(-(Lock) \vee\ -(Reinforce)\big) \vee -(Reinforce) \vee (Bribe \vee Overpower \vee Steal\ Keys \vee -(Video\ cameras))$$

$$\Big(-\big(-(Force\ Open \vee Acquire\ Keys)\big)\Big) \vee \Big(-\big(-(Defeat\ Lock)\big) \vee\ -(Reinforce)\Big) \vee -(Reinforce) \vee -(-(Bribe \vee (\ Outnumber \wedge\ Use\ Weapons\ ) \vee Steal\ Keys \vee -(Video\ Cameras)))\}$$

$$\Big(-\big(-(Force\ Open \vee Acquire\ Keys)\big)\Big) \vee \Big(-\big(-(Force\ Open \vee Acquire\ Keys)\big) \vee\ -(Reinforce)\Big) \vee -(Reinforce) \vee -\Big(-\big(Bribe \vee (\ Outnumber \wedge\ Use\ Weapons\ ) \vee Steal\ Keys \vee -(Video\ Cameras)\big)\Big)$$

A potential suggestion that could come to mind is the possibility to simplify the logical form of our tree into a shorter form, where all the "Force Open" and "Acquire Keys" would be considered as one.
Unfortunately, it would not be a good idea. In the case you have not specified it to the tree creator, the tree creator could use the same pattern to achieve a subgoal but in different context. Here for example, we are trying to acquire keys of a front door or a back door, and the context about the keys is not mentioned. We can imagine that the keys of the front door are way more protected than the one of the back door, and then the computation of the tree is also different depending of the subgoal we are trying to achieve.

What to do with this formula?

Now that we have a Boolean logic form of our attack-defense tree, we will have to use it properly to extract the potential attack vectors to reach a specific goal. In order to do so, we will use a SAT solver to solve a satisfiability problem. The concept of SAT solver and satisfiability problem will be presented in the Chapter 4 describing our solution.

As previously mentioned, we can convert “simple” trees into a Boolean logical formula, but the attributes such as the cost, the percentage of feasibility, etc. are not considered under this form. It is not a problem, because the computation of those is just an upper layer that will cover the basic Boolean form, improving the capabilities of our algorithm. We can see that as a process coming after the computation of the solution from a SAT solver.

Chapter 4

SAT Problem

To present our solution, we first need to present the functionalities our program is based on. As we mentioned earlier in this document, we are working with a graphical representation of a tree and we need to compute something based on this tree. To be able to compute efficiently the potentials attack vectors, we transformed our tree into a Boolean logical form. But even with this kind of formula, if we are facing very large tree with large amounts of branches and children, the way to compute the formula will be important because we can face exponential problem computation and then waste large amount of time due to the complexity of the computation.

To avoid this kind of problem, we need to refine our problem into a Satisfiability problem.

Figure 11 : Example of SAT solving [3]

Presentation and Terminology

A satisfiability problem, also called **SAT** problem, is a decision problem based on the propositional logic to establish if a formula is **satisfiable** or not. It has multiple applications such as decision problem, planification problem, diagnostics …

A formula is **satisfiable** if a solution is existing for the problem.
To perform this satisfiability check thanks to a SAT solver, we need to use a specific form of formula called the **CNF** (Conjunctive Normal Form) where the formula is a conjunction of clauses.

A **clause** is a disjunction of the variable considered in the problem. In the SAT vocabulary, a variable is a **literal**.

Operators in Propositional Logical

As we are going to work with propositional logic, we provide you some specifications about the propositional Logic and what it implies.

Operator NOT

Table 1 : Truth Table Operator NOT

Literal Value	Literal Value with NOT operator
0	1
1	0

As its name is mentioning, the "NOT" operator is reverting the result of a value. In our application, the not operator is included into the formula when a node and its children have 2 different types: Attack and Defense.

Regarding the notations used commonly, " ¬ " is the symbol used to indicate the negation. To simplify some work, we also use the symbol " – ".

Operator AND

Table 2 : Truth Table Operator AND

Literal Value A	Literal Value B	A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

In this case, this operator makes sense when two or more literals are concerned. It will be the case for example when we have a conjunctive form of nodes. In other words, it's when we have to accomplish all the subgoals determined after a goal refinement.

About the notation, the symbol ∧ is commonly used.

Operator OR

Table 3 : Truth Table Operator OR

Literal Value A	Literal Value B	A ∧ B
0	0	0
0	1	1
1	0	1
1	1	1

As we were defining the operator AND previously as the one indicating that we have to accomplish **all** the subgoals of an upper goal, here the operator "OR" is indicating that we only have to accomplish **at least one** of the subgoals defined.
The symbol commonly used for the operator "OR" in the propositional logical is " ∨ ".

Operator Imply

This operator is a bit different than the others and has a specificity. The "Imply" operator could be translated like "If Literal A is true, then Literal B must be true".
This operator is commonly represented like an arrow " → ".

Table 4 : Truth Table Operator Imply

Literal Value A	Literal Value B	A → B
0	0	1
0	1	1
1	0	0
1	1	1

Regarding the specificity of the "Imply" operator, you can ask yourself about my translation of the operator, what happens when the literal A is false ?
The question is easily answered : If A is false, all the cases are true.

Operator Double Implication

Table 5 : Truth Table Operator Double Implication

Literal Value A	Literal Value B	A → B
0	0	1
0	1	0
1	0	0
1	1	1

Here the operator “double implication” could have been translated into this sentence : “If Literal A is True, then Literal B must be True AND If Literal B is True, then Literal A must be True” or in other words : “Literal A and Literal B must have the same value”.

Boolean Formula to CNF Formula

In order to perform the transformation of a formula of any form to a Conjunctive Normal Form, we need to use a transformation process able to create this formula of this conjunctive form and to keep the equisatisfiability between the two formulas.
These processes are often based on inference rules, that have been proved and demonstrated. Here regarding the goal of this report, we will only present the rules, but we won’t demonstrate these.
Eleven inference rules can be used to perform the transformation of our formula into a CNF formula.

Regarding the solution we chose to implement, I based my transformation process on the one described by Professor Jason Eisner in his course [4].

The pseudo-code created will be presented in the chapter where I present the solution. For the moment, we will go through the three transformations law that are used by this process and that are commonly used to obtain CNF formulas.

De Morgan’s Laws

The first set of transformation rules we are seeing is called the “De Morgan’s Laws”. These transformation rules explain how an operator “NOT” change the behavior of a formula.

Here are the two transformations possible :

If we have a negation applied to a conjunction of literals/clauses, then the result is the disjunction of two negation literals or clauses :

$$\neg (A \wedge B) \leftrightarrow \neg A \vee \neg B$$

If we have a negation applied to a disjunction of literals/clauses, then the result is the conjunction of two negation literals or clauses.

$$\neg (A \vee B) \leftrightarrow \neg A \wedge \neg B$$

The equisatisfiability can be proven thanks to the truth tables below :

Table 6 : Truth Table De Morgan's Law #1

A	B	$A \wedge B$	$\neg(A \wedge B)$	$\neg A$	$\neg B$	$\neg A \vee \neg B$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Table 7 : Truth Table De Morgan's Law #2

A	B	$A \vee B$	$\neg(A \vee B)$	$\neg A$	$\neg B$	$\neg A \wedge \neg B$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

These transformation rules will be widely used because when we are converting a formula into a CNF formula, we must put the negation at the deeper level possible (the level the nearest from the leaf of our tree / literals in our logical sentence).

The Distributive Laws

Often used into the mathematical domain, we have normally all already used these laws to perform multiplication for example.

Thanks to these laws, we will be able to transform some conjunctions into clauses , perfect for our Conjunctive Normal Form formula.

Here is the law described :

$$A \wedge (B \vee C) \leftrightarrow (A \wedge B) \vee (A \wedge C)$$
$$A \vee (B \wedge C) \leftrightarrow (A \vee B) \wedge (A \vee C)$$

Again, we can see that thanks to the truth table below that the two formulas are equisatisfiable.

Table 8 : Truth Table Distribution #1

A	B	C	$B \vee C$	$A \wedge (B \vee C)$	$(A \wedge B)$	$(A \wedge C)$	$(A \wedge B) \vee (A \wedge C)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Table 9 : Truth Table Distribution #2

A	B	C	$B \wedge C$	$A \vee (B \wedge C)$	$(A \vee B)$	$(A \vee C)$	$(A \vee B) \wedge (A \vee C)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

The eleven rules of Transformation

About the transformation process we chose to implement; eleven rules are used to create the CNF formula from the well-formed formula.

The Base cases

Two rules are considered as the base one : it's the one managing directly the literals.

$$CNF(p) \equiv p$$
$$CNF(\neg p) \equiv \neg p$$

The inductive cases

In the next lines, we will present you the 9 rules of transformation. The following symbol $\otimes$ will be used and is meaning that a distribution between the two formulas will be performed.

Transformation of **an implication**

$$CNF(A \to B) \equiv CNF(\neg A) \otimes CNF(B)$$

Transformation of **a conjunction**

$$CNF(A \wedge B) \equiv CNF(A) \wedge CNF(B)$$

Transformation of **a disjunction**

$$CNF(A \vee B) \equiv CNF(A) \otimes CNF(B)$$

Transformation of **a double implication**

$$CNF(A \to B) \wedge CNF(B \to A)$$

Transformation of **a double negation**

$$CNF(\neg\neg A) \equiv CNF(A)$$

Transformation of **a negation on an implication**

$$CNF\big(\neg(A \to B)\big) \equiv CNF(A) \wedge CNF(\neg B)$$

Transformation of **a negation on a conjunction**

$$CNF\big(\neg(A \wedge B)\big) \equiv CNF(\neg A) \otimes CNF(\neg B)$$

Transformation of **a negation on a disjunction**

$$CNF\big(\neg(A \vee B)\big) \equiv CNF(\neg A) \wedge CNF(\neg B)$$

Transformation of **a negation on a double implication**

$$CNF\big(\neg(A \leftrightarrow B)\big) \equiv CNF(A \wedge \neg B) \otimes CNF(\neg A \wedge B)$$

Chapter 5

SMT Problem

We talked about the SAT decision problem, base of our solution. But a problem quickly comes in mind.
With SAT, we are working with literals that are in fact Boolean values. According to the formalism of attack-defense trees, each leaf of our tree should be able to contain also attributes. These attributes are obviously not simply Boolean values, such as the cost of an operation that would require to perform an attack or even the percentage of success of a goal.

We then have to find a way to work with values such as integers or float numbers, and to compute quickly if a solution exists or not. That's why we used a SMT (Satisfiability Modulo Theories) solver into our solution.

The purpose of a SMT solver is to tell if a problem modelized thanks to a logical formula and theories is satisfiable. We will then in this section present how is working a SMT solver, what it is using, and the challenges encountered during the development of our solution.

An SMT problem

Satisfiability Modulo Theories is , as presented [5], a domain of automated deduction studying the methods to check the satisfiability of first-order formulas as we are working for the SAT solver in addition of what has been called the **theories**.

Here is an example of the syntax we will be using to specify our problem and that will be submitted to a SMT solver.

```
(set-logic AUFNIRA)
(declare-const RootNode1 Int)
(declare-const RootNode2 Int)
(declare-const RootNode3 Int)
(declare-const RNP1 Real)
(declare-const RNP2 Real)
(declare-const RNP3 Real)

(assert (<= RootNode1 1))
(assert (>= RootNode1 0))
(assert (<= RootNode2 1))
(assert (>= RootNode2 0))
(assert (<= RootNode3 1))
(assert (>= RootNode3 0))

(assert (<= RNP1 1))
(assert (>= RNP1 0))
(assert (<= RNP2 1))
(assert (>= RNP2 0))
(assert (<= RNP3 1))
(assert (>= RNP3 0))

(assert (>= (+ RootNode1 RootNode2 RootNode3) 1))
(assert (=> (= RootNode1 1) (= RNP1 0.5)))
(assert (=> (= RootNode1 0) (= RNP1 1)))
(assert (=> (= RootNode2 1) (= RNP2 0.6)))
(assert (=> (= RootNode2 0) (= RNP2 1)))
(assert (=> (= RootNode3 1) (= RNP3 0.7)))
(assert (=> (= RootNode3 0) (= RNP3 1)))
(assert (>= (* RNP1 RNP2 RNP3) 0.7))

(check-sat)
(get-model)
```

Figure 12 : SMT Input Format

This example is defining the following problem :

If at least one of the three variables (RootNode1, RootNode2 and RootNode3) having respectively the following probabilities 50%, 60% and 70% are active, and that the product of the probabilities are above 70%, the model is satisfiable, and a solution will be returned.

Let's analyze each part of this example.

```
(set-logic AUFNIRA)
(declare-const RootNode1 Int)
(declare-const RootNode2 Int)
(declare-const RootNode3 Int)
(declare-const RNP1 Real)
(declare-const RNP2 Real)
(declare-const RNP3 Real)
```

Figure 13 : SMT Logic Declaration & Variable Declaration

First we have the declaration of the variable and of the context that we are going to use. The first line is the declaration of the logic that we are going to use for this problem. The logic will be defined later in this thesis. It will be always the same logic as we don't intend to change the types of the variables and attributes implemented in the solution. Then we have 3 lines dedicated to the declaration of the variables.
For the leaves of our attack-defense tree, we are defining it as Integers, where it was defined as Booleans into the SAT formulas. As Booleans actually are similar to an assignation of value 0 or 1, it was easier for the computation of the attributes to consider these as Integers.

After these 3 lines, we have 3 other lines that are defining the attributes of each variable. Depending on the number of attributes we want to consider into a problem, the number of the lines written here will be equal to ***Number of Attributes * Number of Variables***. Here in that case, we only have 1 attribute corresponding to the probability of an event to occur (or a variable to succeed).

```
(assert (<= RootNode1 1))
(assert (>= RootNode1 0))
(assert (<= RootNode2 1))
(assert (>= RootNode2 0))
(assert (<= RootNode3 1))
(assert (>= RootNode3 0))

(assert (<= RNP1 1))
(assert (>= RNP1 0))
(assert (<= RNP2 1))
(assert (>= RNP2 0))
(assert (<= RNP3 1))
(assert (>= RNP3 0))
```

Figure 14 : Boundaries of Variable

On this part of the screen, we are defining that the leaves (originally Booleans) have a value between 0 and 1. And as these have been declared as Integers, only 2 possible values will be assigned : 0 and 1.
For the other lines, we are defining that the values of the attribute's "*percentage*" needs to be between 0 and 1. Here, as they have been declared as Reals, an infinite number of assignation is possible, as long it's between 0 and 1 included.

```
(assert (>= (+ RootNode1 RootNode2 RootNode3) 1))
(assert (=> (= RootNode1 1) (= RNP1 0.5)))
(assert (=> (= RootNode1 0) (= RNP1 1)))
(assert (=> (= RootNode2 1) (= RNP2 0.6)))
(assert (=> (= RootNode2 0) (= RNP2 1)))
(assert (=> (= RootNode3 1) (= RNP3 0.7)))
(assert (=> (= RootNode3 0) (= RNP3 1)))
(assert (>= (* RNP1 RNP2 RNP3) 0.7))
```

Figure 15 : Formula and Attributes Value

Here we are exploring the part specific to the formula definition.
Each clause that we created by transforming the formula into a Conjunctive Normal Form one is taking a line. As the formula here presented is quite simple, it's taking only one line.
Starting at the second line and going to the seventh line, we are having implications that we did not transform as it was to consider the attributes. It's a kind of assignation that we needed to perform, that will be mentioned in the difficulties we encountered later in this thesis.
And the last line is the computation of the complete percentage value in this case.

```
(check-sat)
(get-model)
```

Figure 16 : Results and Model

Those two lines are important and will always appear in input file of the solvers. The first one is checking if a satisfiable solution exist, and if that's the case, the model with the value of each variable will be specified.

The Theories

These theories are in other words what types of data we can consider during the solving of a decision problem. The functions or the operators that the theory is including are also presented on the website of SMT-LIB. SMT-LIB presented the following list of theories :

- ArraysEx

  With this theory, you can use Functional Arrays into your problem modeling.

- FixedSizeBitVectors

  With this one, working with bits vectors of a fixed size is possible.

- Core

  Define the basic Boolean operators, used with first-order logical formula. As its name indicate it, it will be commonly integrated in all the logics that we will present after the theories.

- FloatingPoint

  Thanks to this one, you will have the ability to use Floating Point numbers in the

conditions of success of your decision problem.

- Ints

  Here is added the possibility to work with integers.

- Reals

  This theory consider the real numbers as operands.

- Reals_Ints

  Theory that mix the usage of Reals and Integers.
  It will be one of the most interesting theory in the case of our solution, as we will be working with Integers such than cost value and reals such than percentage of probability for example.

- Strings

  Consider Unicode character strings and regular expressions.

All these theories aren't used directly in the input file, as you can see in the example we made earlier. They are combined into another specificity of the logic, called the **Logics**.

Logics

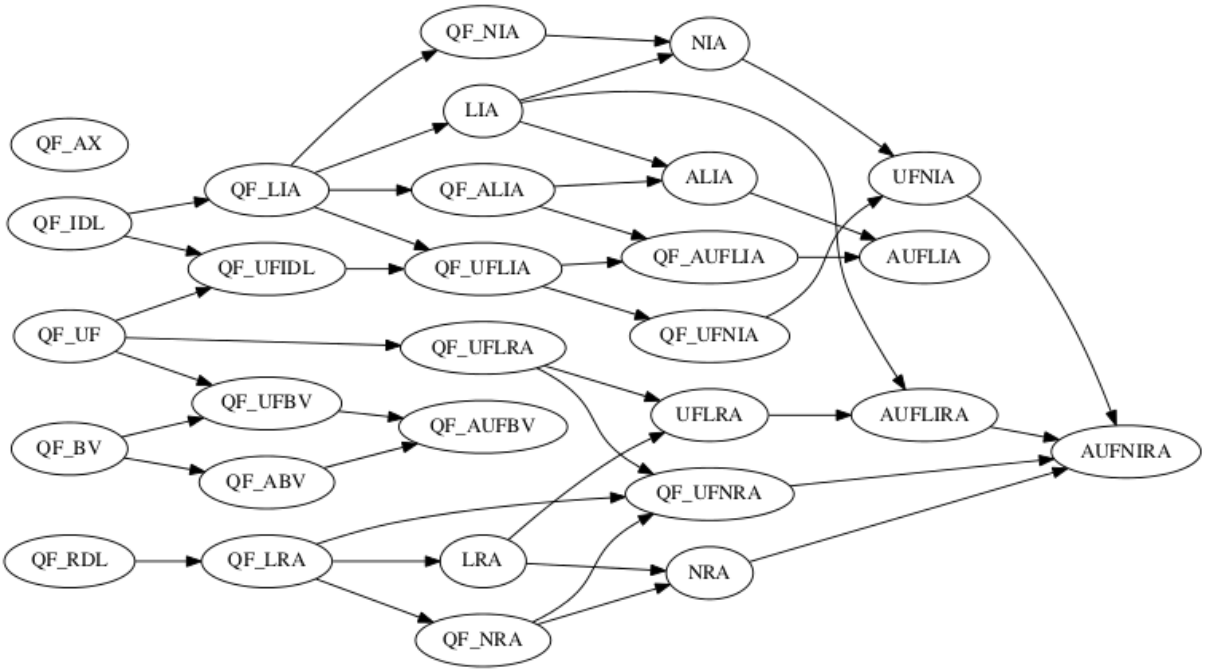


Figure 17 : Logics dependencies

In the graph just above are presented the Logics that are usable by the SMT solver. Each relation presented in this graph means that if a formula of Logic L1 is linked to a Logic L2, all the formulas that are fitting L1 are fitting the L2 Logic.

We will go across a non-exhaustive list of the logics available, but first, here is a presentation of the letters and their meanings to :

- QF is specified to mention that we are working with Quantifier Free Formula.
- A and AX are using the theory ArraysEx
- BV is mentioned for FixedSiteBitVectors theory
- FP (not present in the graph for the moment) is specified for the theory Floating Point
- IA is mentioned for the theory Ints, relating to the Integers Functions. (Integer Artihmetics)
- RA for the theory Reals (Reals Arithmetic)
- IRA for the theory mixing the Reals and the Integers (Integer Real Arithmetic)
- IDL used for the Integer Difference Logic
- RDL for Rational Difference Logic
- When L is specified before IA, RA or IRA, it's used for linear arithmetic
- When N is specified before IA, RA or IRA, it's used for Non-linear arithmetic
- UF is an extension allowing the sort and functions symbols

Here is a short list presenting some of the logics available :

- AUFLIA

  As its name indicate, thanks to the letters explanation we saw earlier, we can see that this logic is using respectively the ArraysEx, allowing the sort and functions symbols with Linear arithmetic on Integers.

- AUFLIRA

  Same than the previous logic except for the last part, that is using linear arithmetic on Reals and Integers.

- AUFNIRA

  Usage of non-linear arithmetic on Reals and Integers, with the usage of ArraysEx Theory.

  Based on the graph that has been presented earlier, you can see that this theory is the one with the reals and integers containing most of the functions developed for the other logics. That's why as a first approach we use this logic.
  Regarding the type of the attributes we could specify, some changes about the logic considered for a problem would be interesting as an upgrade of our solution.

Chapter 6

The Solution

In this chapter, we will present the solution we developed to provide a tool simplifying the creation of attack-defense trees and usable by anyone, even non-security aware users or non-IT persons.

First, we will then present the interface and explain each aspect of the interface, with the explanation of the commands and actions available on the solution.

Then we presented the graphical formalism we chose to use in our solution.

After that, we will then see what are the settings available and how can it be set up.
A brief presentation of the possibilities will be done.

After that, we will go through the functionalities more deeply by presenting pseudo-codes about the algorithms we implemented into the solution, and how has been done the implementation.

Finally, a section about the problems encountered is providing you all the information about the problems we had to manage and how we managed it.

The interface and the interactions

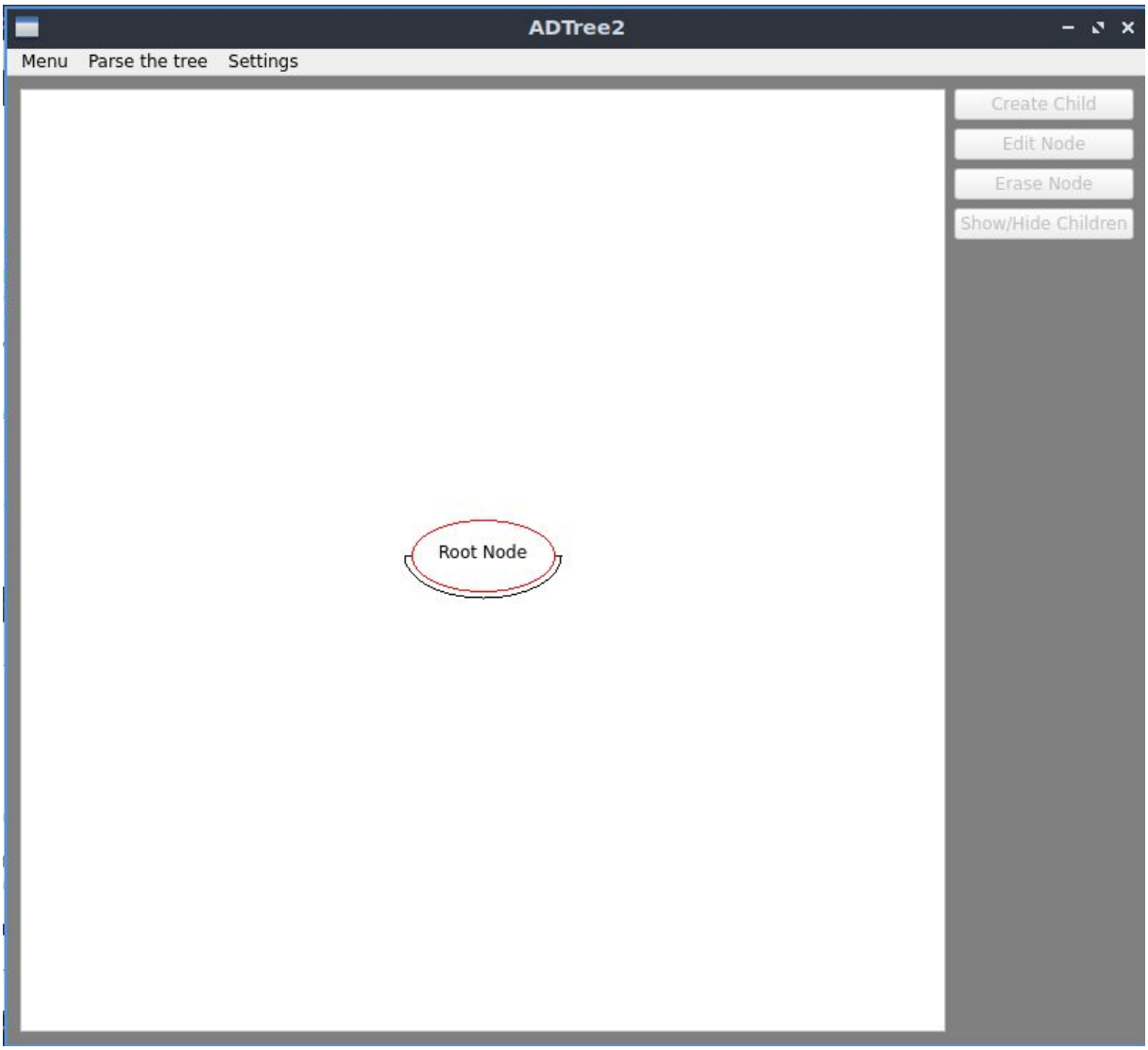


Figure 18 : Solution Main Interface

Here is the main interface that you will see if you open our solution.

On the left, we have the canvas where the tree will be presented and modified. Each node is selectable and editable thanks to the menu present on the right of the main window. At the startup of the solution, a root node is automatically created, that can change of properties.

On top of the window, three actions are possible :

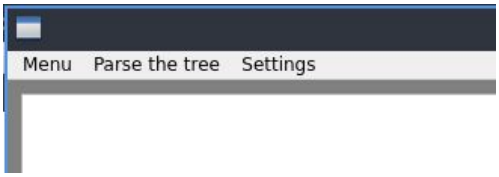


Figure 19 : Top Menu of Solution

- Menu

  In this menu, you can save the configuration of your solution, such as the constraints of the attack nodes, the constraints of the defense nodes and the SMT solver path. In addition to these, you have the possibility to save and load the attack-defense trees.

- Parse the tree

  With this menu, you compute the solutions available for your attack-defense tree. It will open a new window where you can indicate what are the conditions for a solution to be considered as successful.

  The number of constraints showed here depends on the constraints configured in the project thanks to the settings menu presented later.

Figure 20 : Solution parameters Window

- Settings

  In the settings menu, you have the possibility to first set up the path of the SMT Solver you want to use. To be compatible with our solution, the SMT solver must be compatible with the SMT-LIB input file format.
  In addition to the path of the SMT Solver, the constraints applicable for the attacking nodes are customizable with the possibility to mention the boundaries of each constraint, the default value, the unit and the type of computation applied to the constraints (product for the probability, and addition for the costs for example). The constraints for the defending nodes are defined the same way but are separated from the attacking nodes, as the constraints that would be interesting for the solutions could be different.

On the right of the interface, we have the node menu.
Activated only when a node is selected , after a double click on a node.

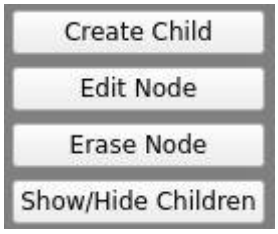


Figure 21 : Right menu of Solution

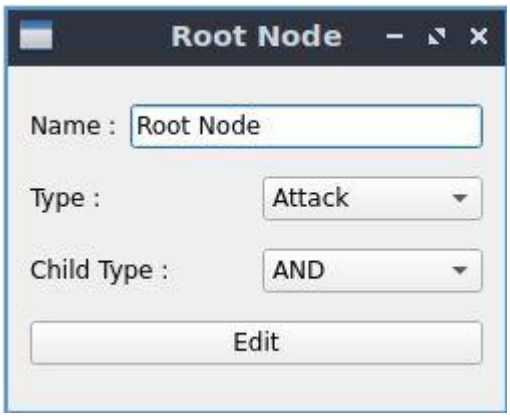
Here 4 actions are possible :

- Create Child

  Create a new node, that will consider the actual selected node as its parent. The parameters of the parent of the node are also filled into this child.

- Edit Node

  With this action, you have the possibility to modify the information of a node, such as the name, the type of the node, its refinement type and the constraints values in the case of a leaf.



- Erase Node

  With this action, the suppression of the node selected is done. All the children of this node are also suppressed from the tree, such as all the attributes values that could have been assigned.

- Show/Hide Children

  To choose to present only a part of the tree, we often do not need to have all the information that are presented in the other branches. To simplify a presentation purpose, you can thanks to this action collapse a complete branch of a tree by hiding all the children of a node.

The graphical identity

Regarding the graphical identity of our solution, here is the way we chose to represent the nodes. We chose to have a graphical identity as simple as possible to avoid an overload of graphical representation in the case of a rebuilding of a tree.

First, about the shape of the nodes, we are keeping an oval shape for the attacking nodes and for the defensive nodes. The only thing differing is the shape of the border line and its color.

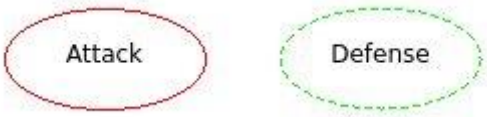


Figure 22 : Attack and Defense Representation

For the refinement of a goal, we simply choose to represent the relation between two nodes by linking them together with a straight line.
In the case of a countermeasure introduced in the tree, here the representation of the relation matches the representation of the node. We have then a dotted green line that is connected to the goal.

Figure 23 : Refinement & Countermeasure

Regarding the refinement type of a goal, we have then the conjunction and disjunction. The representation of a conjunction is made thanks to an arc of circle represented below the goal. The absence of this arc of circle represents a disjunction.

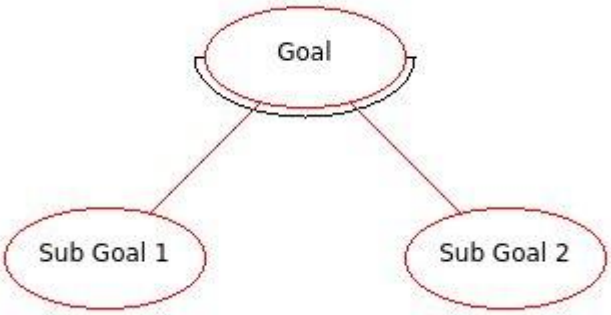


Figure 24 : Conjunctive Refinement

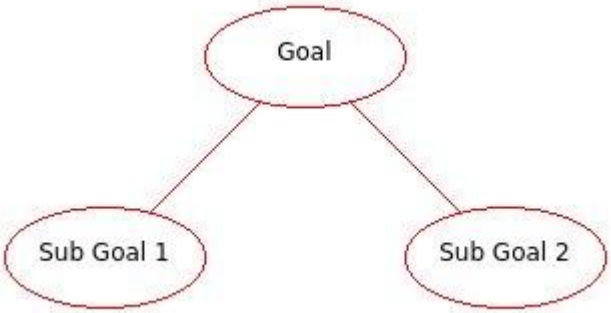


Figure 25 : Disjunctive Refinement

About the results found in a tree, a window presenting the different solution with the values of the constraints is shown and the leaves that made this solution satisfiable are filled in green.

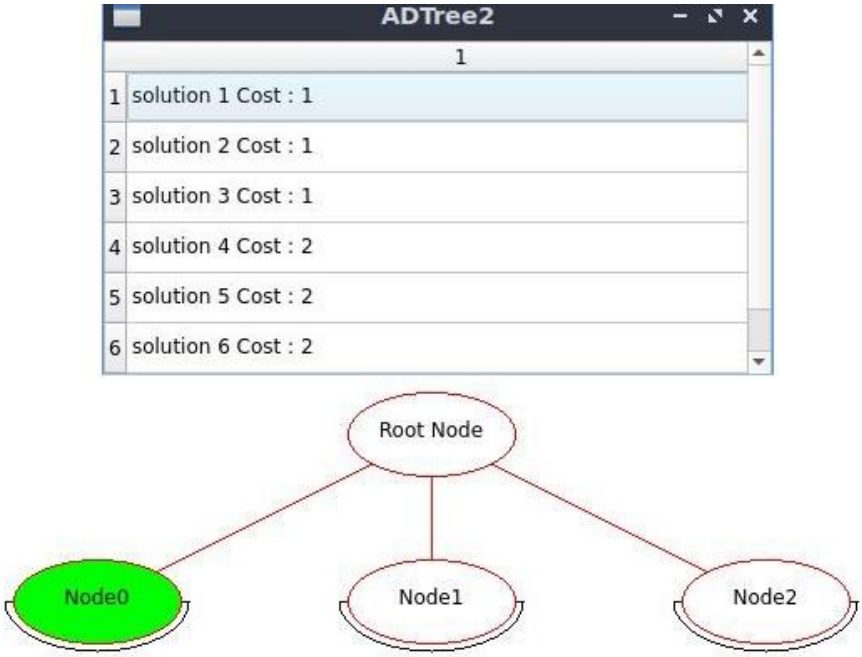


Figure 26 : Results Presentation

The settings

We will here present the settings and see how to configure your solution.
Currently the settings possibilities are limited to three main actions :

- Setting up the SMT Solver Path

  Our solution can perform the computation with any SMT solver as long as the solver accept the SMT-LIB input format. To do this, we then let the user install on its machine the SMT solver and inform the path of this one to the solution. Thanks to this type of configuration, anybody can perform a comparative of SMT solver performance or choose to use its own preferred method.

- Setting up the constraints of the attacking nodes

  Here is the configuration of the attributes contained by the nodes. You can define the name of the constraint, its boundaries, its default value and the type of the constraint : PRODUCT or ADDITION. That type is mentioned for the computation of the tree when there is a need to compute a sum or product of constraints, such as the product of percentages.

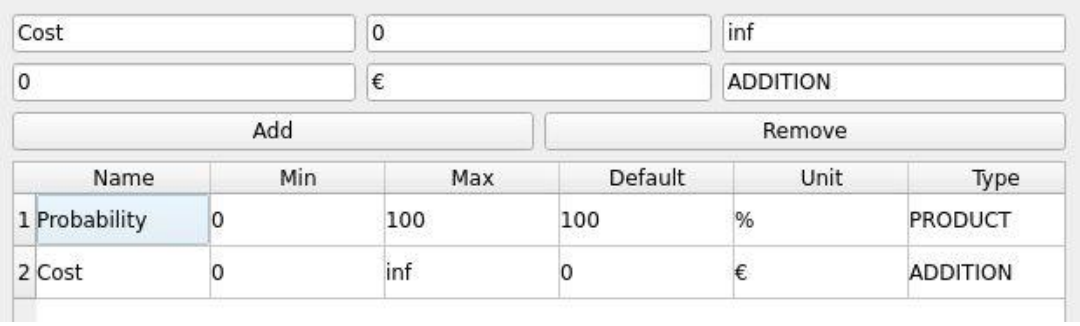


Figure 27 : Settings Window

- Setting up the constraints of the defensive nodes

  Between the configuration of the attacking nodes and the defensives nodes, there is not a lot of configuration aspects that are changing. Every step mentioned in the section just above are valid here.

The functionalities

In this section, we will present some pseudo-code of the functionalities we developed to create this solution.

To be able to construct the SAT sentence , transformed into a CNF formula, we needed to parse the tree into a sentence. To do so, we need to perform a Depth First Search thanks

to the objects ANDFormula, ORFormula and NOTFormula, inheriting from the object Formula.

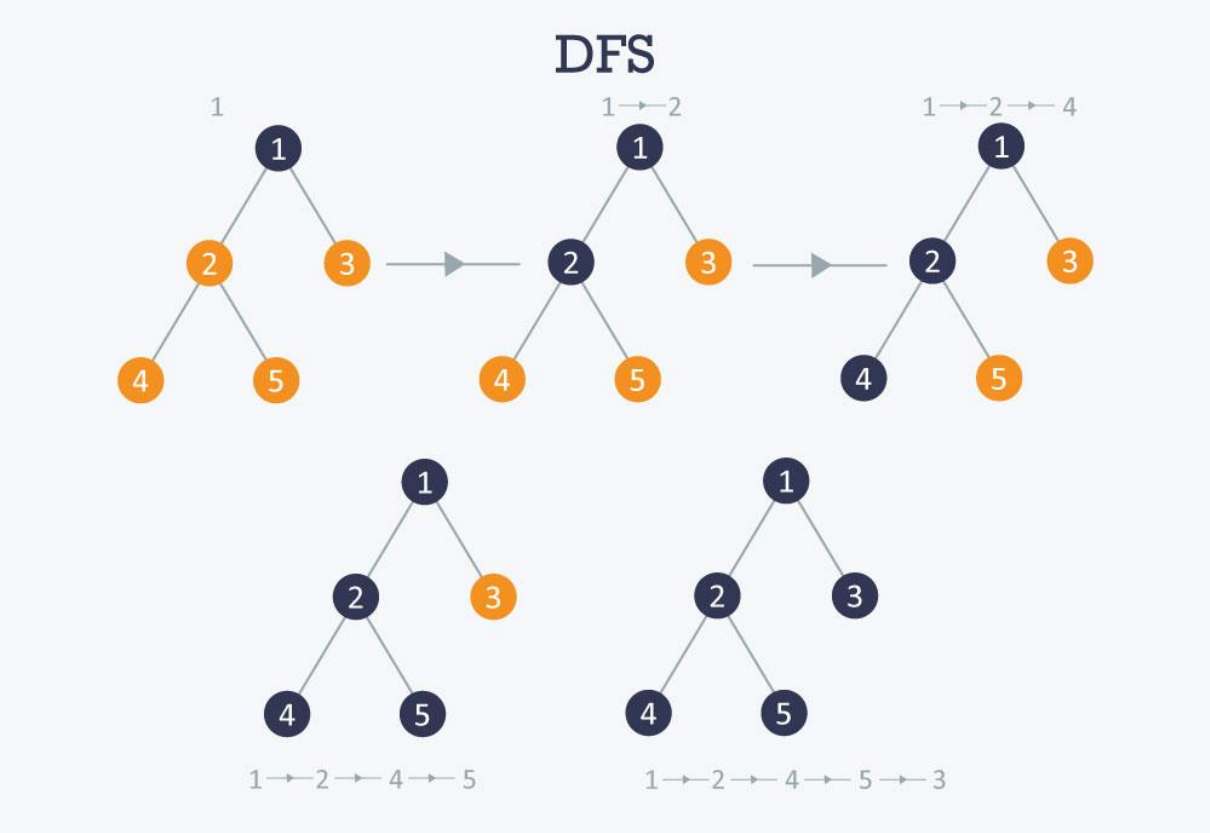


Figure 28 : Depth First Search Algorithm [9]

```
Function GetLogicFormula() {

        LET N ← This node
        LET FA ← ANDFormula(N.name)
        LET FO ← ORFormula(N.name)

        FOR child ∈ N.children{
                LET CF ← child.GetLogicFormula()
                FA.add(CF)
                FO.add(CF)
        }

        IF ∃ N.parent AND N.type ≠ N.parent.type{
                LET FN ← NOTFormula();
                IF N.child_type = AND → FN.add(FA);
                IF N.child_type = OR → FN.add(FO);
                return FN
        }
        ELSE{
                IF N.child_type = AND ➔ RETURN FA
                IF N.child_type = OR ➔ RETURN FO
        }
}

Function add(child){
        // Add a child to the children list
}
```

Thanks to these lines, we have now an object of type Formula, that contain all the architecture of the tree and is derived into ANDFormula, ORFormula and NOTFormula object. We can convert with this object the tree into a first-order logic formula matching the CNF requirements with our function **get_CNF**.

Depending on the type of the node, this function is converting the formula into the CNF one thanks to the conversion methods we presented into Chapter 4.

```
Function ANDFormula::get_CNF(){

        LET N ← This Formula Object
        LET NO ← ANDFormula(N.name);

        FOR child ∈ N.children{
                LET TF ← child.get_CNF()
                NO.add(TF)
        }
        RETURN NO
}

Function ORFormula::get_CNF(){

        LET N ← This Formula Object
        LET NO ← ANDFormula(N.name);
        LET NCA ← Array of Formula Object
        LET RA ← Array of array of Formula
        LET COM ← Array of Formula

        FOR child ∈ N.children{
                LET TF ← child.get_CNF()
                NCA.add(TF)
        }
        this->distribute(NCA, COM, RA);
        FOR result ∈ RA {
                ORF ← ORFormula(N.name)
                FOR RC ∈ result{
                        ORF.add(RC)
                }
                NO.add(ORF)
        }
        RETURN NO
}
Function NOTFormula::get_CNF(){
        LET N ← This Formula Object
        IF N.child_type = N.child.child_type ➔ RETURN N.child.child.getCNF();
        ELSE {
                N.child.negate()
                IF N.child.type = N.type ➔ RETURN N.child
                ELSE ➔ RETURN N.child.getCNF()
        }
}
```

```
Formula distribute(FS, VC, RE){ // Where FS is the set of formulas, VC is an array for the computation and RE are the results

        LET N ← This Formula Object
        IF |FS| = 0 ➔ RE.add(VC)
        ELSE{
                LET FSL ← FS[0].get_children()
                LET NF ← Array of Formula Object

                FOR formula ∈ formulas{
                        NF.add(formula)
                }

                IF |FSL| = 0 {
                        VC.add(FS[0])
                        N.distribute(NF, VC, RE)
                        VC.remove(FS[0])
                }
                ELSE{
                        FOR child ∈ FSL{
                                VC.add(child)
                                N.distribute(NF, VC, RE)
                                VC.remove(child)
                        }
                }
        }
}

Formula ANDFormula::negate(){

        LET N = This Formula Object
        IF |N.children| = 0 {
                LET NN ← NOTFormula Object
                NN.add(N)
                RETURN NN
        }
        ELSE{
                LET NOF ← ORFormula Object
                FOR child ∈ children{
                        NOF.add(child.negate())
                }
                RETURN NOF
        }
}
```

```
Formula ORFormula::negate(){

        LET N = This Node
        IF |N.children| = 0 {
                LET NN ← NOTFormula();
                NN.add(N)
                RETURN NN
        }
        ELSE{
                LET NAF = ANDFormula Object
                FOR child ∈ children {
                        NAF.add(child.negate())
                }
                RETURN NAF
        }

Formula NOTFormula::negate(){
        LET N = This Node
        RETURN N.child.negate()
}
```

The formula object has been converted into a formula object containing a hierarchy of Formula object, fitting the CNF requirements.

```
Function GetCNFFormula(){
        LET RES ← Empty String
        LET N ← This Node

        IF |N.children| = 0 ➔ RETURN N.name
        ELSE {
                IF N.type = NOT ➔ RES += "-"
                IF |N.children| > 1 & N.type = AND ➔ RES += "("
                FOR child ∈ children {
                        RES += child.GetCNFFormula()
                        IF child ≠ children.last_child {
                                IF child.type = AND ➔ RES += " & "
                                IF child.type = OR ➔ RES += " | "
                        }
                }
                IF |N.children| > 1 & N.type = AND ➔ RES += ")"
        }

        RETURN RES
}
```

Now we have a first-order logic formula, that has been converted into a CNF one for our SAT or SMT solver.

We need to convert the CNF formula into the SMTLIB input format and add the constraints to the model itself. For this part, we created an object SMTConverter that after having been feed transform itself into a sentence corresponding to the SMTLIB input format.

Thanks to the structure of a CNF formula, the formatting into the SMT solver is easily done as each clause is now a disjunction. The addition of the value of each variable present into a disjunction should produce a result of at least 1 to be satisfiable.

First of all, all the leaves are extracted from the tree to inform the object SMTConverter of the variables to consider. When each variable in inserted into the object, the exact number of constraint according to this value is also included in the object with the lines corresponding to the boundaries of each constraint.

For each constraint, a "Total" variable is inserted into the variable, and is linked to the constraints linked to the leaves into the "ADDITION" or "PRODUCT" mode explained earlier.

The problems encountered

Regarding the problems encountered, the biggest problem was the consideration of the attributes that needs to perform a product to be computed.
As an example, we have the percentage of success of a task , represented by a leaf on the attack-defense tree. To be able to compute the total percentage of success of an attack vector, we need to compute the product of these percentages together.

Let pretend that we have a main goal that refined his problem into 3 tasks, and at least 1 or more of these tasks needs to be completed in order to make the main goal successful. We have then P1 that is referring to the task T1, P2 to T2 and P3 to T3.

If the attack vector is considering that T1 and T2 has been done but T3 isn't, the problem still should be successful. But the problem is that we can't only consider T1 and T2 in the formula, we have to specify T3 to the decision problem as the SMT solver will be the one to decide if T1, T2 and T3 are active. We will then have a computation like this one :

$$(T1 * P1) * (T2 * P2) * (T3 * P3)$$

The problem in this kind of computation is that if we have a disjunctive refinement of goal, due to the product between the variables, all the variables need to have a probability higher than 0% and all the variables needs to be considered as active.

But thanks to the logics and theories in SMT-LIB, this problem isn't present anymore as we can use the implication to perform this operation.

```
(assert (>= (+ RootNode1 RootNode2 RootNode3) 1))
(assert (=> (= RootNode1 1) (= RNP1 0.5)))
(assert (=> (= RootNode1 0) (= RNP1 1)))
(assert (=> (= RootNode2 1) (= RNP2 0.6)))
(assert (=> (= RootNode2 0) (= RNP2 1)))
(assert (=> (= RootNode3 1) (= RNP3 0.7)))
(assert (=> (= RootNode3 0) (= RNP3 1)))
(assert (>= (* RNP1 RNP2 RNP3) 0.7))
```

Figure 29 : Assignation for the Product problem

Thanks to these lines, with the symbol "=>" meaning an implication (not to be confused with the higher or equal symbol present on the first line), we transformed our equation previously presented into this one.

$$RNP1 * RNP2 * RNP3$$

Where $RNP1 = Probability\ of\ Task\ T1\ if\ T1 = 1$

OR

$$RNP1 = 1\ if\ T1 = 0$$

(The same logic is applied to T2 and T3)

Another problem related to this solution is a graphical representation problem. As the tree representation is difficult to organize in a graphical way, the problem of potential overlapping between the leaves or the intermediary nodes has to be solved, and has been solved thanks to margin recomputation between the branches based on the position of the most left children and most right children at each level of the tree.

Chapter 7

Conclusion

The actual solution we developed is far to be perfect, and as a first version, needs some improvements to be usable by any interested user. Some improvements have already been imagined and are then presented right now in this section.

What improvements would be interesting

Usage of the Tseitin transformation

To perform a resolution of a problem SAT, the transformation of our well-formed formula into a CNF formula equisatisfiable is primordial to reduce the resolution complexity, and to fit the multiple tools SAT and SMT that we are using.

Currently, we are using a conversion process that are based on eleven rules, that are applied depending on what we meet as a formula part.
Unfortunately, depending on the size of the formula, the size of the CNF formula can quicky grow and then take much more space than expected.

For example, if we meet this formula :

$$(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee \ldots \vee (p_n \wedge q_n)$$

The output of this formula would look like :

$$(p_1 \vee p_2 \vee \ldots \vee p_n) \wedge (q_1 \vee p_2 \vee \ldots \vee p_n) \wedge \ldots \wedge (q_1 \vee q_2 \vee \ldots \vee q_n)$$

The number of clauses produce would quickly explode, and in that case, we would have 2^n clauses to solve.

An alternative to the method we were using is the Tseitin transformation.

The principle behind the Tseitin transformation is the usage of new "auxiliary" variables, that will be assigned to each sub formulas. These auxiliary variables represent the output of a sub formula.

If we take a small example, here a conversion into a CNF formula :

Initial formula well-formed

$$\big((p \vee q) \wedge r\big) \rightarrow (\neg s)$$

We first take all the sub-formulas (without considering the literals themselves)

$$\neg s$$
$$p \vee q$$
$$(p \vee q) \wedge r$$
$$\big((p \vee q) \wedge r\big) \rightarrow (\neg s)$$

We introduce a new variable to each sub formula

$$x_1 \leftrightarrow \neg s$$
$$x_2 \leftrightarrow p \vee q$$
$$x_3 \leftrightarrow x_2 \wedge r$$
$$x_4 \leftrightarrow x_3 \rightarrow x_1$$

Then we have this resulting formula :

$$x_4 \wedge (x_4 \leftrightarrow x_3 \rightarrow x_1) \wedge (x_3 \leftrightarrow x_2 \wedge r) \wedge (x_2 \leftrightarrow p \vee q) \wedge (x_1 \leftrightarrow \neg s)$$

We still have a conversion to CNF to perform, to have a conjunction of disjunction.
We take then each clause separately, and convert them on the base of the same rules we used in our solution

$$(x_4 \leftrightarrow x_3 \rightarrow x_1) \equiv \big(x_4 \rightarrow (x_3 \rightarrow x1)\big) \wedge \big((x_3 \rightarrow x_1) \rightarrow x_4\big)$$
$$(x_4 \leftrightarrow x_3 \rightarrow x_1) \equiv \big(x_4 \rightarrow (\neg x_3 \vee x1)\big) \wedge \big((\neg x_3 \vee x_1) \rightarrow x_4\big)$$
$$(x_4 \leftrightarrow x_3 \rightarrow x_1) \equiv (\neg x_4 \vee \neg x_3 \vee x_1) \wedge (\neg(\neg x_3 \vee x_1) \vee x_4)$$
$$(x_4 \leftrightarrow x_3 \rightarrow x_1) \equiv (\neg x_4 \vee \neg x_3 \vee x_1) \wedge ((x_3 \wedge \neg x_1) \vee x_4)$$
$$(x_4 \leftrightarrow x_3 \rightarrow x_1) \equiv (\neg x_4 \vee \neg x_3 \vee x_1) \wedge (x_3 \vee x_4) \wedge (\neg x_1 \vee x_4)$$

$$(x_3 \leftrightarrow x_2 \wedge r) \equiv (x_3 \rightarrow x_2 \wedge r) \wedge \big((x_2 \wedge r) \rightarrow x_3\big)$$
$$(x_3 \leftrightarrow x_2 \wedge r) \equiv \big(\neg x_3 \vee (x_2 \wedge r)\big) \wedge (\neg(x_2 \wedge r) \vee x_3)$$
$$(x_3 \leftrightarrow x_2 \wedge r) \equiv (\neg x_3 \vee x_2) \wedge (\neg x_3 \vee r) \wedge (\neg x_2 \vee \neg r \vee x_3)$$

$$(x_2 \leftrightarrow p \vee q) \equiv \big(x_2 \rightarrow (p \vee q)\big) \wedge \big((p \vee q) \rightarrow x_2\big)$$
$$(x_2 \leftrightarrow p \vee q) \equiv (\neg x_2 \vee p \vee q) \wedge (\neg(p \vee q) \vee x_2)$$
$$(x_2 \leftrightarrow p \vee q) \equiv (\neg x_2 \vee p \vee q) \wedge \big((\neg p \wedge \neg q) \vee x_2\big)$$
$$(x_2 \leftrightarrow p \vee q) \equiv (\neg x_2 \vee p \vee q) \wedge (\neg p \vee x_2) \wedge (\neg q \vee x_2)$$

$$(x_1 \leftrightarrow \neg s) \equiv (x_1 \rightarrow \neg s) \wedge (\neg s \rightarrow x_1)$$
$$(x_1 \leftrightarrow \neg s) \equiv (\neg x_1 \vee \neg s) \wedge (s \vee x_1)$$

Everything together

$$x_4 \wedge (\neg x_4 \vee \neg x_3 \vee x_1) \wedge (x_3 \vee x_4) \wedge (\neg x_1 \vee x_4) \wedge (\neg x_3 \vee x_2) \wedge (\neg x_3 \vee r) \wedge (\neg x_2 \vee \neg r \vee x_3) \wedge (\neg x_2 \vee p \vee q) \wedge (\neg p \vee x_2) \wedge (\neg q \vee x_2) \wedge (\neg x_1 \vee \neg s) \wedge (s \vee x_1)$$

Thanks to this transformation, the size of the CNF formula produced will be linearly proportional to the size of the original one. We then avoid the exponential explosion of the formula. If we take the first formula mentioned in this subchapter, we will only have $2 * n + 1$ clauses to solve.

Transform the implementation into a Web Application

One of the improvements I consider interesting to implement is the creation of a solution, based on the same technologies used for this one but on the web, with web languages such as node.js or PHP.

Even if we used for this solution C++ that can be compiled on multiple OS, there is still the "negative aspect" to install something on a machine. If everything was available on the web, the people using our solution would only have to keep their project file that will be lighter than a complete executable, and that will save time as they won't have to install something on their computer.

The architecture of the solution would be the following one : A web server is installed and deployed locally or on internet to be widely accessible, and everyone can create its tree thanks to the interface. When the tree is submitted, the tree is exported without transformation into a queue , that will be used by a Tree Solver Service installed on the same machine or another one. The communications between the web server and the tree solver service would be encrypted. The tree solver would transform the tree into a Conjunctive Normal Form and append the numerical conditions based on the attributes, and the SMT solver also present next to the tree solver would check all the possible solutions.

The results are returned to the web server into another format and only reachable by the account owner of the tree. The possibilities as solutions are then visible like our actual solution.

An additional point to do this web implementation is the potential to share with persons (previously accepted into the account for example) the content of the project, and to consult it together for example.

Permitting a user to re-use a sub-tree created

In the actual solution, to create content into the tree, we must create a new node. According to the fact that if our solution is used, it will be used to represent large and complex architecture. Obviously, it is also possible that a branch would be really similar with some slight changes into the nodes, like the context of the intermediary nodes, or the values of the attributes.

A potential improvement for the future version of our solution would be the possibility to create pattern of sub-tree, that would be reusable in our tree directly to avoid multiple creation and attributes tuning of nodes.
The advantage would be a considerable gain of time, especially if the architecture is complex.

Permitting a user to use multiple times the same "basic action"

For the actual implementation, all the leaves are considered as unique and cannot be reused for the rest. It is however possible that an action is present into two different branches of the tree, and then don't need to be considered as different. An improvement would be to assign a unique ID, that could be reused in other leaves of the tree, confirming that if the action is satisfiable on one side, it needs to be satisfiable also on the other side, where conditions could also be added such as existing countermeasure.

Change of the logic used by SMT Solver dynamically

Here as we presented, we are using the logic corresponding to non-linear arithmetic on Reals and Integers with in addition the ArraysEx Theory. But obviously, not all the trees that will be created are considering Reals as a constraint and even more are not considering the ArraysEx Theory.
A suggestion would be to create a logic decision process into the solution, regarding what is expected as constraint types of the tree, and then considering which logic are suiting the most.
An observation about the timing results with this additional time consumption would be needed to see if that improvement would be useful or not.

Switch between SAT and SMT Solving solutions depending on attributes

Regarding the actual development, all the trees created by any user on the software are transformed into a SAT formula , respecting the conditions to be a conjunctive normal form, and then the conditions are applied thanks to the attributes of each leaf of the tree.

A suggestion to extend the usage of this solution would be to reduce the usage of SMT solvers when for example, no numerical constraints (reals or integers) are applied to the

problem. In that case, a SAT solver would be sufficient to compute the complete tree structure.

In this implementation, we mainly used the Z3 solver as SMT solver. By regarding the experimental results of this comparison of multiple SAT solvers where Z3 is also present, we saw that other SAT solvers may sometimes produce results way more efficient than the Microsoft Z3 ones. As Z3 can also resolve SMT problems, it has also more functionalities that may require additional time of computation.

A way to avoid that case would be to analyze the problem itself first, to check if there is any Integer or Reals that need to be considered. If that's not the case, the solution would switch into a "SAT mode" where the problem would be solve as a SAT problem directly. It would need then a SAT solver path and a SMT solver path configuration.

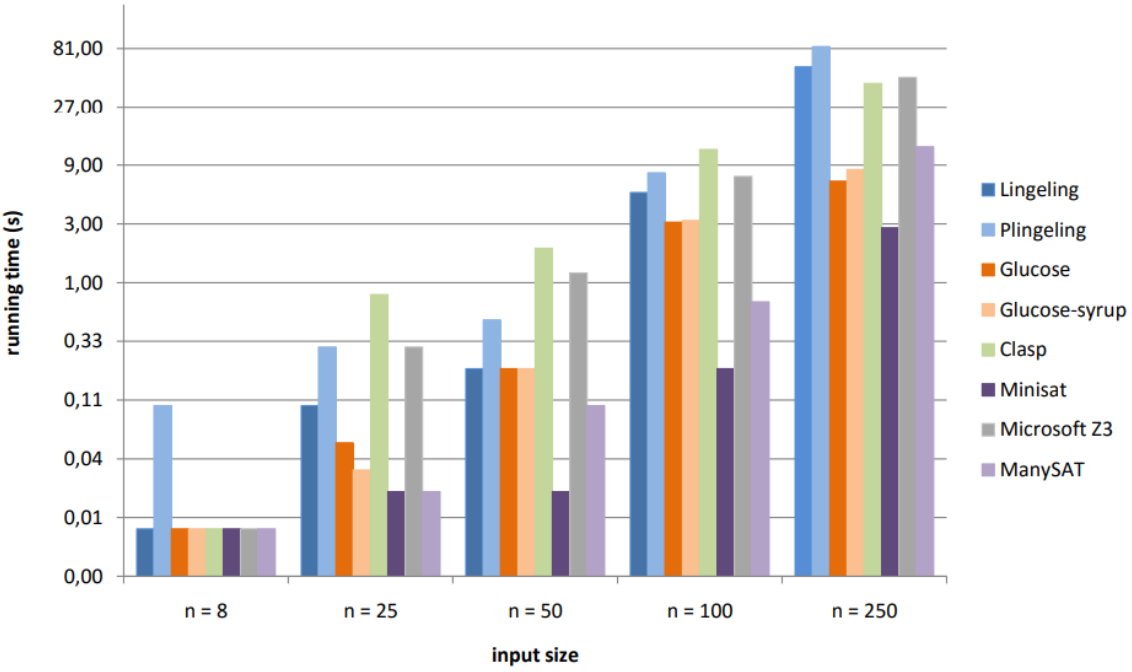


Figure 30 : Comparison of efficiency on N-Queens SAT problem [6]

Automatic Detection of SAT/SMT Solvers present on the system

Currently our solution is working with a setting set up by a human, indicating the executable of the SMT solver. As this solution is dedicated also to non-IT persons, a suggestion would be the auto-discovery of the SAT and SMT solver present on the system thanks to a small database of famous solver known and searching into the full file hierarchy.

Another possibility is also to join to this installation some solvers directly included, to avoid the process to search and install a solver, that needs to fit the requirements of input format, such as SMT-LIB.

Bibliography

[1] B. KORDY, S. MAUW, S. RADOMIROVIC et P. SCHWEITZER, «Attack-defense trees,» *Journal of Logic and Computation,* 2014.

[2] D. Tang, «CS241 -- Lecture Notes: Trees,» Octobre 2013. [En ligne]. Available: https://www.cpp.edu/~ftang/courses/CS241/notes/trees.htm.

[3] «Qt,» [En ligne]. Available: https://www.qt.io.

[4] N. B. Leonardo De Moura, «Z3: an efficient SMT solver,» chez *Tools and Algorithms for the Construction and Analysis of Systems*, 2008, pp. 337-340.

[5] B. Schneier, «Modeling Security Threats,» *Dr. Dobb's Journal,* 1999.

[6] Y. Okamoto, «Sudoku Solver Document,» [En ligne]. Available: http://okmt1230z.com/sudoku.html.

[7] J. Eisner, «How to convert a Formula to CNF,» 14 February 2014. [En ligne]. Available: https://www.cs.jhu.edu/~jason/tutorials/convert-to-CNF.html.

[8] [En ligne]. Available: https://smtlib.cs.uiowa.edu/about.shtml.

[9] P. Garg, «Depth First Search,» [En ligne]. Available: https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/.

[10] T. Sidoruk, «Comparing Efficiency of SAT-solvers and Investigating Characteristics of Specific SAT Instances,» 08 Octobre 2018. [En ligne]. Available: https://ipipan.waw.pl/pliki/seminaria/20181008-Sidoruk.pdf.

[11] A. Singhal, «Tree Data Structure | Tree Terminology,» [En ligne]. Available: https://www.gatevidyalay.com/tree-data-structure-tree-terminology/.

[12] «An introduction to SAT Solving - Applied logic for Computer Science,» 3 Decembre 2017. [En ligne]. Available: https://www.csd.uwo.ca/~mmorenom/cs2209_moreno/slide/lec23-SATsolving.pdf.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**