The ADTool Manual

Piotr Kordy and Patrick Schweitzer

Janvier 2015

Abstract

This is the manual for the ADTool, a graphical software for the construction of ADTrees and their quantitative bottom-up evaluation. The tool is accessible at `http://satoss.uni.lu/projects/atrees/adtool`.

Contents

1 Introduction **5**
- 1.1 General Description and Features . . . 5
- 1.2 Requirements and Distribution . . . 5
- 1.3 Structure of the Manual . . . 6

2 Getting Started — An Introductory Example **6**

3 ADTree Edit Window **9**
- 3.1 Right Mouse Click on Node . . . 9
- 3.2 Mouse Wheel . . . 10
- 3.3 Left Mouse Click on Background . . . 10
- 3.4 Left Mouse Click on Node . . . 10
- 3.5 Click on Node . . . 10
- 3.6 Keyboard Shortcut Example . . . 10

4 ADTerm Edit Window **11**
- 4.1 ADTerm Console . . . 11
- 4.2 *Validate* Button . . . 11
- 4.3 *Revert* Button . . . 11

5 Attribute Window **12**
- 5.1 Attribute Domain Selection . . . 12
- 5.2 Attribute Window Features . . . 13

6 Menubar and Toolbar **14**
- 6.1 File . . . 14
- 6.2 Edit . . . 15
- 6.3 View . . . 16
- 6.4 Windows . . . 16
- 6.5 Domains . . . 16
- 6.6 Help . . . 16

7 XML Files Compatible with ADTool **17**
- 7.1 XML Files without Attribute Values . . . 17
- 7.2 XML Files with Attribute Values . . . 19
- 7.3 Importing and Exporting of XML Files with Attribute Values . . . 21
  - 7.3.1 Exporting to XML Files . . . 21
  - 7.3.2 Importing XML Files . . . 24

8 Changelog **25**

9 Concluding Remarks **26**

Copyright Notice

Acknowledgments

ADTool has been developed at the University of Luxembourg as part of the ATREES project [5] and the EU project TREsPASS [16]. It is a joint collaboration between the SaToSS research group and the Interdisciplinary Centre for Security, Reliability and Trust and has been funded by the National Research Fund, Luxembourg (FNR), under grants No. C08/IS/26 and PHD-09-167, as well as the European Commission's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 318003 (TREsPASS). We would also like to thank all the users of ADTool for their valuable feedback. In particular, we would like to thank Tejeddine Mouelhi, Thea Peacock, Jean Schweitzer, Daniel Marnach, Didier Vojtisek, and the TREsPASS project members, especially Christian Probst, Aleksandr Lenin, and Jan Willemson.

Team

Piotr Kordy: Programming
Barbara Kordy: Functional design and beta testing
Patrick Schweitzer: Manual and functional design
Jean-Paul Weber: Predecessor software

1 Introduction

1.1 General Description and Features

The Attack–Defense Tree Tool (ADTool) [6, 7] allows users to model and display attack–defense scenarios, through the use of attack–defense trees (ADTrees) [9] or an alternative term-based representation of ADTrees called attack–defense terms (ADTerms). It supports the methodology developed within the ATREES project [5]. Since attack trees [15, 12], protection trees [4], and defense trees [3] are formally instances of attack–defense trees, the ADTool can also be employed to automate and facilitate the usage of all aforementioned formalisms. Furthermore, the ADTool allows to perform quantitative analyses on ADTrees/ADTerms. This means that a user is able to answer questions such as: What are the costs of an attack, what is the minimal skill level required for the attacker, how long does it take to implement all necessary defenses or who is the winner of the considered attack–defense scenario, and many others.

In short:

- The ADTool allows the user to model attack–defense scenarios using ADTrees and ADTerms.
- The ADTool allows the user to perform quantitative analyses on ADTrees/ADTerms.

In detail:

- Graphical and user-friendliness features
  - Various tree display methods are available. For example it is possible to collapse and expand trees at all nodes, to move and center the tree in all nodes, and to zoom into and out of the tree. These functions allow a user to model large, real-life scenarios.
  - A user can export ADTrees to image or LaTeXfiles.
  - Input and editing of ADTrees and ADTerms can be done with the keyboard as well as the mouse.
  - Various themes and layouts enhance the user experience.
- Quantitative and computational features
  - The tool automatically converts back and forth between ADTrees and ADTerms. Among others, this provides a tool to learn one of the syntaxes, provided the other one is known.
  - The tool possesses full ADTerm syntax handling including syntax error highlighting.
  - A large number of commonly used attribute domains have been specified and are universally usable.
  - A user can save/load attack–defense scenarios with attribute domains and attribute values.

1.2 Requirements and Distribution

The ADTool is a platform independent software that runs on all common operating systems. The tool requires JDK 6 or later. Additionally the ADTool depends on the following libraries: abego TreeLayout [1], implementing an efficient and customizable tree layout algorithm in Java, and InfoNode Docking Windows [13], a pure Java Swing based docking windows framework.

There are three versions available. First, a jar-file for download with all dependencies included, second, a light version jar-file that only contains the core program, and third, an online version that uses Java Web Start technology. All versions provide the same functionalities and can be found at [11]. To start the program, execute the jar-file or simply click the launch button on the webpage to use the Java Web Start version. More information on Java Web Start can be found at [14]. Starting from the ADTool version 1.4, it is also possible to launch the program from the command line.

- To launch the program type
  `java -jar ADTool-1.4-jar-with-dependencies.jar`
- To launch the program with a given tree (tree.adt) already opened type
  `java -jar ADTool-1.4-jar-with-dependencies.jar tree.adt`
- To launch the program with a given tree (tree.xml) already opened type
  `java -jar ADTool-1.4-jar-with-dependencies.jar tree.xml`
  For more information concerning XML files compatible with ADTool, see Section 7.

1.3 Structure of the Manual

We start out by providing an introductory example to the ADTool in Section 2. This section is recommended for first time users of the ADTool. The remaining parts of the manual explain which functions are available in the ADTool. Section 3 explains the ADTree edit window. In the ADTree edit window an attack–defense scenario can be constructed as a graphical ADTree. Section 4 explains the ADTerm edit window in which the attack–defense scenario can be constructed using term based ADTerms. The tool automatically synchronizes between the ADTree and the ADTerm representations. Section 5 elaborates on the attribute window(s) in which quantitative analysis on a constructed attack–defense model can be performed. Section 6 lists available options in the menubar toolbar.

Note, whenever letters of command options are underlined, the **[ALT]** key in combination with the letter can be used as a shortcut. These shortcuts are not mentioned in this manual. In some cases alternative shortcuts exist. These are explicitly mentioned in **bold font**.

2 Getting Started — An Introductory Example

Before looking at all available options in the ADTool systematically, we start out with a simple, intuitive example that explains the basics of how to use the ADTool. For this, consider the following attack–defense scenario, depicted in Figure 1. The root of the tree represents an attack on a server. Three ways to accomplish this attack are depicted: an insider attack, an outsider attack and stealing the server. To achieve his goal, an insider needs to be internally connected and have the correct user credentials. An attack by an outsider can be prevented if a properly configured firewall is installed.

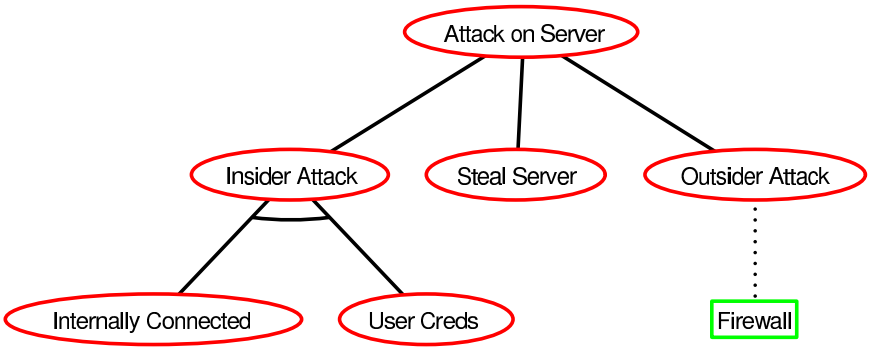

Figure 1: An ADTree for how to attack a server

Let us recreate the tree in the ADTool. For this, we launch the Java Web Start at [11]. After an initialization procedure, Figure 2 depicts the initial interface of the application. The initial interface consists of two parts: the ADTerm edit window in the top and ADTree edit window in the bottom of the application window. In this introductory example, we use ADTrees (the bottom window) to recreate the tree.

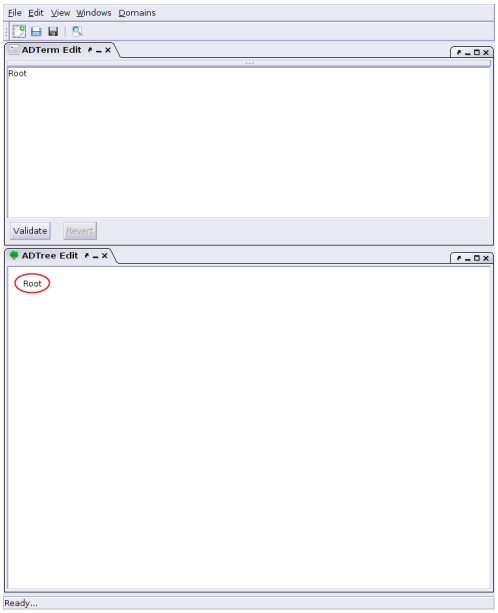

Figure 2: Startup view

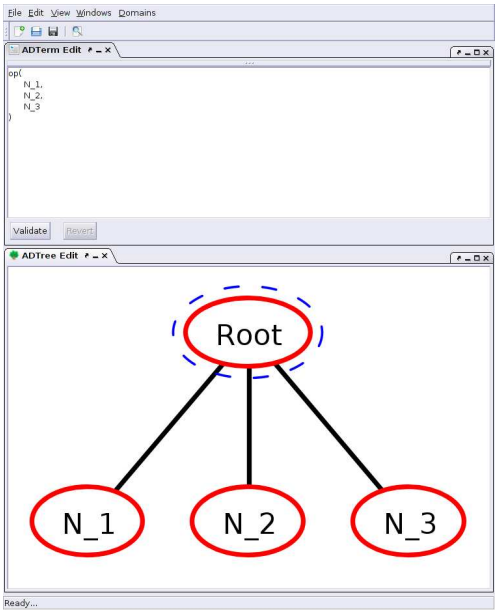

Figure 3: Intermediate step

To start the creation, we right-click into the root node and select *add child*. We repeat this procedure twice more, until the tree looks like Figure 3. Next, we right-click into the node (still) labeled `N_2` and *add child*, which we also repeat. Then, we right-click again into the node `N_2` but select *change operator*, to switch the disjunctive (OR-) refinement into a conjunctive (AND-) refinement. Finally we click into the node labeled `N_4`, and select *add countermeasure*. After this step we have already managed to recreate the structure of the tree from our example. All that remains is to change the node labels. For that we simple double click into each node. A new window opens into which we can insert our desired labels. When all nodes are relabeled the tree is completed and should look like the one in Figure 4. Naturally, the tree does not need to be constructed in the presented order. It is, for example, always possible to change the label of the node immediately after the node has been created.

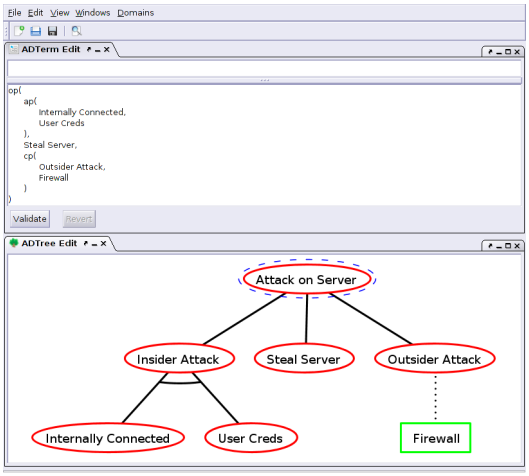

Figure 4: Completed example

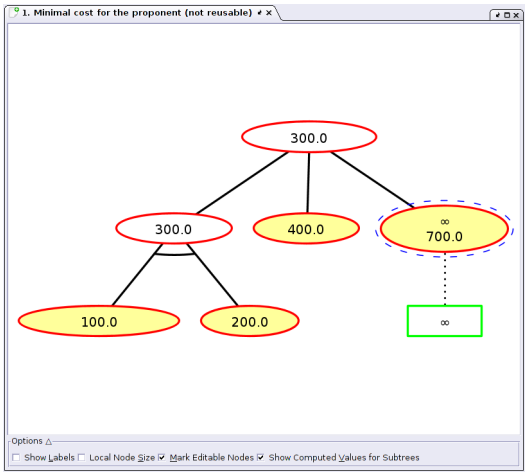

Figure 5: Minimal cost computation

Going to *File, Save*, allows us to save the tree. Using *File, Open* we are then able to load it again, next time we launch the program.

During the construction of the tree a user might have already seen that there are numerous other ways of inserting and afterwards manipulating a desired tree. These methods are detailed in the following sections, which are split up according to the functional components of the ADTool.

To conclude this introductory example, we briefly look at how a quantitative analysis with the help of attributes is performed on an ADTree.

To be able to do this, we need to extend the considered scenario. Let us assume that we want to analyze the attribute "minimal costs". In other words, we ask ourselves: "What are the minimal costs of the proponent[1], assuming that reusing tools is infeasible?" To proceed with the example, we need to determine an appropriate attribute (a selection from 13 predefined attribute domains) and values for all basic actions (user inputs). The question can be answered using the attribute *minimal cost of the proponent (not reusable)*. This attribute is internally modeled by the attribute domain $(\mathbb{R}, \min, +, +, \min, +, \min)$. Furthermore, let us assume that the attacker's costs are the following: 100€ for being internally connected, 200€ for user creds, 400€ in order to steal the server and 799€ to perform an outsider attack. Additionally the defender's costs are 500€ to install a firewall.

To perform a quantitative analysis in the ADTool, we first have to add an attribute domain. We do this by selecting *Domains, Add Attribute Domain* from the menubar. Then the attribute domain selection window opens, see Figure 6.

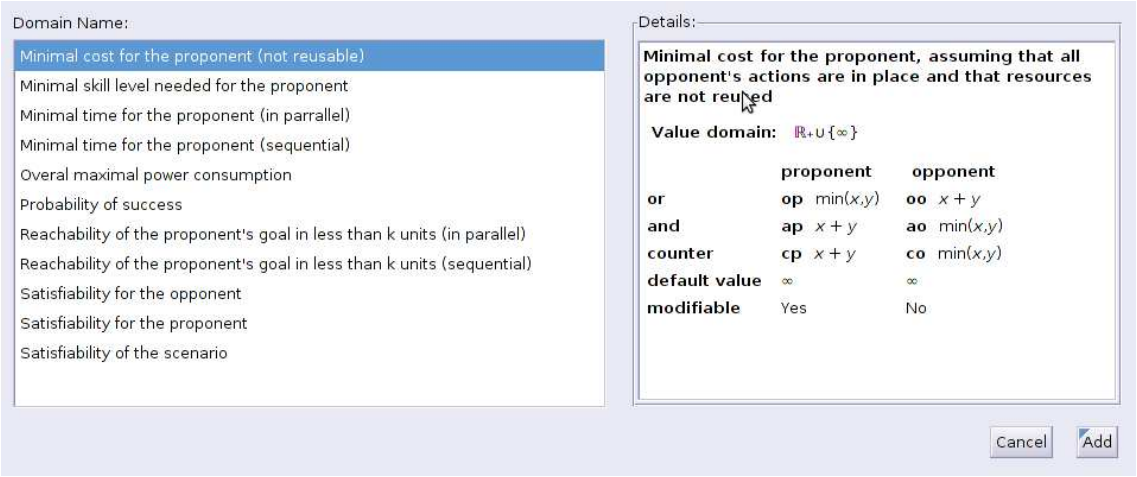

Figure 6: Available attribute domains

In the new window we select the attribute domain *minimal cost of the proponent (not reusable)* and click *Add*. Again a new window opens. This window displays a copy of our tree, where the tree is decorated with default attribute values. The default values are set up in such a way that they correspond to the worst case scenario for a given attribute. (In our case all values are ∞). Nodes that are highlighted in yellow contain values that we can change. To do this, we double click in the corresponding nodes and insert our desired value. We thusly provide four values (100, 200, 400, 799) for all attacker's basic actions. Due to our selection of the attribute domain, we do not have to provide values for the defender (the opponent). For this attribute domain the tool assumes the strongest defender and therefore assigns ∞. Every time a value is provided for a basic action, all dependent values (the ones higher up in the tree) are computed automatically. Naturally, this includes computing a value for the root node. The root node value answers the question. For our example, this means that the attacker's costs are 300€, see Figure 5. As previously, going to *File, Save*, allows us to save the tree, including the newly added attribute domains and attribute values.

This concludes the introductory example. In depth knowledge about the ADTree edit window is given in Section 3, the ADTerm edit window is explained in Section 4 and quantitative analysis is treated in Section 5.

[1]The proponent is the actor (attacker / defender) that corresponds to the type of the root node, the opponent is the other actor

3 ADTree Edit Window

When starting up the software, the ADTree edit window is the lower of the two windows that are opened by default. In this window, an attack–defense scenario can be modeled using the ADTree structure. Operations that change the structure of the tree can be initiated with a mouse right-click or with keyboard shortcuts, see Section 3.1. Other mouse actions change the layout or display of the tree are explained in Sections 3.2-3.5. After the structure of the tree has been changed, the corresponding ADTerm (in the top window, see Section 4) is updated instantaneously. If a user input changes a non-refined node into a refined node, labels are kept in the ADTree display, even though they no longer exist in the corresponding ADTerm.

A mouse right-click also selects the node as active. The fact that a node is active is indicated by a dashed blue line outlining the node. Keyboard shortcuts that change the structure of the tree are only valid for the active node and are explained together with the right-click options in Section 3.1. Using the arrows on the keyboard, it is possible to navigate the tree and change the active node. For an comprehensive example on keyboard input, see Section 3.6.

3.1 Right Mouse Click on Node

Right clicking a node is the most common way to change the tree structure. Alternatively, if a node is active, all indicated keyboard shortcuts can be used. Depending on the structure of the tree some, or all of the following options are available.

Change Label **[CTRL+L]** opens a dialog box in which a new label can be typed.

Change Operator **[CTRL+J]** changes a conjunctive node into a disjunctive node an vice versa.

Fold Above **[SHIFT+Space]** is a display option that promotes the currently active node the root node of the displayed tree. Other nodes are only hidden and not deleted. That fact that a node is folded above is indicated with a triangle tip.

Expand Above **[SHIFT+Space]** is a display option that restores the tree to the display status it was in, before the node was folded above. *Expand above* is only available after a node was folded above and therefore uses the same keyboard shortcut as *fold above*.

Fold Below **[Space]** is a display option that hides all children of the currently active node. The children are only hidden and not deleted. The fact that a node is folded below is indicated with bottom of a triangle.

Expand Below **[Space]** is a display option that restores the tree to the display status it was in, before the node was folded below. *Expand below* is only available after a node was folded below and therefore uses the same keyboard shortcut as *fold below*.

Add Child **[CTRL+N]** adds a child node to the currently active node. The child is of the same type as the active parent node. It is inserted to the right of the already existing children of the same type and gets an automatic label.

Add Countermeasure **[CTRL+I]** adds a child node to the currently active node. The child is of the opposite type as the active node. It is inserted to the right of the already existing children and gets an automatic label. Due to constraints in the attack–defense language, every node is only allowed to have one countermeasure. If several countermeasures are supposed to be modeled, the initial countermeasure should be refined.

Add Left Sibling **[CTRL+S]** adds a sibling to the currently active node. The node is of the same type as the active node. It is inserted to the left of the active node and gets an automatic label. It is not possible to add siblings to countermeasures.

Add Right Sibling **[CTRL+SHIFT+S]** adds a sibling to the currently active node. The node is of the same type as the active node. It is inserted to the right of the active node and gets an automatic label. It is not possible to add siblings to countermeasures.

Remove Subtree **[CTRL+R]** deletes the subtree rooted in the currently active node. This includes deleting the currently active node.

Remove Children **[SHIFT+R]** deletes the subtree rooted in the currently active node without deleting the currently active node.

3.2 Mouse Wheel

With the mouse wheel it possible to zoom into and out of the displayed tree. Alternative keyboard shortcuts are **[CTRL++]** and **[CTRL+−]**. For the zoom to work the ADTree edit window must be active and the cursor must be inside the window.

3.3 Left Mouse Click on Background

When clicking and holding the background in the ADTree edit window, it is possible to slide the scroll bars, if they appear.

3.4 Left Mouse Click on Node

When clicking and holding a node of the tree in the ADTree edit window, it is possible to move the tree around in the window.

3.5 Click on Node

A click on a non-active node, selects the node as active. A click on an active node, opens up a dialog box to change the label of the node. Changing the label of the active node can also be done with the keyboard shortcut [CTRL+L].

In Version 1.2, a possibility of creating multi-line labels has been added. In order to ensure that trees created with ADTool have an appealing layout, the user is *strongly advised to use similar size of labels for all the nodes*. In general, node labels should be as short as possible, preferably of the form verb+noun, e.g., 'steal money', 'access file', etc.

3.6 Keyboard Shortcut Example

To conclude the description of the ADTree edit window, we describe how to create the example tree described in Section 2 almost entirely without the use of the mouse:

After initially opening the program, the root node in the ADTree edit window needs to be selected with a left-click. Once it has a dashed blue outline, the following sequence of keys (re-)creates the initial example.

- **[CTRL+L]**, Type: Attack on Server, **[Enter]**
- **[CTRL+N]**, **[↓]**, **[CTRL+L]**, Type: Steal Server, **[Enter]**
- **[CTRL+S]**, **[←]**, **[CTRL+L]**, Type: Insider Attack, **[Enter]**
- **[CTRL+N]**, **[↓]**, **[CTRL+L]**, Type: Internally Connected, **[Enter]**
- **[CTRL+SHIFT+S]**, **[→]**, **[CTRL+L]**, Type: User Creds, **[Enter]**
- **[↑]**, **[→]**, **[CTRL+SHIFT+S]**, **[→]**, **[CTRL+L]**, Type: Outsider Attack, **[Enter]**
- **[CTRL+I]**, **[↓]**, **[CTRL+L]**, Type: Firewall, **[Enter]**

4 ADTerm Edit Window

In the ADTerm edit window, the model of an attack–defense scenario can be constructed with help of an ADTerm by typing it in the input window. The syntax consists of six operators:

- op(): The OR operator of the proponent ($\vee^{\mathrm{P}}$).
- ap(): The AND operator of the proponent ($\wedge^{\mathrm{P}}$).
- oo(): The OR operator of the opponent ($\vee^{\mathrm{O}}$).
- ao(): The AND operator of the opponent ($\wedge^{\mathrm{O}}$).
- cp(): The countermeasure operator of the proponent (c^{P}).
- co(): The countermeasure operator of the opponent (c^{O}).

The operators are case insensitive and take labels (no comas and brackets) or operators as arguments. More precisely, the AND and OR operators are n-ary operators and take labels, operators of the same type and at most one countermeasure operator as arguments. The countermeasure operators are binary and take labels or an operator of the same type followed by an operator of the other type or a combination of labels and correctly typed operators as arguments. A short example is given in Section 4.1. For a detailed use of the attack–defense term language, we refer to [8].

4.1 ADTerm Console

In this window, an ADTerm can be typed. For example typing

```
op(
      ap(
            Internally Connected,
            User Creds
      ),
      Steal Server,
      cp(
            Outsider Attack,
            Firewall
      )
)
```

partially displays the tree of the introductory example, given in Section 2 after the input is validated with the *validate button*, see Section 4.2. As labels of refined nodes do not appear in the ADTerm language, they can also not inserted via terms. They need to be added in the ADTree edit window, see Section 3. ADTerms are automatically formated into an intuitive layout.

4.2 *Validate* Button [CTRL+Enter]

The *validate button* checks the input and displays the inserted ADTerm as ADTree. In case of syntax errors, the program highlights the most probable place and provides a short error message.

4.3 *Revert* Button

The *revert button* is normally inactive. It becomes active when a user creates a syntactically invalid ADTerm and tries to validate it. Pressing the revert button restores last valid ADTerm corresponding to the current ADTree. Changes made to the ADTerm are lost.

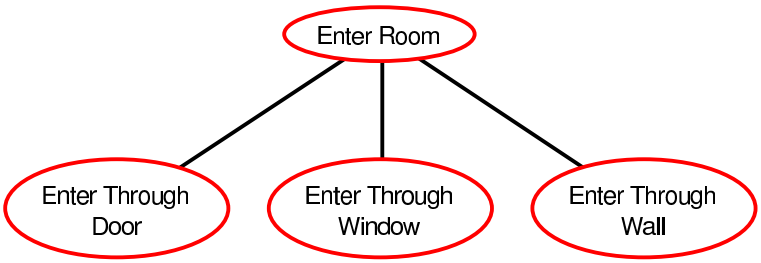

Figure 7: How to enter a room

5 Attribute Window

Quantitative analyses on ADTree/ADTerms are performed with the help of attributes. Since different attributes require different computations, the attribute needs to be specified. If several questions are supposed to be answered on ADTree, then several attributes need to be specified. Specification is done with the help of attribute domains. We briefly exemplify why different quantitative questions require different attribute domains and how to select an appropriate domain in Section 5.1. For a comprehensive treatment of quantitative questions and how they correspond to attribute domains, we refer to [10].

Every time a new attribute domain is selected a new attribute window opens up. This window displays a copy of the ADTree along with default values that depend on the chosen attribute domain. Features of the attribute window and how values can be changed is explained in Section 5.2.

5.1 Attribute Domain Selection

We shortly elaborate why the choice of the attribute domain is crucial. Suppose we want to enter a room. We have modeled three ways to get into the room, either through a door, through a window or through the wall, as depicted in Figure 7.

Suppose we are interested in whether or not it is possible to enter the room and how much it would cost. As basic assignments we use 1 for "enter through door" and "enter through window" and 0 for "enter through wall", indicating that it is possible to enter through the door or the window, but not through the wall. We also assign costs values to the basic actions and use 1€ as cost of entering through the door, 100€ as cost for entering through the window and ∞€ as cost for entering through the wall. To automatically compute the value at the root node, we assign a functional operator to the disjunctive node. We use max to compute whether or not it is possible to enter the room and and min to compute the costs of entering the room. As result we get $\max\{1, 1, 0\} = 1$. This means, that, yes, it is possible to enter the room. The second computation yields: $\min\{1, 100, \infty\} = 1$. This means that it would cost 1€ to perform a successful attack. Together the two questions show that depending on the question, the functional operators need to be different.

This example obviously only illustrates the functional operator for a disjunctive node. Since, functional operators for all cases (disjunctive, conjunctive and countermeasures) are gathered in the attribute domains, selecting a different attribute domain changes the computation. As mentioned in the introductory example in Section 2, correctly specified quantitative questions help us choosing correct attribute domain. A correctly specified question contains information about the "notion", the "modality", the "owner" and the "execution."

In the ADTool there are 13 predefined attribute domains that cover the most common questions. In the following we list the predefined attribute domains:

Difficulty for the proponent (L,M,H) The minimal difficulty level for the proponent, on the scale Low-Medium-High, assuming that all opponent's actions are in place and that the set of difficulty levels $\{L, M, H\}$ is linearly ordered $L < M < H < E$.

Difficulty for the proponent (L,M,H,E) The minimal difficulty level for the proponent, on the scale Low-Medium-High-Extreme, assuming that all opponent's actions are in place and that the set of difficulty levels $\{L, M, H, E\}$ is linearly ordered $L < M < H < E$.

Minimal cost for the proponent (not reusable) Minimal cost for the proponent, assuming that all opponent's actions are in place and that resources are not reused.

Minimal skill level needed for the proponent Minimal skill level needed for the proponent, assuming that all opponent's actions are in place and that the set of skill levels is linearly ordered.

Minimal time for the proponent (in parallel) Minimal time for the proponent, assuming that all opponent's actions are in place and that actions are executed in parallel.

Minimal time for the proponent (sequential) Minimal time for the proponent, assuming that all opponent's actions are in place and that actions are executed on after another.

Overall maximal power consumption The overall maximal power consumption of the scenario, knowing that sharing of power is impossible.

Probability of success Probability of success, assuming that all actions are mutually independent.

Reachability of the proponent's goal in less than k units (in parallel) Reachability of the proponent's goal in less than k units assuming that all opponent's actions are in place and that actions can be executed in parallel.

Reachability of the proponent's goal in less than k units (sequential) Reachability of the proponent's goal in less than k units assuming that all opponent's actions are in place and that actions cannot be executed in parallel.

Satisfiability for the opponent Satisfiability for the opponent, assuming that all proponent's actions are in place.

Satisfiability for the proponent Satisfiability for the proponent, assuming that all opponent's actions are in place.

Satisfiability of the scenario Satisfiability of the scenario.

It is also possible to, e.g., compute the minimal number of specialists using the "Minimal cost for the proponent (not reusable)" domain. For other uses of the given attribute domains, we refer to [10].

5.2 Attribute Window Features

Every time a new attribute domain is chosen and accepted by clicking *add*, the attribute window opens. The attribute domain window depicts a copy of the original ADTree and displays attribute values. Initially default values are displayed. The default values depend on the selected attribute domain. All non-refined nodes are highlighted in yellow, indicating that these values can be changed. For non-refined nodes that are counteracted, two values are displayed: the bottom value depicts the basic assignment and the top value the computed value. To change a value, a node is first activated (dashed blue line) with a single left click. When activated node is clicked again, a dialog box that allows to change the value opens up. New values are automatically computed for relevant parts of the tree, after attribute values have been changed. For refined nodes, i.e., nodes that are not highlighted, the computed values are shown.

Note that, when non-refined nodes have the same label, they are automatically assigned the same values.

There are a few check boxes underneath the attribute window that allow the user to decide which quantitative aspects are displayed on the tree. Additionally the valuations view window and the domain details window found in the menubar under *view* also display various attribute (domain) properties. We elaborate on these additional features in the following.

Show Labels Checking this box adds all node labels of the original ADTree (from the ADTree edit window) also on the tree in the attribute window.

Local Node Size Checking this box resizes the tree displayed in the attribute windows by adjusting each node to its content. This usually shrinks the tree, but the optical alikeness to the tree in the ADTree edit window is lost.

Mark Editable Nodes Unchecking this box removes the yellow highlighting from editable nodes.

Show Computed Values for Subtrees Unchecking this box removes the computed value from non-refined nodes that are countered.

Remove Domain By clicking the Remove Domain button, the user closes the domain as well as permanently removes the corresponding inserted and computed values. Remove Domain button was added in Version 1.2.

Valuations View Window If the valuations view window is selected, it displays information belonging to the currently active, i.e., the last selected, attribute window. In the valuations view window a tabular representation of all non-refined nodes is given. In this view, it is also possible to change the attribute value.

Domain Details View Window If the domain details view window is selected, it displays information belonging to the currently active, i.e., the last selected, attribute window. The window displays the basic information about the attribute domain which is also displayed during the process when adding an attribute domain.

6 Menubar and Toolbar

This section contains a brief description of the features available from the menubar and the toolbar.

6.1 File

File, New **[ALT+N]** clears the existing ADTree and ADTerm. (Also selectable as the first icon from the toolbar.)

File, Open **[ALT+O]** displays a file system browser dialog to select and load a file. (Also selectable as the second icon from the toolbar.)

File, Save **[ALT+S]** displays a file system browser to insert a file name and save the current tree under this file name. The default file name is the same as the root node of the tree. (Also selectable as the third icon from the toolbar.)

File, Load Example Tree **[ALT+L]** displays a file system browser in which the user can select predefined attack–defense trees.

File, Export to, ... Offers the option to export the currently displayed tree to .pdf, .png, .jpg, or .tex. When several trees windows are open, e.g., two trees displaying different attributes and the original tree describing the structure, the program will export the last activated tree. To activate a tree (e.g., after loading) a click on a node of the tree is necessary. Blue dashed line around a node indicates that the tree is indeed activated. The .tex file makes use of the PSTricks package, so only a direct latex and no pdflatex compilation is possible.

In Version 1.2, the possibility of exporting ADTrees created with the ADTool to XML files has been added. XML files resulting from exporting .adt files are conform with a specific XSD schema, explained in detail in Section 7. Information on possible options to export ADTrees to XML files containing attribute values is given in Section 7.3.

File, Import from ... In Version 1.2, the possibility of importing into ADTool ADTrees written in xml format has been added. In order to be readable by ADTool, the XML files must comply with a specific XSD schema, explained in detail in Section 7.

File, Print **[ALT+P]** prints the current ADTree.

File, Print Preview **[CTRL+P]** displays a print preview of the current ADTree. It is possible to customize the number of pages on which the tree is printed to allow for readable printouts of large scale trees. (Also selectable as the fourth icon from the toolbar.) Printing on 30 pages is displayed in Figure 8.

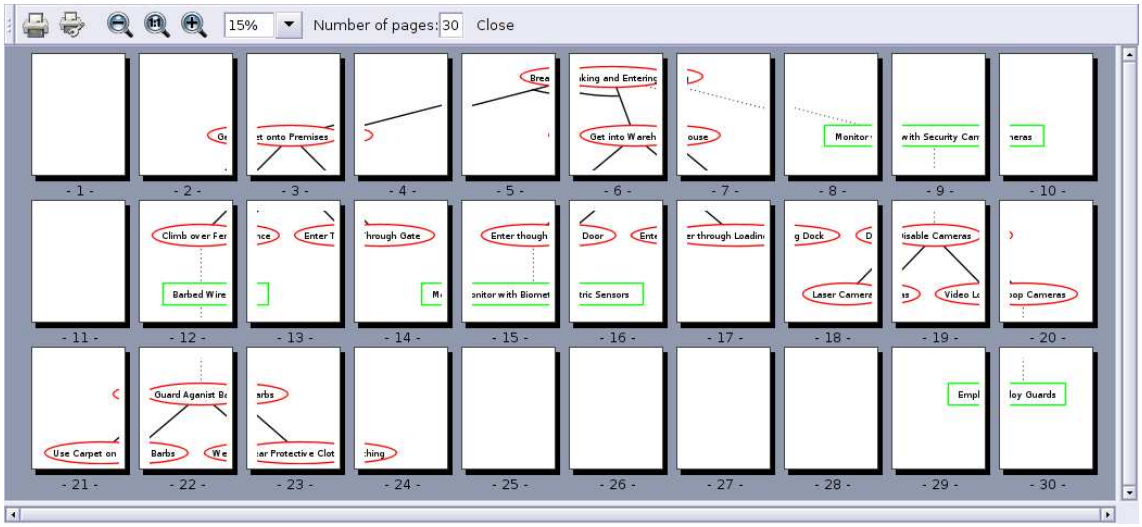

Figure 8: Printing an ADTree large-scale.

File, Exit **[CTRL+Q]** Closes the application. If there are unsaved changes, the user is asked if the changes should be saved.

6.2 Edit

Edit, Cut **[CTRL+X]** allows cutting parts of the ADTerm in the ADTerm edit window.

Edit, Copy **[CTRL+C]** allows copying parts of the ADTerm in the ADTerm edit window.

Edit, Paste **[CTRL+V]** allows pasting parts an ADTerm in the ADTerm edit window.

Edit, Paste **[CTRL+V]** allows pasting parts an ADTerm in the ADTerm edit window.

Edit, Switch Attacker/Defender roles **[CTRL+W]** changes the role of the root and all other nodes accordingly. Attack nodes, displayed as red ovals, become defense nodes and defense nodes, displayed as green rectangles, become attack nodes. This was it is possible to display an ADTree with a defense node as root.

Edit, Validate Terms **[CTRL+Enter]** checks whether a term entered in the ADTerm window is syntactically correct. If the term is not correct, the program highlights a problematic part and displays a brief error message.

6.3 View

This category offers various display tweaks.

View, Themes, ... offers various themes that can be selected to change the color and the border of tall windows in the ADTool.

View, Layout, ... offers options to store and restore the current positions of the windows.

View, Expand all nodes nullifies all tree display changes that were caused by *fold above* and *fold below* to display all nodes of the tree.

View, Fit to window selects a zoom level such that the entire ADTree is displayed in the ADTree edit window. The option fits either the tree in the ADTree edit window or the attribute window, depending on which one was last selected.

6.4 Windows

This category allows a user to restore closed windows.

Windows, ADTree Edit restores the ADTree edit windows, when it was closed.

Windows, ADTree Edit restores the ADTerm edit windows, when it was closed.

Windows, Valuations View restores the valuations view window that displays a tabular list of the attribute values. The list belongs to the currently selected attribute window.

Windows, Domain Details View restores the domain details window that displays a summary of the selected attribute domain. The details belong to the currently selected attribute window.

6.5 Domains

Domains, Add Attribute Domain opens the attribute domain selection dialog, see Figure 6 in which 13 predefined attribute domains can be selected. If any useful attribute domains should be missing, you can always contact us at [5].

6.6 Help

Help, About **[ALT+H A]** indicates the current version of the tool and its release date. It also gives information about funding sources.

7 XML Files Compatible with ADTool

Starting from Version 1.2, ADTool supports exporting and importing ADTrees written in the XML form. Representing ADTrees as XML files has been made possible by specifying an XSD schema describing the ADTree language, called adtree.xsd. The ADTool considers XML files conforming with this schema to be valid, and only such files can be imported by the application. Likewise, every XML file resulting from exporting an .adt file into .xml conforms with the aforementioned schema.

7.1 XML Files without Attribute Values

To ease the presentation, we first provide the adtree.xsd schema specifying how to construct an XML file which does not contain any attribute values. The schema is given in Figure 9.

```
1  <?xml version="1.0"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <xsd:element name="adtree" type="adtreeType">
4    </xsd:element>
5    <xsd:complexType name="adtreeType">
6      <xsd:sequence>
7        <xsd:element name="node" type="nodeType" minOccurs="1" maxOccurs="1"/>
8      </xsd:sequence>
9    </xsd:complexType>
10   <xsd:complexType name="nodeType">
11     <xsd:sequence>
12       <xsd:element name="label" type="labelType" minOccurs="1" maxOccurs="1"/>
13       <xsd:element name="node" type="nodeType" minOccurs="0" maxOccurs="unbounded"/>
14     </xsd:sequence>
15     <xsd:attribute name="refinement" type="refinementType" use="optional" default="disjunctive"/>
16     <xsd:attribute name="switchRole" type="booleanType" use="optional" default="no"/>
17   </xsd:complexType>
18   <xsd:simpleType name="labelType">
19     <xsd:restriction base="xsd:string">
20       <xsd:pattern value="[0-9A-Za-z\s\?!\-_’]+"/>
21       <xsd:whiteSpace value="preserve"/>
22     </xsd:restriction>
23   </xsd:simpleType>
24   <xsd:simpleType name="refinementType">
25     <xsd:restriction base="xsd:string">
26       <xsd:enumeration value="disjunctive"/>
27       <xsd:enumeration value="conjunctive"/>
28     </xsd:restriction>
29   </xsd:simpleType>
30   <xsd:simpleType name="booleanType">
31     <xsd:restriction base="xsd:string">
32       <xsd:enumeration value="yes"/>
33       <xsd:enumeration value="no"/>
34     </xsd:restriction>
35   </xsd:simpleType>
36 </xsd:schema>
```

Figure 9: XSD schema adtree.xsd specifying XML files compatible with ADTool

Below, we briefly explain elements, attributes, and types declared by the adtree.xsd schema.

- The XML file conforming with adtree.xsd contains exactly one "adtree" element representing the entire tree (see lines $3-9$ in Figure 9).
- Each node of an ADTree is represented by a "node" element (lines $10-17$), having
  - exactly one sub-element "label", corresponding to the label of the node,
  - a finite number of "node" sub-elements, corresponding to the node's children, including countering nodes.
- There are two attributes that characterize "node" elements: "refinement" attribute and "switchRole" attribute.
  - "refinement" attribute represents the refinement of the node in the ADTree. This attribute has two possible values: "disjunctive" and "conjunctive" (lines $24-29$). The default value of the "refinement" attribute is "disjunctive", thus only conjunctive nodes require an explicit specification of the value for the "refinement" attribute.
  - "switchRole" attribute is used to switch between an attacker and a defender role. This is a Boolean attribute with two possible values "yes" and "no" (lines $30-35$). When "switchRole" is set to "yes", the node corresponding to the current element belongs to the opposite player than its parent node.
- The values of element "label" (lines $18-23$) are strings that can be composed of the following characters: numbers from 0 to 9, capital and small letters of the English alphabet, space, horizontal tab, enter, ?, !, -, _, and ' (note that, starting from Version 1.3, labels of ADTree nodes can contain more special characters, as specified by line 22 of the extended schema given in Figure 12.)

Figure 10 represents an ADTree for stealing money using an ATM machine. Exporting this tree into the XML format results in the XML file illustrated in Figure 11.

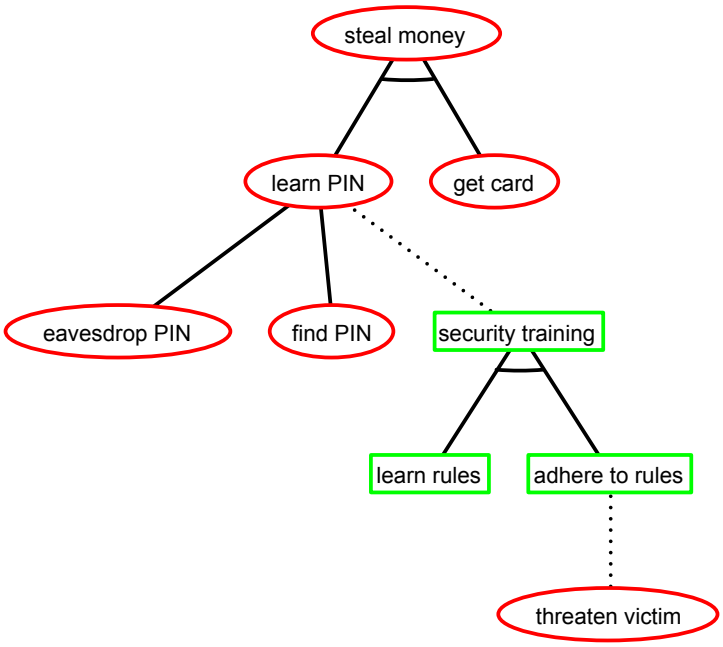

Figure 10: ADTree for stealing money using an ATM.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <adtree>
3    <node refinement="conjunctive">
4      <label>steal money</label>
5      <node refinement="disjunctive">
6        <label>learn PIN</label>
7        <node refinement="disjunctive">
8          <label>eavesdrop PIN</label>
9        </node>
10       <node refinement="disjunctive">
11         <label>find PIN</label>
12       </node>
13       <node refinement="conjunctive" switchRole="yes">
14         <label>security training</label>
15         <node refinement="disjunctive">
16           <label>learn rules</label>
17         </node>
18         <node refinement="disjunctive">
19           <label>adhere to rules</label>
20           <node refinement="disjunctive" switchRole="yes">
21             <label>threaten victim</label>
22           </node>
23         </node>
24       </node>
25     </node>
26     <node refinement="disjunctive">
27       <label>get card</label>
28     </node>
29   </node>
30 </adtree>
```

Figure 11: ADTree from Figure 10 in the XML form.

7.2 XML Files with Attribute Values

Starting from Version 1.3, the ADTool users can also import/export XML files containing information about attribute values stored at the nodes of ADTrees. The extended schema specifying how such XML file should be constructed is given in Figure 12. Below we explain the elements, attributes, and types declared by this schema.

- The adtree element contains exactly one root node (line 5) and may contain a finite number of "domain" elements (line 6)
- Each node of an ADTree is represented by a "node" element (lines $10-18$), having
  - exactly one sub-element "label", corresponding to the label of the node,
  - a finite number of "parameter" sub-elements, corresponding to quantitative or qualitative measures and their values, and
  - a finite number of "node" sub-elements, corresponding to the node's children, including countering nodes.
- There are two attributes that characterize "node" elements: the "refinement" attribute and the "switchRole" attribute.
  - The "refinement" attribute represents the refinement of the node in the ADTree. This attribute has two possible values: "disjunctive" and "conjunctive" (lines $36-41$). The

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="adtreeType">
    <xsd:sequence>
      <xsd:element name="node" type="nodeType" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="domain" type="domainType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="nodeType">
    <xsd:sequence>
      <xsd:element name="label" type="labelType" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="parameter" type="parameterType" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="node" type="nodeType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="refinement" type="refinementType" use="optional" default="disjunctive"/>
    <xsd:attribute name="switchRole" type="booleanType" use="optional" default="no"/>
  </xsd:complexType>

  <xsd:simpleType name="labelType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0-9A-Za-z\s\?!\-_':;#\[\{\}\]=\*/\\\|@\^‘&amp;quot;&amp;amp;$~\.&amp;lt;&amp;gt;\+%]+"/>
      <xsd:whiteSpace value="preserve"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="parameterType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="domainId" type="xsd:string" use="required"/>
        <xsd:attribute name="category" type="categoryType" use="optional" default="basic"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

  <xsd:simpleType name="refinementType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="disjunctive"/>
      <xsd:enumeration value="conjunctive"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="booleanType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="yes"/>
      <xsd:enumeration value="no"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="categoryType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="basic"/>
      <xsd:enumeration value="default"/>
      <xsd:enumeration value="derived"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="domainType">
    <xsd:sequence>
      <xsd:element name="class" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="range" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="tool" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
  </xsd:complexType>

  <xsd:element name="adtree" type="adtreeType">
    <xsd:key name="DomainKey">
      <xsd:selector xpath=".//domain"/>
      <xsd:field xpath="@id"/>
    </xsd:key>
    <xsd:keyref name="DomainKeyRef" refer="DomainKey">
      <xsd:selector xpath=".//parameter"/>
      <xsd:field xpath="@domainId"/>
    </xsd:keyref>
  </xsd:element>
</xsd:schema>
```

Figure 12: XSD schema adtree.xsd specifying XML files containing attribute values, compatible with ADTool

default value of the "refinement" attribute is "disjunctive", thus only conjunctive nodes require an explicit specification of the value for the "refinement" attribute.

- The "switchRole" attribute is used to switch between an attacker and a defender role. This is a Boolean attribute with two possible values "yes" and "no" (lines $43-48$). When "switchRole" is set to " yes", the node corresponding to the current element belongs to the opposite player than its parent node.

- The values of element "label" (lines $20-25$) are strings that can be composed of the following characters: numbers from 0 to 9, capital and small letters of the English alphabet, space, horizontal tab, enter, ?, !, -, _, ', semicolon, colon, #, [, {, },], =, *, /, \, |, @, ^, ‘, ", &, $, ~, ., < ,>, +, %.

- The "parameter" element contains the value of the quantitative or qualitative measure that it represents (lines $27-34$).

- There are two attributes that characterize the "parameter" elements: "domainId" attribute (line 30) and "category" attribute (line 31)
  - The (required) "domainId" attribute references to the corresponding " domain" element
  - The (optional) "category" attribute (lines $50-56$) specifies
    - which values can be edited (category="basic"), these are values of the non-refined nodes that can be modified by the user,
    - which are non-editable default values (category="default"), these are basic values assigned to non-refined nodes that cannot be changed by the user, and
    - which values are results of a computation (category="derived"), these are values computed at the refined nodes.

- Each "domain" element (lines $58-65$) contains exactly one "class" sub-element, zero or one "range" sub-element, and a finite number of "tool" sub-elements.
  - The "class" sub-element represents the name of the Java class that implements the corresponding domain (line 60).
  - In the case when the domain is restricted (e.g., Reachability of the proponent's goal in less than k units), the sub-element "range" (line 61) stores the value of the used restriction parameter (i.e., k).
  - The "tool" elements (line 62) list the tools in which the current domain is used. The ADTool ignores the content of all "domain" elements which do not contain the "tool" sub-element having value 'ADTool'.

- The attribute called "id" uniquely identifies each "domain" element (line 64).

- Lines $67-71$ ensure that each "domain" element has a unique identifier "id".

- Lines $72-76$ guarantee that the "domainId" attribute of the "parameter" elements references to one of the declared domains.

7.3 Importing and Exporting of XML Files with Attribute Values

7.3.1 Exporting to XML Files

When exporting an ADTree containing attribute values to an XML file, the user specifies whether he wants to export only the tree structure or the values as well.

- In order to export information concerning attribute values, the user needs to check the box "Include domains" in the Export tree window

- The user has a choice of exporting only the values corresponding to the basic assignment or to also include the computed values in the exported XML file. In order to export computed values, it is necessary to check the "Add derived values" box in the Export tree window.

Figure 13 depicts the XML file resulting form exporting the tree given in Figure 10 including two parameters (minimal time and probability).

```
<?xml version="1.0" encoding="UTF-8"?>
<adtree>
  <node refinement="conjunctive">
    <label>steal money</label>
    <node refinement="disjunctive">
      <label>learn PIN</label>
      <node refinement="disjunctive">
        <label>eavesdrop PIN</label>
        <parameter domainId="MinTimeSeq1">10.0</parameter>
        <parameter domainId="ProbSucc2">0.5</parameter>
      </node>
      <node refinement="disjunctive">
        <label>find PIN</label>
        <parameter domainId="MinTimeSeq1">100.0</parameter>
        <parameter domainId="ProbSucc2">0.1</parameter>
      </node>
      <node refinement="conjunctive" switchRole="yes">
        <label>security training</label>
        <node refinement="disjunctive">
          <label>learn rules</label>
          <parameter domainId="ProbSucc2">0.7</parameter>
        </node>
        <node refinement="disjunctive">
          <label>adhere to rules</label>
          <parameter domainId="ProbSucc2">0.3</parameter>
          <node refinement="disjunctive" switchRole="yes">
            <label>threaten victim</label>
            <parameter domainId="MinTimeSeq1">5.0</parameter>
            <parameter domainId="ProbSucc2">0.9</parameter>
          </node>
        </node>
      </node>
    </node>
    <node refinement="disjunctive">
      <label>get card</label>
      <parameter domainId="MinTimeSeq1">20.0</parameter>
      <parameter domainId="ProbSucc2">0.25</parameter>
    </node>
  </node>
  <domain id="MinTimeSeq1">
    <class>lu.uni.adtool.domains.predefined.MinTimeSeq</class>
    <tool>ADTool</tool>
  </domain>
  <domain id="ProbSucc2">
    <class>lu.uni.adtool.domains.predefined.ProbSucc</class>
    <tool>ADTool</tool>
  </domain>
</adtree>
```

Figure 13: ADTree from Figure 10 with minimal time and probability parameters

Figure 14 illustrates the XML file resulting form exporting the tree given in Figure 10 on which two parameters (minimal time and probability) were evaluated. It contains all: basic, default and computed values.

```
<?xml version="1.0" encoding="UTF-8"?>
<adtree>
  <node refinement="conjunctive">
    <label>steal money</label>
    <parameter category="derived" domainId="MinTimeSeq1">35.0</parameter>
    <parameter category="derived" domainId="ProbSucc2">0.13461249999999997</parameter>
    <node refinement="disjunctive">
      <label>learn PIN</label>
      <parameter category="derived" domainId="MinTimeSeq1">15.0</parameter>
      <parameter category="derived" domainId="ProbSucc2">0.5384499999999999</parameter>
      <node refinement="disjunctive">
        <label>eavesdrop PIN</label>
        <parameter domainId="MinTimeSeq1">10.0</parameter>
        <parameter domainId="ProbSucc2">0.5</parameter>
      </node>
      <node refinement="disjunctive">
        <label>find PIN</label>
        <parameter domainId="MinTimeSeq1">100.0</parameter>
        <parameter domainId="ProbSucc2">0.1</parameter>
      </node>
      <node refinement="conjunctive" switchRole="yes">
        <label>security training</label>
        <parameter category="derived" domainId="MinTimeSeq1">5.0</parameter>
        <parameter category="derived" domainId="ProbSucc2">0.020999999999999994</parameter>
        <node refinement="disjunctive">
          <label>learn rules</label>
          <parameter category="derived" domainId="MinTimeSeq1">Infinity</parameter>
          <parameter domainId="ProbSucc2">0.7</parameter>
        </node>
        <node refinement="disjunctive">
          <label>adhere to rules</label>
          <parameter category="default" domainId="MinTimeSeq1">Infinity</parameter>
          <parameter category="derived" domainId="MinTimeSeq1">5.0</parameter>
          <parameter domainId="ProbSucc2">0.3</parameter>
          <parameter category="derived" domainId="ProbSucc2">0.029999999999999992</parameter>
          <node refinement="disjunctive" switchRole="yes">
            <label>threaten victim</label>
            <parameter domainId="MinTimeSeq1">5.0</parameter>
            <parameter domainId="ProbSucc2">0.9</parameter>
          </node>
        </node>
      </node>
    </node>
    <node refinement="disjunctive">
      <label>get card</label>
      <parameter domainId="MinTimeSeq1">20.0</parameter>
      <parameter domainId="ProbSucc2">0.25</parameter>
    </node>
  </node>
  <domain id="MinTimeSeq1">
    <class>lu.uni.adtool.domains.predefined.MinTimeSeq</class>
    <tool>ADTool</tool>
  </domain>
  <domain id="ProbSucc2">
    <class>lu.uni.adtool.domains.predefined.ProbSucc</class>
    <tool>ADTool</tool>
  </domain>
</adtree>
```

Figure 14: ADTree from Figure 10 with inserted and computed values for the minimal time and probability parameters

7.3.2 Importing XML Files

- The values of the parameters which in the XML file have category="derived" are ignored when the file is imported by ADTool.
- When the imported XML file lacks a parameter element for some declared domain and some non-refined node, the ADTool inserts the default value defined for this domain at that node.
- If the XML file contains a node with the parameter characterized by category="default", but the value specified in the XML file is not the default value implemented in the ADTool for the corresponding domain, the ADTool replaces the value from the XML file with its own default value when importing the file.

8 Changelog

Version 1.0 (May 16, 2013)

- Release of an updated version of the BankAccount.adt example.

Version 1.1 (July 5, 2013)

- Added attribute domain for "Difficulty for the proponent (L,M,H)"
- Added attribute domain for "Difficulty for the proponent (L,M,H,E)"
- Updated attribute domain "Minimal skill level needed for the proponent"
  - Default value for the proponent changed from ∞ to k
  - Improved description of the attribute on the right hand side of the *Add Attribute Domain* window
- Updated attribute domain for "Probability of success"
  - Rounding of inserted values removed
- Added *Help* menu item to the menubar
- Modified default name of the file for the *File, Save* option

Version 1.2 (October 1, 2013)

- Added a possibility to create multi-line node labels (see Section 3.5)
- Added the feature of exporting and importing trees written as XML files (see Section 6.1 and 7)
- Added the Remove Domain button to the attribute window
- Improved management of the Valuations View and Domain Details View windows
- Several bugs removed

Version 1.3 (December 6, 2013)

- Improving the time complexity of outputting an ADTree after having modified its ADTerm. The algorithm to calculate the tree edit distance has been changed from a cubic one to a suboptimal one but with complexity $\mathcal{O}(n^2)$. The new algorithm is based on Euler string for trees and Levenshtein distance [2].
- Enhancing import and export of XML files with a possibility to also include attribute values. For details see Sections 7.2 and 7.3.
- Added a possibility of using special characters in node labels. (Comma and parentheses are still not allowed.) For details, see line 22 of the schema given in Figure 12.
- Added possibility to (alphabetically) sort columns in the valuations view window.
- Added the feature of highlighting on the ADTree representation the node currently selected in the valuations view window.
- Implemented the keyboard shortcut [CTRL+Enter] to close the input dialog for editing node labels.

Version 1.4 (January 21, 2015)

- Added possibility of launching the program from the command line, see Section 1.2.
- Updated example "Breaking into a Warehouse" (typo).

9 Concluding Remarks

For further questions or suggestions concerning the ADTool, please contact the ATREES project team [5].

References

[1] abego Software. abego TreeLayout. `http://code.google.com/p/treelayout/`. Accessed November 5, 2013.

[2] Tatsuya Akutsu. A relation between edit distance for ordered trees and edit distance for Euler strings. *Inf. Process. Lett.*, 100(3):105–109, 2006.

[3] Stefano Bistarelli, Fabio Fioravanti, and Pamela Peretti. Defense Trees for Economic Evaluation of Security Investments. In *ARES*, pages 416–423. IEEE Computer Society, 2006.

[4] Kenneth S. Edge, George C. Dalton II, Richard A. Raines, and Robert F. Mills. Using Attack and Protection Trees to Analyze Threats and Defenses to Homeland Security. In *MILCOM*, pages 1–7. IEEE, 2006.

[5] Barbara Kordy, Piotr Kordy, Sjouke Mauw, Saša Radomirović, Patrick Schweitzer, and Jean-Paul Weber. The ATREES Project, funded by the Fonds National de la Recherche, Luxembourg under grants C08/IS/26 and PHD-09-167. `http://satoss.uni.lu/projects/atrees/`, 2009–2012. Accessed December 6, 2013.

[6] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. ADTool: Security Analysis with Attack–Defense Trees. In Kaustubh R. Joshi, Markus Siegle, Mariëlle Stoelinga, and Pedro R. D'Argenio, editors, *QEST*, volume 8054 of *Lecture Notes in Computer Science*, pages 173–176. Springer, 2013.

[7] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. ADTool: Security Analysis with Attack–Defense Trees (Extended Version). *CoRR*, abs/1305.6829, 2013.

[8] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Foundations of Attack–Defense Trees. In Pierpaolo Degano, Sandro Etalle, and Joshua D. Guttman, editors, *FAST*, volume 6561 of *LNCS*, pages 80–95. Springer, September 2010.

[9] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Attack–Defense Trees. *Journal of Logic and Computation*, pages 1–33, 2012. available online `http://logcom.oxfordjournals.org/content/early/2012/06/21/logcom.exs029.short?rss=1`.

[10] Barbara Kordy, Sjouke Mauw, and Patrick Schweitzer. Quantitative Questions on Attack–Defense Trees. In *ICISC*, volume 7839 of *LNCS*, pages 49–64. Springer, 2013.

[11] Piotr Kordy and Patrick Schweitzer. The ADTool. `http://satoss.uni.lu/members/piotr/adtool/index.php`, 2012. Accessed December 6, 2012.

[12] Sjouke Mauw and Martijn Oostdijk. Foundations of Attack Trees. In Dongho Won and Seungjoo Kim, editors, *ICISC*, volume 3935 of *LNCS*, pages 186–198. Springer, 2005.

[13] NNL Technology AB. InfoNode Docking Windows. `http://www.infonode.net/index.html?idw`. Accessed November 5, 2013.

[14] Oracle. Common Java Web Start Problems. `http://docs.oracle.com/javase/tutorial/deployment/webstart/problems.html`. Accessed November 5, 2013.

[15] Chris Salter, O. Sami Saydjari, Bruce Schneier, and Jim Wallner. Toward a secure system engineering methodolgy. In *Proceedings of the 1998 New Security Paradigms Workshop*, pages 2–10, 1998.

[16] TREsPASS. Technology-supported Risk Estimation by Predictive Assessment of Socio-technical Security, FP7 project, grant agreement 318003. Website: `http://www.trespass-project.eu/`, 2012–2016.