AT MAGI

Generated by Doxygen 1.9.1

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ctypes.Structure

- **at_magi.Struct.CustomList** **7**
- **at_magi.Struct.CustomNode** **7**
- **at_magi.Struct.EdgeNode** **7**
- **at_magi.Struct.FormulaNode** **8**
- **at_magi.Struct.FullList** **8**

at_magi.Benchmark.timeout **10**

QObject

- **at_magi.Comparison.Comparison** **4**
- **at_magi.Comparison.cmpWorker** **3**
- **at_magi.ParserWorker.ParserWorker** **8**

2 Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

3 File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

4 Class Documentation

4.1 at_magi.Comparison.cmpWorker Class Reference

Comparison Worker Class.

Public Member Functions

- def **run** (self)

Public Attributes

- **cnf**
- **list_var**
- **sol_array**

Static Public Attributes

- **finished** = pyqtSignal()

4.1.1 Detailed Description

Comparison Worker Class.

Compute the solutions of a given formula

The documentation for this class was generated from the following file:

- Comparison.py

4.2 at_magi.Comparison.Comparison Class Reference

Comparison Class.

Public Member Functions

- def **__init__** (self, parent=None)
- def tree_comparison (self, fileName1, fileName2, text1, text2)

  Launch each tree computation : Create Workers QThreads to compute each tree.
- def clean_Worker (self, nbr, text)

  Clean the SAT workers : Get results and delete workers.
- def compare (self)

  Launch the comparison : Create Workers QThreads to compute the comparison between the two trees.
- def clean_cmpWorker (self, nbr)

  Clean the comparison workers : Get results and delete workers.
- def subplot (self, node_list, edge_list, web, name)

  Plot Trees : Use results to save and plot the trees.
- def get_canvas (self, ln, le, filename)

  Creation of the Digraph using Networkx and Pyvis : Create graph from given information by adding logic nodes, Get the layout from Networkx and send it to a Pyvis network, Use settings for Pyvis from a JSON file and save the graph to a HTML file.

Public Attributes

- **formula1**
- **cnf1**
- **sol_array1**
- **var_array1**
- **formula2**
- **cnf2**
- **sol_array2**
- **var_array2**
- **formula3**
- **cnf3**
- **sol_array3**

- **var_array3**
- **sem**
- **worker1**
- **worker2**
- **cnt**
- **t1**
- **t2**
- **worker3**
- **t3**
- **formula4**
- **worker4**
- **t4**
- **boolean_sol_arr3**
- **cnf4**
- **var_array4**
- **sol_array4**
- **boolean_sol_arr4**

Static Public Attributes

- **finished** = pyqtSignal()

4.2.1 Detailed Description

Comparison Class.

Create and manage the Comparison of two attack trees

4.2.2 Member Function Documentation

4.2.2.1 clean_cmpWorker() `def at_magi.Comparison.Comparison.clean_cmpWorker (`

```
self,
nbr )
```

Clean the comparison workers : Get results and delete workers.

Parameters

self	The object pointer.
nbr	Number of the tree worker.

4.2.2.2 clean_Worker() `def at_magi.Comparison.Comparison.clean_Worker (`

```
self,
```

```
        nbr,
        text )
```

Clean the SAT workers : Get results and delete workers.

Parameters

self	The object pointer.
nbr	Number of the tree worker.
text	Pointer to the GUI text holder.

4.2.2.3 compare() `def at_magi.Comparison.Comparison.compare (`

```
        self )
```

Launch the comparison : Create Workers QThreads to compute the comparison between the two trees.

Parameters

self	The object pointer.

4.2.2.4 get_canvas() `def at_magi.Comparison.Comparison.get_canvas (`

```
        self,
        ln,
        le,
        filename )
```

Creation of the Digraph using Networkx and Pyvis : Create graph from given information by adding logic nodes, Get the layout from Networkx and send it to a Pyvis network, Use settings for Pyvis from a JSON file and save the graph to a HTML file.

Parameters

self	The object pointer.
ln	List of the Nodes of the JSON file
le	List of the Edges of the JSON file
filename	Name of the file

4.2.2.5 subplot() `def at_magi.Comparison.Comparison.subplot (`

```
        self,
        node_list,
        edge_list,
        web,
        name )
```

Plot Trees : Use results to save and plot the trees.

Parameters

self	The object pointer.
node_list	List of nodes.
edge_list	List of edges.
web	WebEngine.
name	Name of the file.

4.2.2.6 tree_comparison()

```
def at_magi.Comparison.Comparison.tree_comparison (
        self,
        fileName1,
        fileName2,
        text1,
        text2 )
```

Launch each tree computation : Create Workers QThreads to compute each tree.

Parameters

self	The object pointer.
fileName1	First tree filename.
fileName2	Second tree filename.
text1	Pointer to the GUI text holder for the first tree.
text2	Pointer to the GUI text holder for the second tree.

The documentation for this class was generated from the following file:

- Comparison.py

4.3 at_magi.Struct.CustomList Class Reference

The documentation for this class was generated from the following file:

- Struct.py

4.4 at_magi.Struct.CustomNode Class Reference

The documentation for this class was generated from the following file:

- Struct.py

4.5 at_magi.Struct.EdgeNode Class Reference

The documentation for this class was generated from the following file:

- Struct.py

4.6 at_magi.Struct.FormulaNode Class Reference

The documentation for this class was generated from the following file:

- Struct.py

4.7 at_magi.Struct.FullList Class Reference

The documentation for this class was generated from the following file:

- Struct.py

4.8 at_magi.ParserWorker.ParserWorker Class Reference

ParserWorker Class.

Public Member Functions

- def **get_file_so** (self)
- def **run** (self)

Public Attributes

- **file_so**
- **node_list**

Static Public Attributes

- **finished** = pyqtSignal(int)

4.8.1 Detailed Description

ParserWorker Class.

Create and manage the parsing of an attack tree from GUI given text

The documentation for this class was generated from the following file:

- ParserWorker.py

4.9 at_magi.SATWorker.SATWorker Class Reference

SATWorker Class.

Public Member Functions

- def **run** (self)
- def **get_file_so** (self)
- def **working** (self)
- def **start_with_assumptions** (self)

Public Attributes

- **file_so**
- **node_list**
- **edge_list**
- **formula**
- **formula_cm**
- **str_formula**
- **str_formula_cm**
- **str_cnf**
- **str_cnf_cm**
- **uniq_node_list**
- **uniq_node_list_cm**
- **sol_array**

Static Public Attributes

- **finished** = pyqtSignal()
- **finishedWithError** = pyqtSignal()

4.9.1 Detailed Description

SATWorker Class.

The documentation for this class was generated from the following file:

- SATWorker.py

4.10 at_magi.SMTWorker.SMTWorker Class Reference

SMTWorker Class.

Public Member Functions

- def **run** (self)

Public Attributes

- **type**
- **values_array**

Static Public Attributes

- **finished** = pyqtSignal()

4.10.1 Detailed Description

SMTWorker Class.

The documentation for this class was generated from the following file:

- SMTWorker.py

4.11 at_magi.Benchmark.timeout Class Reference

Public Member Functions

- def **__init__** (self, seconds=1, error_message='Timeout')
- def **handle_timeout** (self, signum, frame)
- def **__enter__** (self)
- def **__exit__** (self, type, value, traceback)

Public Attributes

- **seconds**
- **error_message**

The documentation for this class was generated from the following file:

- Benchmark.py

4.12 at_magi.ATMAGI.Window Class Reference

Window Class.

Public Member Functions

- def __init__ (self, parent=None)

  The constructor.
- def get_canvas (self, ln, le)

  Creation of the Digraph using Networkx and Pyvis : Create graph from given information by adding logic nodes, Get the layout from Networkx and send it to a Pyvis network, Use settings for Pyvis from a JSON file and save the graph to a HTML file.
- def plot (self, node_list, edge_list)

  Action called at the end of the import process : Clear the figure of the GUI, launch the html graph creation from the nodes and edges, retrieves the html and loads it on the canvas.
- def getfileJSON (self, already_chosen=False)

  Action called by the import JSON button : Use a file explorer to choose the JSON file to import Create a new thread for the Worker class.
- def stopImport (self, proper_close)

  Action called on the worker finished or finishedWithError signals Clean resources and enable Import and Reload Buttons.
- def getfileGrammar (self)

  Action called by the import Grammar button : Use a file explorer to choose the TXT file to import Set the text in the corresponding QTextEdit.
- def outputCNFformula (self)

  Action called by the CNF Formula button : Set the output formula to its CNF form.
- def outputCompleteformula (self)

  Action called by the Complete Formula button : Set the output formula to its Complete form.
- def outputSolution (self)

  Action called by the Solve button : Get the index of the solution from the QSpinBox Create a new graph HTML with the nodes taken from the solution and put them in a specific group to highlight them.
- def recur_path (self, current_edge, path_count_set, disabled_node, taken)

  Recursive iteration on the nodes : Recursively goes up in the tree by iterating on the edges Set node taken in a style group to color them.
- def outputClear (self)

  Action called by the Clear button : Clear the output and reload the graph from the HTML file.
- def getRandomTree (self)

  Action called by the Random Tree button : Get the values from the three QSpinBox below the button Set the grammar text with the generated strings Parse the grammar.
- def outputUsingAssumptions (self)

  Action called by the Fix Input button :
  Create QGridLayout pop-up to help the user toggle nodes and CMs
  Recompute the solutions for this graph with the new assumptions.
- def changeState (self, coord, type)

  Change State of QGridLayout Elements
  States possible are : undefined, true, false.
- def changeStateCounter (self, state, coord)

  Change State of CM element and changes the depending leaf nodes of the grid accordingly
  Keep counter of the number of nodes affected to see if a recomputation is needed.
- def computeUsingAssumptions (self)

  Recompute the tree solutions using the assumptions if needed :
  Launch a new Worker with the assumptions list.
- def show_nx_nodes (self)

  Action called by the nx nodes button : Print the nodes in a pop-up QListWidget.
- def show_nx_edges (self)

  Action called by the nx edges button : Print the edges in a pop-up QListWidget.
- def show_sol (self)

  Action called by the solutions list button : Print the list of solutions in a QDialog.

- def setCNFTransform (self, bool)

  Action called by two CNF transform buttons : Set the type of CNF transformation used by the Worker.
- def callSMT (self, type=0)

  Action called by min cost and max proba buttons : Create the Qthread and SMTWorker to compute the solutions needed.
- def SMTcleaning (self, type)

  Called by the finished signal of the SMTWorker : Print and Store the solutions found, then clean the resources.
- def compareTrees (self)

  Action called by the Comparison button : Create a new QDialog with the needed elements to plot the trees and their comparison Then call self.call_compare with the corresponding arguments.
- def call_compare (self, form1, form2, form3, web1, web2, path1, path2, sol1, sol2, results_print)

  Create the Comparison Object and call its tree_comparison method :
- def showResults (self)

  Add comparator solutions to the designated QWidget :
- def compareFrequency (self)

  Action called by the Node Frequencies button :
- def cleaning (self, bool_plot=0)

  Get the Worker results and update them in the Window before deleting :
- def parser (self)

  Action called by the parser button : Create a ParserWorker thread to create a tree from the grammar given in the GUI.
- def enable_parser (self)

  Enable buttons when processing is over.
- def reduceSolutions (self)

  Action of the Use reduced solutions buttons Use a new list of filtered solutions which are not redundant.
- def show_popup (self, error_id)

  Show Error Pop-up QMessageBox : Message of a detected error and its type.

Public Attributes

- **width**
- **height**
- **canvas**
- **buttonImportJson**
- **buttonImportGrammar**
- **buttonParse**
- **jsonName**
- **pathFile**
- **tracesFound**
- **tableSol**
- **buttonReload**
- **sol_button**
- **sol_spin**
- **clear_button**
- **reduce_button**
- **cnf_button**
- **complete_button**
- **max_spin**
- **ouputAssumption_button**
- **cost_button**
- **max_cost**
- **cost_label**
- **proba_button**

- **min_proba**
- **proba_label**
- **rndtree_button**
- **rnd_spin_1**
- **rnd_spin_2**
- **rnd_spin_3**
- **result_layout**
- **grammarText**
- **curr_formula**
- **curr_cnf**
- **sol_array**
- **uniq_node_list**
- **uniq_node_list_cm**
- **useTseitin**
- **values_array**
- **current_network**
- **current_digraph**
- **thread**
- **worker**
- **cur_net_nodes_dict**
- **cur_digraph_nodes_dict**
- **popup_assumpt**
- **mandatory_cm_counter**
- **grid_fix_input**
- **grid_fix_input_cm**
- **msg**
- **var_array**
- **boolean_sol_arr**
- **boolean_sol_array_full**
- **boolean_sol_array_reduced**
- **comp**
- **comparator**
- **curr_formula_cm**
- **curr_cnf_cm**
- **basic_nodes**
- **basic_edges**
- **parser_thread**
- **parser_worker**

4.12.1 Detailed Description

Window Class.

Main GUI interface of the application

4.12.2 Member Function Documentation

4.12.2.1 call_compare() def at_magi.ATMAGI.Window.call_compare (

```
self,
form1,
form2,
form3,
web1,
web2,
path1,
path2,
sol1,
sol2,
results_print )
```

Create the Comparison Object and call its tree_comparison method :

Parameters

self	The object pointer.
form1	QTextEdit for the first tree formula.
form2	QTextEdit for the second tree formula.
form3	QTextEdit for the conjunction formula of both trees.
web1	First QWebEngineView.
web2	Second QWebEngineView.
path1	First File Path.
path2	Second File Path.
sol1	QTextEdit for the solutions found for the first tree.
sol2	QTextEdit for the solutions found for the second tree.
results_print	QWidget used for the solutions comparison.

4.12.2.2 callSMT() def at_magi.ATMAGI.Window.callSMT (

```
self,
type = 0 )
```

Action called by min cost and max proba buttons : Create the Qthread and SMTWorker to compute the solutions needed.

Parameters

self	The object pointer.
type	Integer value corresponding to the type of SMT operation (0:cost 1:proba)

4.12.2.3 changeState() def at_magi.ATMAGI.Window.changeState (

```
self,
coord,
type )
```

Change State of QGridLayout Elements
States possible are : undefined, true, false.

Parameters

self	The object pointer.
coord	The coordinates of the element in the grid.
type	Type of the element : CM or leaf.

4.12.2.4 changeStateCounter() `def at_magi.ATMAGI.Window.changeStateCounter (`

```
self,
state,
coord )
```

Change State of CM element and changes the depending leaf nodes of the grid accordingly
Keep counter of the number of nodes affected to see if a recomputation is needed.

Parameters

self	The object pointer.
state	State of the current CM node.
coord	The coordinates of the element in the grid.

4.12.2.5 cleaning() `def at_magi.ATMAGI.Window.cleaning (`

```
self,
bool_plot = 0 )
```

Get the Worker results and update them in the Window before deleting :

Parameters

self	The object pointer.
bool_plot	Boolean value used to print formula and plot graph.

4.12.2.6 compareFrequency() `def at_magi.ATMAGI.Window.compareFrequency (`

```
self )
```

Action called by the Node Frequencies button :

Parameters

self	The object pointer.

4.12.2.7 compareTrees() def at_magi.ATMAGI.Window.compareTrees (self)

Action called by the Comparison button : Create a new QDialog with the needed elements to plot the trees and their comparison Then call self.call_compare with the corresponding arguments.

Parameters

self	The object pointer.

4.12.2.8 computeUsingAssumptions() def at_magi.ATMAGI.Window.computeUsingAssumptions (self)

Recompute the tree solutions using the assumptions if needed :
Launch a new Worker with the assumptions list.

Parameters

self	The object pointer.

4.12.2.9 enable_parser() def at_magi.ATMAGI.Window.enable_parser (self)

Enable buttons when processing is over.

Parameters

self	The object pointer.

4.12.2.10 get_canvas() def at_magi.ATMAGI.Window.get_canvas (self, ln, le)

Creation of the Digraph using Networkx and Pyvis : Create graph from given information by adding logic nodes, Get the layout from Networkx and send it to a Pyvis network, Use settings for Pyvis from a JSON file and save the graph to a HTML file.

Parameters

self	The object pointer.
ln	List of the Nodes of the JSON file
le	List of the Edges of the JSON file

4.12.2.11 getfileGrammar() def at_magi.ATMAGI.Window.getfileGrammar (self)

Action called by the import Grammar button : Use a file explorer to choose the TXT file to import Set the text in the corresponding QTextEdit.

Parameters

self	The object pointer.

4.12.2.12 getfileJSON() def at_magi.ATMAGI.Window.getfileJSON (self, already_chosen = False)

Action called by the import JSON button : Use a file explorer to choose the JSON file to import Create a new thread for the Worker class.

Parameters

self	The object pointer.

4.12.2.13 getRandomTree() def at_magi.ATMAGI.Window.getRandomTree (self)

Action called by the Random Tree button : Get the values from the three QSpinBox below the button Set the grammar text with the generated strings Parse the grammar.

Parameters

self	The object pointer.

4.12.2.14 outputClear() def at_magi.ATMAGI.Window.outputClear (self)

Action called by the Clear button : Clear the output and reload the graph from the HTML file.

Parameters

self	The object pointer.

4.12.2.15 outputCNFformula() def at_magi.ATMAGI.Window.outputCNFformula (self)

Action called by the CNF Formula button : Set the output formula to its CNF form.

Parameters

self	The object pointer.

4.12.2.16 outputCompleteformula() def at_magi.ATMAGI.Window.outputCompleteformula (self)

Action called by the Complete Formula button : Set the output formula to its Complete form.

Parameters

self	The object pointer.

4.12.2.17 outputSolution() def at_magi.ATMAGI.Window.outputSolution (self)

Action called by the Solve button : Get the index of the solution from the QSpinBox Create a new graph HTML with the nodes taken from the solution and put them in a specific group to highlight them.

Parameters

self	The object pointer.

4.12.2.18 outputUsingAssumptions() def at_magi.ATMAGI.Window.outputUsingAssumptions (self)

Action called by the Fix Input button :
Create QGridLayout pop-up to help the user toggle nodes and CMs
Recompute the solutions for this graph with the new assumptions.

Parameters

self	The object pointer.

4.12.2.19 parser() def at_magi.ATMAGI.Window.parser (self)

Action called by the parser button : Create a ParserWorker thread to create a tree from the grammar given in the GUI.

Parameters

self	The object pointer.

4.12.2.20 plot() def at_magi.ATMAGI.Window.plot (self, node_list, edge_list)

Action called at the end of the import process : Clear the figure of the GUI, launch the html graph creation from the nodes and edges, retrieves the html and loads it on the canvas.

Parameters

self	The object pointer.
node_list	List of the Nodes of the JSON file
node_list	List of the Edges of the JSON file

4.12.2.21 recur_path() def at_magi.ATMAGI.Window.recur_path (self, current_edge, path_count_set, disabled_node, taken)

Recursive iteration on the nodes : Recursively goes up in the tree by iterating on the edges Set node taken in a style group to color them.

Parameters

self	The object pointer.
current_edge	Current edge to evaluate.
path_count_set	Dictionary of nodes found with a counter to enable them if needed or block the recursion.
disabled_node	Set of nodes which can be used.

4.12.2.22 reduceSolutions() def at_magi.ATMAGI.Window.reduceSolutions (self)

Action of the Use reduced solutions buttons Use a new list of filtered solutions which are not redundant.

Parameters

self	The object pointer.

4.12.2.23 setCNFTransform() def at_magi.ATMAGI.Window.setCNFTransform (self, bool)

Action called by two CNF transform buttons : Set the type of CNF transformation used by the Worker.

Parameters

self	The object pointer.
bool	The boolean value of Tseitin encoding usage.

4.12.2.24 show_nx_edges() def at_magi.ATMAGI.Window.show_nx_edges (self)

Action called by the nx edges button : Print the edges in a pop-up QListWidget.

Parameters

self	The object pointer.

4.12.2.25 show_nx_nodes() def at_magi.ATMAGI.Window.show_nx_nodes (self)

Action called by the nx nodes button : Print the nodes in a pop-up QListWidget.

Parameters

self	The object pointer.

4.12.2.26 show_popup() def at_magi.ATMAGI.Window.show_popup (

```
        self,
        error_id )
```

Show Error Pop-up QMessageBox : Message of a detected error and its type.

Parameters

self	The object pointer.
error	The id of the error.

4.12.2.27 show_sol() def at_magi.ATMAGI.Window.show_sol (

```
        self )
```

Action called by the solutions list button : Print the list of solutions in a QDialog.

Parameters

self	The object pointer.

4.12.2.28 showResults() def at_magi.ATMAGI.Window.showResults (

```
        self )
```

Add comparator solutions to the designated QWidget :

Parameters

self	The object pointer.

4.12.2.29 SMTcleaning() def at_magi.ATMAGI.Window.SMTcleaning (

```
        self,
        type )
```

Called by the finished signal of the SMTWorker : Print and Store the solutions found, then clean the resources.

Parameters

self	The object pointer.
type	Integer value corresponding to the type of SMT operation (0:cost 1:proba)

4.12.2.30 stopImport() def at_magi.ATMAGI.Window.stopImport (

```
        self,
        proper_close )
```

Action called on the worker finished or finishedWithError signals Clean resources and enable Import and Reload Buttons.

Parameters

self	The object pointer.
proper_close	Boolean value used in case of error in the worker.

The documentation for this class was generated from the following file:

- ATMAGI.py

5 File Documentation

5.1 ATMAGI.py File Reference

Main file of the python GUI interface Create GUI using PyQt5.

Classes

- class at_magi.ATMAGI.Window
  Window Class.

Functions

- def at_magi.ATMAGI.start ()
  driver code

Variables

- **at_magi.ATMAGI.dirname** = os.path.dirname(__file__)

5.1.1 Detailed Description

Main file of the python GUI interface Create GUI using PyQt5.

5.2 Benchmark.py File Reference

Comparison of the methods used to compute CNF transformation.

Classes

- class at_magi.Benchmark.timeout

Functions

- def at_magi.Benchmark.compute_truthtable (expr, find_all)

  Compute the truth table of a given formula : Imported from Sympy extension and lightly modified.
- def at_magi.Benchmark.benchmark ()

  Benchmark of the three methods : Uses multiple set of expressions to compare Truth Table, Basic Conversion and Tseitin Transform.

5.2.1 Detailed Description

Comparison of the methods used to compute CNF transformation.

5.2.2 Function Documentation

5.2.2.1 compute_truthtable() `def at_magi.Benchmark.compute_truthtable ( expr, find_all )`

Compute the truth table of a given formula : Imported from Sympy extension and lightly modified.

Parameters

expr	Formula to evaluate.
find_all	Used tp return the first True ouput.

5.3 Comparison.py File Reference

Class Comparison Class cmpWorker Use two JSON files to compare two trees Compute theme separatly then compute their conjunction and disjunction to compare their solutions.

Classes

- class at_magi.Comparison.Comparison

  Comparison Class.
- class at_magi.Comparison.cmpWorker

  Comparison Worker Class.

Variables

- **at_magi.Comparison.dirname** = os.path.dirname(__file__)

5.3.1 Detailed Description

Class Comparison Class cmpWorker Use two JSON files to compare two trees Compute theme separatly then compute their conjunction and disjunction to compare their solutions.

5.4 FreqComparator.py File Reference

Plot the solutions of a Tree On the left : create a tree with nodes having size corresponding to the number of times they where taken On the right : a bar diagram of the number of time each node is taken.

Functions

- def at_magi.FreqComparator.frequency_comparator (nodes, edges, current_network, current_digraph, sol↩ _array, var_array)

  Launch frequency comparator :
- def at_magi.FreqComparator.compute_values (nodes, current_network, current_digraph, sol_array, var_↩ array)

  Compute frequencies by iterating the tree :
- def at_magi.FreqComparator.recur_path (current_edge, path_count_set, taken, cur_net_nodes_dict, cur_↩ digraph_nodes_dict, current_network, values, values_logic)

  Recursively iterate the tree :

5.4.1 Detailed Description

Plot the solutions of a Tree On the left : create a tree with nodes having size corresponding to the number of times they where taken On the right : a bar diagram of the number of time each node is taken.

5.5 GlobalProba.py File Reference

Compute the global probability of success of any attack.

Functions

- def at_magi.GlobalProba.get_global_proba (ln, le)

  Compute Global Probability of Successful Attack :
- def **at_magi.GlobalProba.test** ()

5.5.1 Detailed Description

Compute the global probability of success of any attack.

5.5.2 Function Documentation

5.5.2.1 get_global_proba() `def at_magi.GlobalProba.get_global_proba ( ln, le )`

Compute Global Probability of Successful Attack :

Parameters

ln	List of the Nodes
le	List of the Edges

5.6 ParserWorker.py File Reference

Class ParserWorker used to send the text representation of an attack tree to the C parser.

Classes

- class at_magi.ParserWorker.ParserWorker

  ParserWorker Class.

5.6.1 Detailed Description

Class ParserWorker used to send the text representation of an attack tree to the C parser.

5.7 RandomTree.py File Reference

Random Tree Generation with Grammar.

Functions

- def at_magi.RandomTree.nodeGeneration (Relations, CounterMeasures, Properties, node, depth, maxdepth, branching_factor)

  recursive tree generator 1 This recursive generator will generate a simple tree using the grammar The maxdepth and branchingfactor arguments will forge the global shape of the tree
- def at_magi.RandomTree.nodeGeneration2 (NodeList, Relations, Properties, node, depth, maxdepth, branching_factor)

  recursive tree generator 2 This recursive generator will generate a simple tree using the grammar
  The maxdepth and branchingfactor arguments will forge the global shape of the tree The generation is more complex and can generate shared childrens between the nodes (this function does not produce countermeasures)
- def at_magi.RandomTree.CMGeneration2 (NodeList, CounterMeasures)

  recursive countermeasure generator 2 This recursive countermeasure generator will generate shared countermeasures with the list of node of the already generated tree
- def at_magi.RandomTree.nodeGeneration3 (NodeList, Relations, Properties, node, depth, maxdepth, branching_factor)

  recursive tree generator 3 This recursive generator will generate a simple tree using the grammar
  The maxdepth and branchingfactor arguments will forge the global shape of the tree The generation is more complex and can generate shared childrens between the nodes (this function does not produce countermeasures) This functions creates negative leaf nodes with a probability 1/6
- def at_magi.RandomTree.CMGeneration3 (NodeList, CounterMeasures)

  recursive countermeasure generator 3 This recursive countermeasure generator will generate shared countermeasures with the list of node of the already generated tree
- def at_magi.RandomTree.nodeGeneration4 (NodeList, Relations, Properties, node, depth, maxdepth, branching_factor)

recursive tree generator 4 This recursive generator will generate a simple tree using the grammar
The maxdepth and branchingfactor arguments will forge the global shape of the tree The generation is more complex and can generate shared childrens between the nodes The generated tree can be inconsistent (this function does not produce countermeasures)

- def at_magi.RandomTree.CMGeneration4 (NodeList, CounterMeasures)

  recursive countermeasure generator 4 This recursive countermeasure generator will generate shared countermeasures with the list of node of the already generated tree
- def at_magi.RandomTree.ShuffledNodeGeneration (NodeList, Relations, Properties, node, depth, maxdepth, branching_factor)

  recursive tree generator 5 base on the generator 3 This recursive generator will generate a simple tree using the grammar
  The maxdepth and branchingfactor arguments will forge the global shape of the tree The generation is more complex and can generate shared childrens between the nodes (this function does not produce countermeasures) This functions creates negative leaf nodes with a probability 1/6 The grammar lines of the tree are shuffled
- def at_magi.RandomTree.CMShuffledGeneration (NodeList, CounterMeasures)

  recursive countermeasure generator 3 This recursive countermeasure generator will generate shared countermeasures with the list of node of the already generated tree
- def at_magi.RandomTree.TreeGen (maxdepth, branching_factor, complexity)

  Main function to call to generate a random tree with the grammar.

5.7.1 Detailed Description

Random Tree Generation with Grammar.

5.7.2 Function Documentation

5.7.2.1 CMGeneration2() `def at_magi.RandomTree.CMGeneration2 ( NodeList, CounterMeasures )`

recursive countermeasure generator 2 This recursive countermeasure generator will generate shared countermeasures with the list of node of the already generated tree

Parameters

NodeList	: list of nodes of the generated tree
CounterMeasures	: the string of CounterMeasures

5.7.2.2 CMGeneration3() `def at_magi.RandomTree.CMGeneration3 ( NodeList, CounterMeasures )`

recursive countermeasure generator 3 This recursive countermeasure generator will generate shared countermeasures with the list of node of the already generated tree

Parameters

NodeList	: list of nodes of the generated tree
CounterMeasures	: the string of CounterMeasures

5.7.2.3 CMGeneration4() `def at_magi.RandomTree.CMGeneration4 ( NodeList, CounterMeasures )`

recursive countermeasure generator 4 This recursive countermeasure generator will generate shared countermeasures with the list of node of the already generated tree

Parameters

NodeList	: list of nodes of the generated tree
CounterMeasures	: the string of CounterMeasures

5.7.2.4 CMShuffledGeneration() `def at_magi.RandomTree.CMShuffledGeneration ( NodeList, CounterMeasures )`

recursive countermeasure generator 3 This recursive countermeasure generator will generate shared countermeasures with the list of node of the already generated tree

Parameters

NodeList	: list of nodes of the generated tree
CounterMeasures	: the string of CounterMeasures

5.7.2.5 nodeGeneration() `def at_magi.RandomTree.nodeGeneration ( Relations, CounterMeasures, Properties, node, depth, maxdepth, branching_factor )`

recursive tree generator 1 This recursive generator will generate a simple tree using the grammar The maxdepth and branchingfactor arguments will forge the global shape of the tree

Parameters

Relations	: the string of relations

Parameters

CounterMeasures	: the string of CounterMeasures
Properties	: the string of Properties
node	: the string naming the parent of the node (used to gerenate its own name)
depth	: the actual depth of the recursive calls
maxdepth	: the maximum depth of the tree
branching_factor	: the maximu branching factor of each node

5.7.2.6 nodeGeneration2()

```
def at_magi.RandomTree.nodeGeneration2 (
        NodeList,
        Relations,
        Properties,
        node,
        depth,
        maxdepth,
        branching_factor )
```

recursive tree generator 2 This recursive generator will generate a simple tree using the grammar
The maxdepth and branchingfactor arguments will forge the global shape of the tree The generation is more complex and can generate shared childrens between the nodes (this function does not produce countermeasures)

Parameters

NodeList	: the list of nodes of the tree
Relations	: the string of relations
Properties	: the string of Properties
node	: the string naming the parent of the node (used to gerenate its own name)
depth	: the actual depth of the recursive calls
maxdepth	: the maximum depth of the tree
branching_factor	: the maximum branching factor of each node

5.7.2.7 nodeGeneration3()

```
def at_magi.RandomTree.nodeGeneration3 (
        NodeList,
        Relations,
        Properties,
        node,
        depth,
        maxdepth,
        branching_factor )
```

recursive tree generator 3 This recursive generator will generate a simple tree using the grammar
The maxdepth and branchingfactor arguments will forge the global shape of the tree The generation is more complex and can generate shared childrens between the nodes (this function does not produce countermeasures) This functions creates negative leaf nodes with a probability 1/6

Parameters

NodeList	: the list of nodes of the tree
Relations	: the string of relations
Properties	: the string of Properties
node	: the string naming the parent of the node (used to gerenate its own name)
depth	: the actual depth of the recursive calls
maxdepth	: the maximum depth of the tree
branching_factor	: the maximum branching factor of each node

5.7.2.8 nodeGeneration4()

```
def at_magi.RandomTree.nodeGeneration4 (
        NodeList,
        Relations,
        Properties,
        node,
        depth,
        maxdepth,
        branching_factor )
```

recursive tree generator 4 This recursive generator will generate a simple tree using the grammar
The maxdepth and branchingfactor arguments will forge the global shape of the tree The generation is more complex and can generate shared childrens between the nodes The generated tree can be inconsistent (this function does not produce countermeasures)

Parameters

NodeList	: the list of nodes of the tree
Relations	: the string of relations
Properties	: the string of Properties
node	: the string naming the parent of the node (used to gerenate its own name)
depth	: the actual depth of the recursive calls
maxdepth	: the maximum depth of the tree
branching_factor	: the maximum branching factor of each node

5.7.2.9 ShuffledNodeGeneration()

```
def at_magi.RandomTree.ShuffledNodeGeneration (
        NodeList,
        Relations,
        Properties,
        node,
        depth,
        maxdepth,
        branching_factor )
```

recursive tree generator 5 base on the generator 3 This recursive generator will generate a simple tree using the grammar
The maxdepth and branchingfactor arguments will forge the global shape of the tree The generation is more complex and can generate shared childrens between the nodes (this function does not produce countermeasures) This functions creates negative leaf nodes with a probability 1/6 The grammar lines of the tree are shuffled

Parameters

NodeList	: the list of nodes of the tree
Relations	: the string of relations
Properties	: the string of Properties
node	: the string naming the parent of the node (used to gerenate its own name)
depth	: the actual depth of the recursive calls
maxdepth	: the maximum depth of the tree
branching_factor	: the maximum branching factor of each node

5.7.2.10 TreeGen() `def at_magi.RandomTree.TreeGen ( maxdepth, branching_factor, complexity )`

Main function to call to generate a random tree with the grammar.

Parameters

maxdepth	: the maximum depth of he tree
branching_factor	: the branching factor of each node
complexity	= 1 : relations + Countermeasures + Properties
complexity	= 2 : 1 + shared child + shared CM
complexity	= 3 : 2 + negative leaves (NOT)
complexity	= 4 : 2 + probably inconsistent tree
complexity	= 5 : 3 + shuffled grammar sentences

5.8 SATsolver.py File Reference

Solve the boolean formula.

Functions

- def at_magi.SATsolver.sat_solver (formula, list_var, assumptions=[ ], max_val=20, to_print=True)

  SAT solving method :

Variables

- list **at_magi.SATsolver.list_var** = ["node", "node2"]
- string **at_magi.SATsolver.formula** = " ((node) & (node2)) "
- dictionary **at_magi.SATsolver.glob** = {}
- **at_magi.SATsolver.parsed_formula** = parse_expr(formula, global_dict=glob)
- **at_magi.SATsolver.cnf_formula** = to_cnf(parsed_formula)

5.8.1 Detailed Description

Solve the boolean formula.

5.9 SATWorker.py File Reference

Retrieve the ctype Structure representing the tree Retrieve the tree boolean formula Use a Sat-Solver to solve the formula.

Classes

- class at_magi.SATWorker.SATWorker
  SATWorker Class.

Variables

- string **at_magi.SATWorker.str_formula** = "a & B & C_C | E | SN | C"
- dictionary **at_magi.SATWorker.glob** = {}
- **at_magi.SATWorker.parsed_formula** = parse_expr(str_formula, global_dict=glob)

5.9.1 Detailed Description

Retrieve the ctype Structure representing the tree Retrieve the tree boolean formula Use a Sat-Solver to solve the formula.

5.10 SMTsolver.py File Reference

Solve the boolean formula in term of costs or probabilities.

Functions

- def at_magi.SMTsolver.SMTformatting (solutions, list_var)
  SMT solutions formatting :
- def at_magi.SMTsolver.SMTcost (list_var, list_cost, formula, upper_bound=None)
  SMT cost solving method :
- def at_magi.SMTsolver.SMTproba (list_var, list_proba, formula, lower_bound=None)
  SMT proba solving method :

Variables

- list **at_magi.SMTsolver.list_var** = ["X1", "X2", "X3", "X4"]
- list **at_magi.SMTsolver.list_cost** = [3, 1, 2, 2]
- list **at_magi.SMTsolver.list_proba** = [0.7, 0.66, 0.5, 0.5]
- string **at_magi.SMTsolver.formula** = " (X1 | X2) & (X3 ^ X4) "
- **at_magi.SMTsolver.cost_max** = Fraction(str(6))
- **at_magi.SMTsolver.proba_min** = Fraction(str(0.3))

5.10.1 Detailed Description

Solve the boolean formula in term of costs or probabilities.

5.11 SMTWorker.py File Reference

Retrieve the tree boolean formula Use a SMT-Solver to solve the formula in term of costs or probabilities.

Classes

- class at_magi.SMTWorker.SMTWorker
  SMTWorker Class.

5.11.1 Detailed Description

Retrieve the tree boolean formula Use a SMT-Solver to solve the formula in term of costs or probabilities.

5.12 SolutionSorter.py File Reference

Methods used to sort and filter the list of list of solutions.

Functions

- def **at_magi.SolutionSorter.sorter** (l)
- def **at_magi.SolutionSorter.cmp** (l1, l2, ln)

Variables

- list **at_magi.SolutionSorter.test** = [ ]
- list **at_magi.SolutionSorter.test2** = [ ]

5.12.1 Detailed Description

Methods used to sort and filter the list of list of solutions.

5.13 Struct.py File Reference

Ctypes structures used by the python Worker used to retrieve and manipulate C structures.

Classes

- class at_magi.Struct.CustomNode
- class at_magi.Struct.EdgeNode
- class at_magi.Struct.FormulaNode
- class at_magi.Struct.CustomList
- class at_magi.Struct.FullList

5.13.1 Detailed Description

Ctypes structures used by the python Worker used to retrieve and manipulate C structures.

5.14 Tseitin.py File Reference

Methods use to compute the Tseitin transformation used to convert a boolan formula to its CNF-form.

Functions

- def at_magi.Tseitin.recur_formula (formula, list_subformulas, dict_subs, var_cnt, set_var)

  Create sub-expressions auxiliary variables :
- def at_magi.Tseitin.tseitin (formula)

  Compute Tseitin Transformation :
- def **at_magi.Tseitin.test** ()

5.14.1 Detailed Description

Methods use to compute the Tseitin transformation used to convert a boolan formula to its CNF-form.

5.14.2 Function Documentation

5.14.2.1 recur_formula() `def at_magi.Tseitin.recur_formula (`

```
formula,
list_subformulas,
dict_subs,
var_cnt,
set_var )
```

Create sub-expressions auxiliary variables :

Parameters

formula	The Formula to convert to its CNF form.
list_subformulas	List of subformulas.
dict_subs	Dictionary of sub-expressions correspondence.
var_cnt	Count sub-expressions for naming.
set_var	Set of variables.

5.14.2.2 tseitin() `def at_magi.Tseitin.tseitin ( formula )`

Compute Tseitin Transformation :

Parameters

formula	The Formula to convert to its CNF form.

Index