Model Checking for tcc Calculus Documentation

Release 1.0

Jaime E. Arias Almeida

November 15, 2012

CONTENTS

Contents:

CHAPTER

ONE

FORMULA

This module contains the class to describe a temporal formula.

class formula.**Formula**(*data*)

This class represents a temporal formula.

Parameters **data** (*Dictionary.*) – Structure representing the temporal formula.

Example

$\phi = \Diamond(\mathtt{in} = \mathtt{true} \wedge \neg \circ (\mathtt{x} = 2))$

```
>>> from formula import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
```

Note: Logic operators are represented by the following symbols:

- Globally : []
- Future : <>
- Next : o
- Negation : ~
- Or : v
- And : ^

---

getConnective()

Return the main connective of the formula.

Returns A string representing the main connective of the formula.

Return type String.

Example

$\phi = \Diamond(\mathtt{in} = \mathtt{true} \wedge \neg \circ (\mathtt{x} = 2))$ $\qquad getConnective(\phi) = \Diamond$

```
>>> from formula import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> phi.getConnective()
'<>'
```

getConsistentPropositions()

Returns the consistent propositions of a formula.

Returns A structure representing the consistent proposition of the formula.

Return type Dictionary.

Example

$\phi = (x = 2)$ $consistentPropositions(\phi) = \neg(x = 1)$

```
>>> from formula import *
>>> phi = Formula({"": "x=2"})
>>> phi.getConsistentPropositions()
{'~': 'x=1'}
```

getFormula()

Returns the formula.

Returns A structure representing the formula.

Return type Dictionary.

Example

```
>>> from formula import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> phi.getFormula()
{'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}
```

getNegation()

Returns the negation of the formula.

Returns The negation of the formula.

Return type Formula.

Example

$\phi = \circ(\mathtt{x} = 2)$ $\neg\phi = \neg \circ (\mathtt{x} = 2)$

```
>>> from formula import *
>>> phi = Formula({"o":"x=2"})
>>> negPhi = phi.getNegation()
>>> negPhi.getFormula()
{'~': {'o': 'x=2'}}
```

getPropositionRules()

Returns the consistent propositions of all propositions in the implementation.

Example

Returns A dictionary containing as key a proposition, and value all possible propositions that are consistent.

Return type Dictionary.

```
>>> from formula import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> phi.getPropositionRules()
{'x=1': {'~': 'x=2'}, 'x=2': {'~': 'x=1'}}
```

getSubFormulas()

Returns the subformulas attached to a binary operator.

Returns A list containing the subformulas.

Return type List.

Example

$$\alpha = (\texttt{in} = \texttt{true}) \wedge \neg \circ (\texttt{x} = 2) \qquad \phi = (\texttt{in} = \texttt{true}) \qquad \psi = \neg \circ (\texttt{x} = 2)$$

```
>>> from formula import *
>>> alpha = Formula({"^":{"":"in=true","~":{"o":"x=2"}}})
>>> subformulas = alpha.getSubFormulas()
>>> for subformula in subformulas:
...     print subformula.getFormula()
{'': 'in=true'}
{'~': {'o': 'x=2'}}
```

getValues()

Returns the formula without the outermost unary operator.

Returns A structure representing the formula without the outermost unary operator.

Return type Dictionary

Example

$$\phi = \Diamond(\texttt{in} = \texttt{true} \wedge \neg \circ (\texttt{x} = 2)) \qquad getValues(\phi) = (\texttt{in} = \texttt{true}) \wedge \neg \circ (\texttt{x} = 2)$$

```
>>> from formula import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> phi.getValues()
{'^': {'': 'in=true', '~': {'o': 'x=2'}}}
```

isBasic()

Checks if the formula is a basic formula (i.e. proposition or it has $\circ$ as main connective)

Returns `True` if the formula is a basic formula or `False` otherwise.

Return type Boolean.

Example

```
>>> from formula import *
>>> phi = Formula({"o":"x=2"})
>>> phi.isBasic()
True
```

isNegativeFormula()

Returns if the formula has $\neg$ as main connective.

Returns `True` if the formula has $\neg$ as main connective or `False` otherwise.

Return type Boolean.

Example

```
>>> from formula import *
>>> phi = Formula({"~":{"o":"x=2"}})
>>> phi.isNegativeFormula()
True
```

isNegativeNext()

Checks if the formula is of the form $\neg \circ \phi$.

Returns `True` if the formula is of the form $\neg \circ \phi$ or `False` otherwise.

Return type Boolean.

Example

```
>>> from formula import *
>>> phi = Formula({"~": {"o":"x=2"}})
>>> phi.isNegativeNext()
True
```

isProposition()

Checks if the formula is a proposition.

Returns `True` if the formula is a proposition or `False` otherwise.

Return type Boolean.

Example

```
>>> from formula import *
>>> phi = Formula({"":"x=2"})
>>> phi.isProposition()
True
```

CHAPTER

TWO

CLOSURE

This module contains the functions neccesary to generate the closure of a temporal formula

`closure.`**`getClosure`**(*formula*, *closure*)

Function that generates the closure of a temporal formula.

Parameters

- **formula** (*Formula*) – Temporal formula that we want to find the closure
- **closure** (*List*) – Empty list to store the subformulas of the closure

Example

```
>>> from closure import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> closure = []
>>> getClosure(phi,closure)
>>> for formula in closure:
...     print formula.getFormula()
...
{'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}
{'~': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}
{'o': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}
{'~': {'o': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}}
{'o': {'~': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}}
{'^': {'': 'in=true', '~': {'o': 'x=2'}}}
{'~': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}
{'': 'in=true'}
{'~': 'in=true'}
{'o': 'x=2'}
{'~': {'o': 'x=2'}}
{'o': {'~': 'x=2'}}
{'': 'x=2'}
{'~': 'x=2'}
```

Note: This function is based on the conditions shown in the section 6.1 of the thesis document.

CHAPTER

THREE

MODEL CHECKING GRAPH

This module contains the necessary functions to generate a model checking graph.

modelCheckingGraph.**deleteAtoms**(*atoms*, *index_list*)

Removes atoms from a list of atoms.

Parameters

- **atoms** (*List of lists*) – List of atoms.
- **index_list** (*List*) – Index list of the elements to be removed.

Returns List of atoms with atoms removed.

Return type List.

Example

```
>>> from closure import *
>>> from modelCheckingGraph import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> closure = []
>>> getClosure(phi,closure)
>>> atoms = getAllAtoms(closure)
>>> len(atoms)
16
>>> newAtoms = deleteAtoms(atoms,[0,2,3,4,5,6,7,8,9,10,11,12,14,15])
>>> len(newAtoms)
2
```

See Also:

closure.getClosure(), formula.Formula, getAllAtoms()

modelCheckingGraph.**getAllAtoms**(*closure*)

Returns all possible atoms of the closure.

Parameters **closure** (List of Formula) – Closure of a formula.

Returns List of all atoms of the closure.

Return type List of lists of Formula.

Example

```
>>> from closure import *
>>> from modelCheckingGraph import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> closure = []
>>> getClosure(phi,closure)
```

```
>>> atoms = getAllAtoms(closure)
>>> for index, atom in enumerate(atoms):
...     print "Atom " + str(index) + ":"
...     for formula in atom:
...             print formula.getFormula()
...
Atom 0:
{'o': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}
{'': 'in=true'}
{'o': 'x=2'}
{'': 'x=2'}
{'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}
{'~': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}
Atom 1:
{'~': {'o': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}}
{'': 'in=true'}
{'o': 'x=2'}
{'': 'x=2'}
{'o': {'~': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}}
{'~': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}
{'~': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}
...
```

See Also:

closure.getClosure(), formula.Formula

Note: This function is based on the algorithm shown in [MP95].

modelCheckingGraph.**getBasicFormulas**(*closure*)

Returns the basic formulas (i.e. propositions or formulas with ∘ as main connective) of the closure.

Parameters **closure** (List of Formula) – Closure of a formula.

Returns List of basic formulas of the closure.

Return type List of Formula.

Example

```
>>> from closure import *
>>> from modelCheckingGraph import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> closure = []
>>> getClosure(phi,closure)
>>> basicFormulas = getBasicFormulas(closure)
>>> for formula in basicFormulas:
...     print formula.getFormula()
...
{'o': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}
{'': 'in=true'}
{'o': 'x=2'}
{'': 'x=2'}
```

See Also:

closure.getClosure(), formula.Formula

modelCheckingGraph.**getModelCheckingAtoms**(*tcc_structure*, *atoms*)

Returns the atoms corresponding to the states of a tcc structure.

Parameters

- **tcc_structure** (*Dictionary*) – Structure representing the behaviour of a system.
- **atoms** (*List of atoms*) – List of all possible atoms of closure.

Returns Dictionary that have the states of a tcc structure as keys, and a list of consistent atoms as values.

Return type Dictionary

Example

```
>>> from modelCheckingGraph import *
>>> from closure import *
>>> tcc_structure = {
... 1: {"store": [Formula({"":"in=true"})], "normal": [], "temporal": ["t4","p9"], "edges": [2,3
... 2: {"store": [Formula({"": "x=2"}),Formula({"": "in=true"})], "normal": [], "temporal": ["t4
... 3: {"store": [Formula({"": "x=2"}),Formula({"~": "in=true"})], "normal": ["now2"], "temporal
... 4: {"store": [Formula({"~": "in=true"})], "normal": ["now2"], "temporal": ["t7","p9"], "edge
... 5: {"store": [Formula({"": "x=1"}),Formula({"": "in=true"})], "normal": [], "temporal": ["t4
... 6: {"store": [Formula({"": "x=1"}),Formula({"~": "in=true"})], "normal": ["now2"], "temporal
... }
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> closure = []
>>> getClosure(phi,closure)
>>> atoms = getAllAtoms(closure)
>>> model_checking_atoms = getModelCheckingAtoms(tcc_structure,atoms)
>>> for tcc_node in model_checking_atoms.keys():
...     print "tcc State", tcc_node
...     tcc_atoms = model_checking_atoms.get(tcc_node)
...     for atom_index in tcc_atoms.keys():
...             print "Atom ", atom_index
...             for formula in tcc_atoms.get(atom_index):
...                     print formula.getFormula(), " | ",
...             print "\n"
tcc State 1
Atom  1
{'o': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}  |  {'': 'in=true'}  |  {'o': 'x=2'}  |
Atom  2
{'~': {'o': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}}  |  {'': 'in=true'}  |  {'o': 'x
```

See Also:

`closure.getClosure()`, `formula.Formula`, `getAllAtoms()`

`modelCheckingGraph.`**`getModelCheckingGraph`**(*tcc_structure*, *model_checking_atoms*)

Returns the model checking graph

Parameters

- **tcc_structure** (*Dictionary*) – Estructure representing the behavior of a system.
- **model_checking_atoms** (*Dictionary*) – Atoms of a tcc structure.

Returns Structure representing the model checking graph.

Return type Dictionary

Example

```
>>> from modelCheckingGraph import *
>>> from closure import *
>>> tcc_structure = {
```

```
...  1: {"store": [Formula({"":"in=true"})], "normal": [], "temporal": ["t4","p9"], "edges": [2,
... 2: {"store": [Formula({"": "x=2"}),Formula({"": "in=true"})], "normal": [], "temporal": ["t4
... 3: {"store": [Formula({"": "x=2"}),Formula({"~": "in=true"})], "normal": ["now2"], "temporal
... 4: {"store": [Formula({"~": "in=true"})], "normal": ["now2"], "temporal": ["t7","p9"], "edge
... 5: {"store": [Formula({"": "x=1"}),Formula({"": "in=true"})], "normal": [], "temporal": ["t4
... 6: {"store": [Formula({"": "x=1"}),Formula({"~": "in=true"})], "normal": ["now2"], "temporal
... }
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> closure = []
>>> getClosure(phi,closure)
>>> atoms = getAllAtoms(closure)
>>> model_checking_atoms = getModelCheckingAtoms(tcc_structure,atoms)
>>> getModelCheckingGraph(tcc_structure, model_checking_atoms)
{1: [9, 11, 12, 13, 15], 2: [10, 16, 14], 3: [], 4: [], 5: [9, 11, 12, 13, 15], 6: [10, 16, 14],
```

See Also:

`closure.getClosure()`, `formula.Formula`, `getAllAtoms()`

`modelCheckingGraph.`**`getNoBasicFormulas`**(*closure*)

Returns the formulas of the closure that are not basic formulas.

Parameters closure (List of `Formula`) – Closure of a formula.

Returns List of formulas of the closure that are not basic formulas.

Return type List of `Formula`.

Example

```
>>> from closure import *
>>> from modelCheckingGraph import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> closure = []
>>> getClosure(phi,closure)
>>> noBasicFormulas = getNoBasicFormulas(closure)
>>> for formula in noBasicFormulas:
...     print formula.getFormula()
...
{'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}
{'^': {'': 'in=true', '~': {'o': 'x=2'}}}
```

See Also:

`closure.getClosure()`, `formula.Formula`

`modelCheckingGraph.`**`getTotalNodes`**(*graph*)

Returns the total number of atoms.

Parameters graph (*Dictionary*) – Dictionary representing the atoms in each tcc state.

Returns The total number of atoms.

Return type Integer

Example

```
>>> from modelCheckingGraph import *
>>>> graph = {1: [[Formula({'': 'in=true'}), Formula({'o': 'x=2'})],
... [Formula({'~': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}})]],
... 2: [[Formula({'~': 'in=true'}),Formula({'~': {'o': 'x=2'}})]]}
>>> getTotalNodes(graph)
3
```

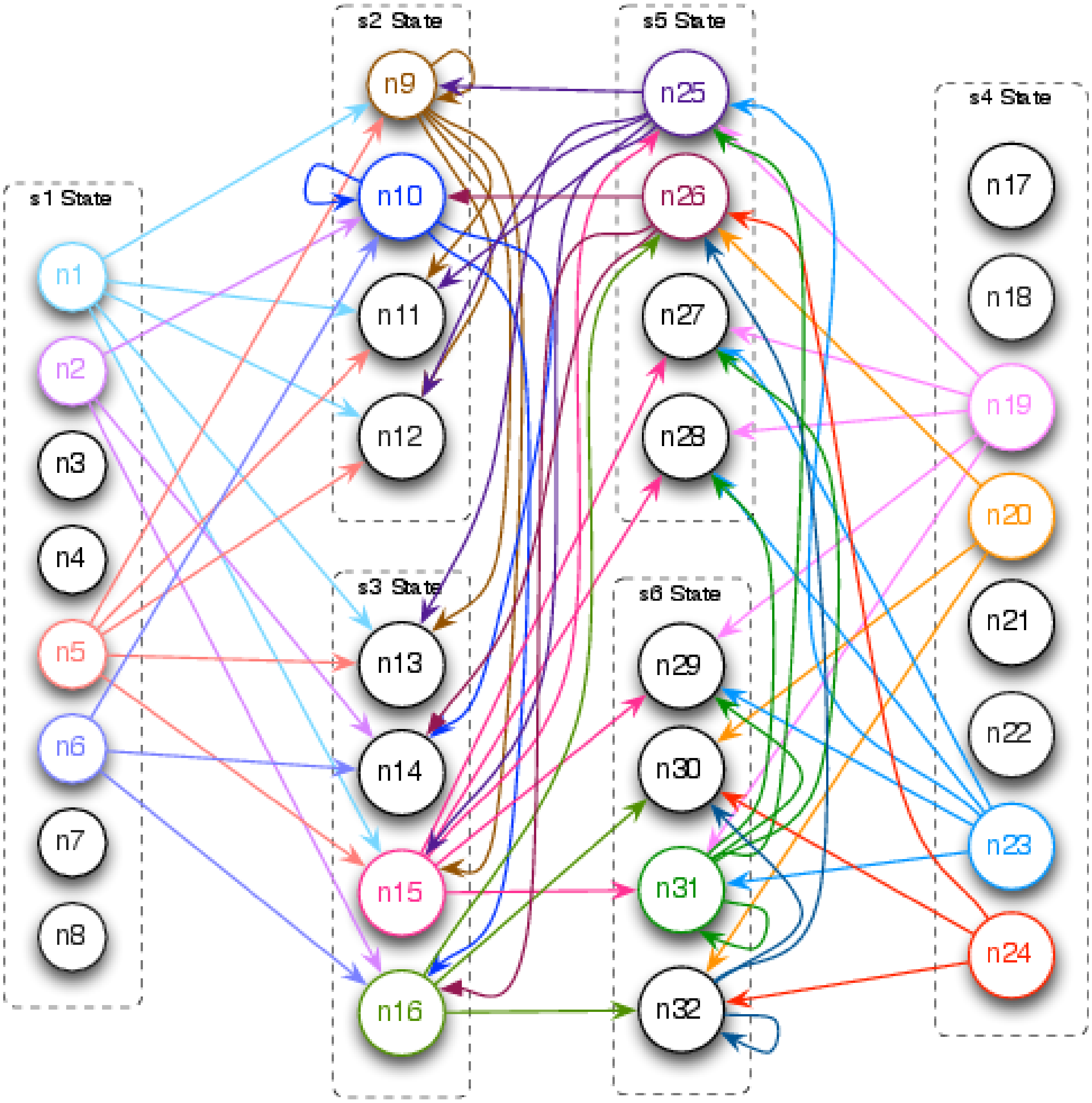

Figure 3.1: Model checking graph generated.

See Also:

`formula.Formula`

`modelCheckingGraph.`**`isConsistent`**(*formula*, *atom*)

Checks if a formula is consistent with the set of formulas in an atom.

Parameters

- **formula** (`Formula`) – Formula
- **atom** (List of `Formula`.) – List of consistent formulas representing an atom of the closure.

Returns `True` if the formula is consistent with the set of formulas in the atom or `False` otherwise.

Return type Boolean

Example

```
>>> from closure import *
>>> from modelCheckingGraph import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> closure = []
>>> getClosure(phi,closure)
>>> atoms = getAllAtoms(closure)
>>> isConsistent(Formula({'': 'x=1'}), atoms[0])
False
```

See Also:

`closure.getClosure()`, `formula.Formula`, `getAllAtoms()`

Note: This function is based on the conditions shown in the defintion 6.1 of the thesis document.

`modelCheckingGraph.`**`isInAtom`**(*formula*, *atom*)

Checks if a formula is in an atom.

Parameters

- **formula** (*Dictionary*) – Structure representing a formula.
- **atom** (List of `Formula`.) – List of consistent formulas representing an atom of the closure.

Returns `True` if the formula is in atom or `False` otherwise.

Return type Boolean

Example

```
>>> from closure import *
>>> from modelCheckingGraph import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> closure = []
>>> getClosure(phi,closure)
>>> atoms = getAllAtoms(closure)
>>> isInAtom({'': 'in=true'}, atoms[0])
True
```

See Also:

`closure.getClosure()`, `formula.Formula`, `getAllAtoms()`

`modelCheckingGraph.`**`isNextState`**(*nextFormulas*, *nextAtom*)

Checks if an atom satisfies a list of formulas with next operator as main connective.

Parameters

- **nextFormulas** (List of `Formula`) – List of formulas with next operator as main connective.
- **nextAtom** (List of `Formula`) – Atom.

Returns `True` if the atom satisfies the termporal formulas or `False` otherwise.

Return type Boolean

Example

```
>>> from modelCheckingGraph import *
>>> atom = [Formula({'o': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}),
... Formula({'': 'in=true'}), Formula({'o': 'x=2'}), Formula({'': 'x=2'}),
... Formula({'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}),
... Formula({'~': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}})]
>>>
>>> formulas = [Formula({'o': 'x=2'})]
>>> isNextState(formulas,atom)
True
```

Note: We say that an atom satisfies a list of formulas when for all the formulas $\circ\phi$ in the list we found a formula ϕ in the atom.

See Also:

`formula.Formula`

`modelCheckingGraph.`**`list2dict`**`(`*lists*, *offset*`)`

Converts a list to a dictionary with ascending numbers as keys.

Parameters

- **lists** (*List*) – List with elements.
- **offset** (*Integer*) – Offset of numeration.

Returns Dictionary with numbers as keys, and elements of the list as values.

Return type Dictionary.

Example

```
>>> from modelCheckingGraph import *
>>> list = ["I", "Love", "Computer", "Science"]
>>> list2dict(list, 2)
{2: 'I', 3: 'Love', 4: 'Computer', 5: 'Science'}
```

`modelCheckingGraph.`**`propositionConsistent`**`(`*formula*, *atom*`)`

Checks if a proposition is consistent with the formulas of an atom.

Parameters

- **formula** (`Formula`) – Formula
- **atom** (List of `Formula`) – Atom

Returns `True` if the proposition is consistent with the atom or `False` otherwise.

Return type Boolean.

Example

```
>>> from closure import *
>>> from modelCheckingGraph import *
>>> phi = Formula({"<>": {"^":{"":"in=true","~":{"o":"x=2"}}}})
>>> closure = []
>>> getClosure(phi,closure)
>>> atoms = getAllAtoms(closure)
>>> proposition = Formula({"~":"x=2"})
>>> atom = atoms[0]
>>> for formula in atom:
...     print formula.getFormula()
...
{'o': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}}
{'': 'in=true'}
{'o': 'x=2'}
{'': 'x=2'}
{'<>': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}
{'~': {'^': {'': 'in=true', '~': {'o': 'x=2'}}}}
>>> propositionConsistent(proposition, atom)
False
```

See Also:

`closure.getClosure()`, `formula.Formula`, `getAllAtoms()`

`modelCheckingGraph.`**`searchFormulas`**(*formulas*, *connective*)

Returns the formulas that have a particular main connective.

Parameters

- **formulas** (List of `Formula`) – List of formulas.
- **connective** (*String*) – The main connective.

Returns List containing the formulas that have the main connective.

Return type List

Example

```
>>> from modelCheckingGraph import *
>>> list = [Formula({'o': 'x=2'}), Formula({'~': {'o': 'x=2'}}), Formula({'o': {'~': 'x=2'}})]
>>> result = searchFormulas(list,'o')
>>> for formula in result:
...     print formula.getFormula()
...
{'o': 'x=2'}
{'o': {'~': 'x=2'}}
```

CHAPTER

FOUR

SEARCHING ALGORITHM

This module contains the necessary functions to check if a model checking graph satisfies a property.

searchingAlgorithm.**getFormulas**(*node*, *model_checking_atoms*)
Returns the formulas of a specific model checking node.

Parameters

- **node** (*Integer*) – Number of the model checking node.
- **model_checking_atoms** (*List of atoms.*) – Model checking atoms.

Returns List of formulas of the node.

Return type List of Formula.

Example

```
>>> from searchingAlgorithm import *
>>> formulas = getFormulas(3, model_checking_atoms)
>>> for formula in basicFormulas:
...     print formula.getFormula()
{'o': {'<>': {'^': {'': 'in=true', '~': {'o': 'x=1'}}}}}
{'': 'in=true'}
{'~': {'o': 'x=1'}}
{'': 'x=1'}
{'o': {'~': 'x=1'}}
{'<>': {'^': {'': 'in=true', '~': {'o': 'x=1'}}}}
{'^': {'': 'in=true', '~': {'o': 'x=1'}}}
```

See Also:

modelCheckingGraph.getModelCheckingAtoms()

searchingAlgorithm.**getInitialNodes**(*tcc_structure*, *model_checking_atoms*)
Returns the initial nodes of a model checking graph.

Parameters

- **tcc_structure** (*Dictionary*) – tcc structure.
- **model_checking_atoms** (*Dictionary.*) – Model checking atoms.

Returns A list with the number of the nodes that are initial nodes.

Return type List of Integers

Example

```
>>> from searchingAlgorithm import *
>>> getInitialNodes(tcc_structure,model_checking_atoms)
[1, 2, 3, 4, 5, 6, 7, 8, 17, 18, 19, 20, 21, 22, 23, 24]
```

See Also:

`modelCheckingGraph.getModelCheckingAtoms()`

`searchingAlgorithm.`**`getModelCheckingSCCSubgraphs`**`(`*scc_list*, *tcc_structure*, *model_checking_atoms*, *model_checking_graph*`)`

Returns the Strongly Connected Component (SCC) subgraphs of a model checking graph.

Parameters

- **scc_list** (*List of Lists*) – List of the nodes corresponding to all of SCCs in the model checking graph.
- **tcc_structure** (*Dictionary*) – tcc structure that represents the behavior of the system.
- **model_checking_atoms** (*List of atoms*) – Model checking atoms.
- **model_checking_graph** (*Dictionary*) – Model Checking graph

Returns A list with the SCC subgraphs.

Return type List

Example

```
>>> from searchingAlgorithm import *
>>> from tarjan import tarjan
>>> strongly_connected_components = tarjan(model_checking_graph)
>>> getModelCheckingSCCSubgraphs(strongly_connected_components, tcc_structure, model_checking_at
[{3: [11, 13], 7: [11, 13], 11: [11, 13], 13: [27, 29], 17: [27, 29], 21: [27, 29], 27: [11, 13]
```

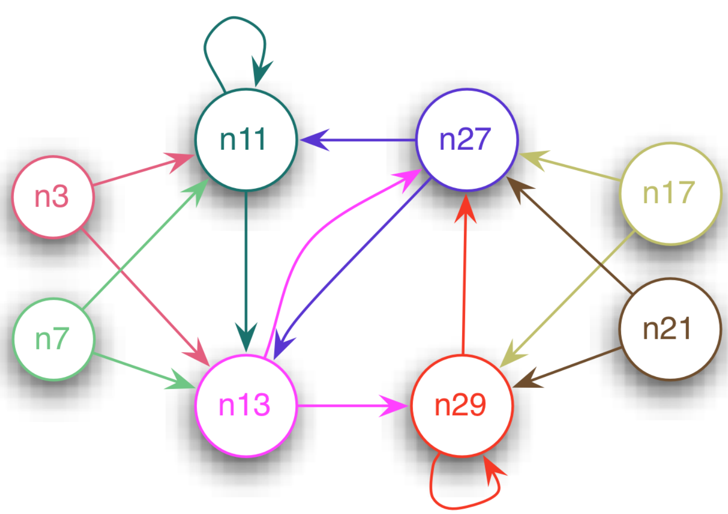

Figure 4.1: SCC subgraph generated.

Note: To generate all the SCCs of a graph we use the Tarjan's Algorithm.

See Also:

`modelCheckingGraph.getModelCheckingAtoms()`, `modelCheckingGraph.getModelCheckingGraph()`

`searchingAlgorithm.`**`initialNodesEntailFormula`**(*scc_graph*, *initial_nodes*, *model_checking_atoms*, *formula*)

Checks if the initial nodes of a model checking graph satisfy a temporal formula.

Parameters

- **scc_graph** –
- **initial_nodes** –
- **model_checking_atoms** –
- **formula** –

Returns

Return type Boolean.

Example

`searchingAlgorithm.`**`isSatisfied`**(*formula*, *tcc_structure*, *model_checking_atoms*, *model_checking_scc_subgraphs*)

`searchingAlgorithm.`**`isSelfFulfilling`**(*scc_graph*, *initial_nodes*, *model_checking_atoms*)

Checks if a SCC graph is a self-fulfilling SCC graph.

Parameters

- **scc_graph** (*Dictionary*) – SCC graph
- **initial_nodes** (*List*) – List of initial nodes of the model checking graph.
- **model_checking_atoms** (*List of atoms.*) – Model checking atoms

Returns `True` if the graph is a self-fulfilling SCC or `False` otherwise.

Return type Boolean

Example

```
>>> from searchingAlgorithm import *
>>> sccGraph = {3: [11, 13], 7: [11, 13], 11: [11, 13], 13: [27, 29], 17: [27, 29], 21: [27, 29]
>>> initialNodes = [1, 2, 3, 4, 5, 6, 7, 8, 17, 18, 19, 20, 21, 22, 23, 24]
>>> isSelfFulfilling(sccGraph, initialNodes, model_checking_atoms)
True
```

See Also:

`modelCheckingGraph.getModelCheckingAtoms()`, `getModelCheckingSCCSubgraphs()`, `getInitialNodes()`

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

BIBLIOGRAPHY

[MP95] Zohar Manna and Amir Pnueli. Temporal Verification of Reactive Systems: Safety. Springer-Verlag New York, Inc., 1995.

PYTHON MODULE INDEX

INDEX