Automata Tools

Generated by Doxygen 1.7.5.1

Sun Aug 5 2012 19:37:28

Contents

Chapter 1

Automata Tools

This library implements finite state automata (deterministic and non-deterministic), regular expressions and regular grammar related functions. The library is developed as part of the lecture "Verlässliches Programmieren in C/C++" of the department – Angewandte Datentechnik at the Uni Paderborn.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 DynArray< T > Class Template Reference

Array of dynamic size.

`#include <RG_DynArray.h>`

Public Member Functions

- DynArray ()

  DynArray Constructor Dynamically allocates Memory and initialises the size.
- ~DynArray ()

  Destructor.
- void doubleSize ()

  Doubles the Size of the DynArray.
- void add (T element)

  adds an element to the DynArray
- T & operator[] (const unsigned int nIndex)

  overrides the [] operator
- unsigned int getLength ()

  returns the Size of the DynArray
- void printArray ()

  prints the elements of the DynArray

4.1.1 Detailed Description

template<class T>class DynArray< T >

Array of dynamic size.

This is a container template class where the different components of a Grammar are stored. It represents a dynamic array with the possibility of dynamically increasing the length. (a self implemented vector class)

4.1.2 Member Function Documentation

4.1.2.1 template<class T> void DynArray< T >::add (T *element*)

adds an element to the DynArray

Parameters

element	the element to add

4.1.2.2 template<class T > void DynArray< T >::doubleSize ( )

Doubles the Size of the DynArray.

called when trying to add an element to a full DynArray.

4.1.2.3 template<class T > unsigned int DynArray< T >::getLength ( )

returns the Size of the DynArray

Returns

Size of the DynArray

4.1.2.4 template<class T > T & DynArray< T >::operator[ ] (const unsigned int *nIndex*)

overrides the [] operator

The elements of the DynArray can then be accessed with the [] operator. This Method also checks is the index can be accepted within the size of the DynArray.

Parameters

nIndex	

Returns

4.1.2.5 template<class T > void DynArray< T >::printArray ( )

prints the elements of the DynArray

works only for Scalar types and strings for now

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

- RG_DynArray.h

4.2 FiniteStateAutomaton Class Reference

Finite State Automaton data structure.

`#include <FSA_FiniteStateAutomaton.hpp>`

Public Member Functions

- FiniteStateAutomaton ()

  Constructor for FSA.
- ~FiniteStateAutomaton ()

  Destructor for FSA.
- void addState (string p_szStateName)

  Add a new state to FSA.
- void addState (State *p_stNewState)

  Add a new state to FSA.
- void removeState (string p_szStateName)

  Remove state from FSA.
- void outputStateList ()

  Output of all states.
- void printStartState ()

  Output the names of all startStates.
- void printFinalState ()

  Output the names of all finalStates .
- bool bStateExists (string p_szName)

  Searches the state list for a state with the given name for existance.
- State * getState (string p_szName)

  Searches the state list for a state with the given name, and returns a pointer to that state.
- State * getStartState ()

  Returns the start state of this automaton.
- vector< State * > * getFinalStates ()

  Returns the final states vector.
- vector< State * > * getStateList ()

  Returns the state vector.
- void addTransition (string p_szBeginningState, string p_szEdge, string p_szFinalState)

  Add a transition to transitionlist.

- void addTransition (State *p_stBeginningState, string p_szEdge, State *p_stFinalState)

  Add a transition to transitionlist.
- void addTransition (string p_szInput)

  Add a transition to transitionlist.
- void removeTransition (string p_szBeginningState, string p_szEdge, string p_szFinalState)

  Remove a transition.
- void outputTransitionList ()

  Output of all transitions.
- vector< string > getEdgesFromTransitionList ()

  Gets all edge names from this automata's transition vector.
- vector< Transition * > * getTransitions ()

  Getter for the transitionlist.
- void read (string p_szFileName)

  Reads data for this automata from a file, and adjusts all objects (setting the lists etc).
- void write (string p_szFileName)

  Writes the data of the automata into a textfile.
- void testEdge (string p_szTestEdge)

  Prints the transition that includes the testedge.
- FiniteStateAutomaton * fsaConvertNEAtoDEA ()

  Converts this finite state automata into a new deterministic finite state automata.
- FiniteStateAutomaton * minimize ()

  Minimizes this FSA using Moore's algorithm.
- void removeEmptyEdges ()

  Tries to remove all empty transitions thate are not essential for the language.
- Grammar * convertToGrammar ()

  Converts this FSA to Regular Grammar.

4.2.1 Detailed Description

Finite State Automaton data structure.

4.2.2 Member Function Documentation

4.2.2.1 void FiniteStateAutomaton::addState (string *p_szStateName*)

Add a new state to FSA.

Parameters

p_szStateName	Name of added state.

Author

fabiani, andreasb

4.2.2.2 void FiniteStateAutomaton::addState (State * p_stNewState)

Add a new state to FSA.

Parameters

**p_stNewState*	Pointer of type state direction to the added state.

Author

fabiani, andreasb

4.2.2.3 void FiniteStateAutomaton::addTransition (string p_szBeginningState, string p_szEdge, string p_szFinalState)

Add a transition to transitionlist.

Parameters

p_szBeginningState	Name of the first state of transition,
p_szEdge	Name of the transition edge,
p_szFinalState	Name of the second state of transition.

Author

fabiani, andreasb

4.2.2.4 void FiniteStateAutomaton::addTransition (State * p_stBeginningState, string p_szEdge, State * p_stFinalState)

Add a transition to transitionlist.

Parameters

**p_stBeginningState*	Pointer of type state (begining state of transition),
p_szEdge	Name of the edge,
**p_stFinalState*	Pointer of type state (final state of transition).

Author

fabiani, andreasb

4.2.2.5 void FiniteStateAutomaton::addTransition (string *p_szInput*)

Add a transition to transitionlist.

Parameters

p_szInput	String that includes the whole transition ("beginingState edge finalState").

Author

fabiani, andreasb

4.2.2.6 bool FiniteStateAutomaton::bStateExists (string *p_szName*)

Searches the state list for a state with the given name for existance.

Parameters

p_szName	Name of the state to search for.

Returns

True if a state with the given name exists in the state list, false if not.

Author

skowelek, fabiani

4.2.2.7 Grammar * FiniteStateAutomaton::convertToGrammar ( )

Converts this FSA to Regular Grammar.

Returns

A new instance of grammar.

Author

fabiani

4.2.2.8 FiniteStateAutomaton * FiniteStateAutomaton::fsaConvertNEAtoDEA ( )

Converts this finite state automata into a new deterministic finite state automata.

There is no check if this automata is already deterministic, the conversion is done no matter what.

Returns

A pointer to the new finite state automata.

Author

skowelek

4.2.2.9 vector< string > FiniteStateAutomaton::getEdgesFromTransitionList ( )

Gets all edge names from this automata's transition vector.

Returns

A vector containing all the edge names.

Author

skowelek

4.2.2.10 vector< State * > * FiniteStateAutomaton::getFinalStates ( )

Returns the final states vector.

Additionally, it updates the vector on everytime by calling FiniteStateAutomata::removeFinalStates to remove non-final states and checks the state vector for final states that are not in the vector.

Returns

The final states vector.

Author

skowelek

4.2.2.11 State * FiniteStateAutomaton::getStartState ( )

Returns the start state of this automaton.

Returns

The start state of this automaton.

Author

skowelek, fabiani

4.2.2.12 State * FiniteStateAutomaton::getState (string *p_szName*)

Searches the state list for a state with the given name, and returns a pointer to that state.

Parameters

p_szName	Name of the state to search for.

Returns

A pointer to the state if it has been found, NULL if not.

Author

skowelek, fabiani

4.2.2.13 vector< State * > * FiniteStateAutomaton::getStateList ( )

Returns the state vector.

Returns

The state vector.

Author

skowelek, fabiani

4.2.2.14 vector< Transition * > * FiniteStateAutomaton::getTransitions ( )

Getter for the transitionlist.

Returns

vector<Transition*>* Pointer of type vector<Transition*>

Author

skowelek

4.2.2.15 FiniteStateAutomaton * FiniteStateAutomaton::minimize ( )

Minimizes this FSA using Moore's algorithm.

Returns

The minimized FSA.

Author

skowelek, fabiani

4.2.2.16 void FiniteStateAutomaton::outputStateList ( )

Output of all states.

Author

fabiani, andreasb

4.2.2.17 void FiniteStateAutomaton::outputTransitionList ( )

Output of all transitions.

Author

fabiani, andreasb

4.2.2.18 void FiniteStateAutomaton::printFinalState ( )

Output the names of all finalStates .

Author

fabiani, andreasb

4.2.2.19 void FiniteStateAutomaton::printStartState ( )

Output the names of all startStates.

Author

fabiani, andreasb

4.2.2.20 void FiniteStateAutomaton::read (string *p_szFileName*)

Reads data for this automata from a file, and adjusts all objects (setting the lists etc).

Parameters

p_szFileName	Path and name of the file.

Author

skowelek

4.2.2.21 void FiniteStateAutomaton::removeEmptyEdges ( )

Tries to remove all empty transitions thate are not essential for the language.

All empty loops are removed. When a state's only outgoing transitions are empty transitions to states that have no other incoming transitions, then these states will be merged (the source state & the target states).

Author

Daniel Dreibrodt

4.2.2.22 void FiniteStateAutomaton::removeState (string *p_szStateName*)

Remove state from FSA.

Parameters

p_szStateName	Name of the removable state.

Author

fabiani,andreasb

4.2.2.23 void FiniteStateAutomaton::removeTransition (string *p_szBeginningState,* string *p_szEdge,* string *p_szFinalState*)

Remove a transition.

Parameters

p_sz-Beginning-State	Name of the transitions beginning state,
p_szEdge	Name of the transitions edge,
p_szFinal-State	Name of the transitions final state.

Author

fabiani, andreasb

4.2.2.24 void FiniteStateAutomaton::testEdge (string *p_szTestEdge*)

Prints the transition that includes the testedge.

Parameters

p_szTest-Edge	Name of the testedge

Author

fabiani

4.2.2.25 void FiniteStateAutomaton::write (string *p_szFileName*)

Writes the data of the automata into a textfile.

Parameters

p_szFile-Name	Path and name of the file.

Author

skowelek

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

- FSA_FiniteStateAutomaton.hpp
- FSA_FiniteStateAutomaton.cpp

4.3 FSAEquivalenceChecker Class Reference

Checks whether two automata represent the same regular language.

`#include <FSA_EquivalenceChecker.hpp>`

Static Public Member Functions

- static bool checkEquivalence (FiniteStateAutomaton *fsa1, FiniteStateAutomaton *fsa2)

  Checks the equivalence of two given finite state automata.

4.3.1 Detailed Description

Checks whether two automata represent the same regular language.

Author

Daniel Dreibrodt

4.3.2 Member Function Documentation

4.3.2.1 bool FSAEquivalenceChecker::checkEquivalence (FiniteStateAutomaton * *fsa1*, FiniteStateAutomaton * *fsa2*) [static]

Checks the equivalence of two given finite state automata.

The performed algorithm is based on the JFLAP implementation found in automata.graph.DFAEqualityChecker coded by Susan H. Rodger.

Parameters

fsa1	The one automaton.
fsa2	The other automaton.

Returns

Whether fsa1 accepts the same language as fsa2.

Author

Daniel Dreibrodt

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

- FSA_EquivalenceChecker.hpp
- FSA_EquivalenceChecker.cpp

4.4 FSAtoREConverter Class Reference

Converts Finite State Automata to Regular Expressions.

`#include <FSA_FSAtoREConverter.hpp>`

Static Public Member Functions

- static RegularExpression * toRE (FiniteStateAutomaton *fsa)

  Converts a FSA to a regular expression using Brzozowski's algebraic method.

4.4.1 Detailed Description

Converts Finite State Automata to Regular Expressions.

Author

Daniel Dreibrodt

4.4.2 Member Function Documentation

4.4.2.1 RegularExpression * FSAtoREConverter::toRE (FiniteStateAutomaton * *fsa*) [static]

Converts a FSA to a regular expression using Brzozowski's algebraic method.

The method was implemented according to information found at http://cs.-stackexchange.com/questions/2016/how-to-convert-finite-automata-to-regular-expressi . This algorithm was then adapted to the existing data structures and improved.

Parameters

fsa	The FSA to convert.

Returns

A regular expression equivalent to the given automaton.

Author

Daniel Dreibrodt

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

- FSA_FSAtoREConverter.hpp
- FSA_FSAtoREConverter.cpp

4.5 Grammar Class Reference

Represents a Grammar.

`#include <RG_Grammar.h>`

Public Member Functions

- Grammar ()

  Constructor.
- ∼Grammar ()

  Destructor.
- void addProduction (Production ∗prod)

  adds a production to the Grammar's Productions container.
- void setStartSymbol (string s)

  A Setter Method for the Start Symbol.
- void addTerminal (string s)

  adds a Terminal Symbol to the Grammar's Terminals container.
- void addNonTerminal (string s)

  adds a Non-Terminal Symbol to the Grammars's Terminals container.
- void processGrammarProductions ()

  processes the Grammar Productions
- DynArray< string > getTerminals ()

  a getter Method
- DynArray< string > getNonTerminals ()

  a getter Method
- string getStartSymbol ()

  a getter Method
- DynArray< Production ∗ > getProductions ()

  a getter Method
- int getNumberOfProductions ()

  a getter method
- Production ∗ getProduction (int index)

  a getter method
- int isStartSymbol (string symbol)

  compares the startSymbol of the Grammar with the parameter symbol
- void checkIfRegular ()

  checks if the Grammar is regular or not.
- void initConvert ()

  checks if Grammar is a regular Grammar, if it is not the conversion cannot be proceeded.
- FiniteStateAutomaton ∗ convertToFSA ()

  converts a Regular Grammar to a Finite States Automaton

4.5.1 Detailed Description

Represents a Grammar.

(the fact that the Grammar is a Regular Grammar or not is checked (if wanted) after reading (or creating) with the method Grammar::checkIfRegular()

A Grammar is defined through its Terminals, Non-Terminals and the Start Symbol. For this purpose, the chosen container class is a Dynamic Array defined in DynArray.h

see checkIfRegular() for more information about a REGULAR Grammar

4.5.2 Member Function Documentation

4.5.2.1 void Grammar::addNonTerminal (string *s*)

adds a Non-Terminal Symbol to the Grammars's Terminals container.

Parameters

s	

4.5.2.2 void Grammar::addProduction (Production * *prod*)

adds a production to the Grammar's Productions container.

Parameters

prod	a pointer on a Production Object.

4.5.2.3 void Grammar::addTerminal (string *s*)

adds a Terminal Symbol to the Grammar's Terminals container.

Parameters

s	a Terminal Symbol.

4.5.2.4 void Grammar::checkIfRegular ( )

checks if the Grammar is regular or not.

the information is stored in isRegular .

we use in the project a right-linear regular Grammar.

a Grammar is said to be right-linear if every production in P is of the form:

A --> x B or

A --> x

where A and B are Non-Terminals and x is any string of Terminals or the empty string check if the First Symbol of the Substitution is a Terminal

check if the Form of the Substitution corresponds to a Regular Grammar Form

4.5.2.5 FiniteStateAutomaton * Grammar::convertToFSA ( )

converts a Regular Grammar to a Finite States Automaton

Returns

the FSA

First end of the Transition

if Production has the form: A --> <epsilon>

For every symbol in the Substitution

the current symbol is a Terminal

the current Symbol is the last one in the Substitution, and it is a Terminal

the current symbol is not the last one in the substitution

the next symbol in the Substitution is a NonTerminal

the next symbol in the Substitution is a Terminal

4.5.2.6 DynArray< string > Grammar::getNonTerminals ( )

a getter Method

Returns

all the Grammar Non-Terminals stored in a DynArray

4.5.2.7 int Grammar::getNumberOfProductions ( )

a getter method

Returns

how many Productions in the Grammar

4.5.2.8 Production * Grammar::getProduction (int *index*)

a getter method

Parameters

index	Productions are stored in a Dynamic Array, index is the index of the needed Production

Returns

4.5.2.9 DynArray< Production * > Grammar::getProductions ( )

a getter Method

Returns

all the Grammar Productions stored in a DynArray

4.5.2.10 string Grammar::getStartSymbol ( )

a getter Method

Returns

the Start Symbol of the Grammar

4.5.2.11 DynArray< string > Grammar::getTerminals ( )

a getter Method

Returns

all the Grammar Terminals stored in a DynArray

4.5.2.12 int Grammar::isStartSymbol (string *symbol*)

compares the startSymbol of the Grammar with the parameter symbol

Parameters

symbol	is one string used in the grammar

Returns

4.5.2.13 void Grammar::processGrammarProductions ( )

processes the Grammar Productions

This is the last called method in RG_Reader.read() . the read method stores a -Production at first as a whole string, and this Method decompose it into a left side and its substitution in the right side

4.5.2.14 void Grammar::setStartSymbol (string *s*)

A Setter Method for the Start Symbol.

Parameters

s	the Start Symbol.

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

- RG_Grammar.h
- RG_Grammar.cpp

4.6 Group Class Reference

Group class used for Moore's minimizing algorithm.

`#include <FSA_Group.hpp>`

Public Member Functions

- Group (FiniteStateAutomaton ∗p_fsaAutomata)

  Standard constructor.
- Group (FiniteStateAutomaton ∗p_fsaAutomata, string p_szName)

  Constructor.
- void addElementToGroup (GroupElement ∗p_geElement)

  Adds a group element to the elements vector.
- void removeElementFromGroup (GroupElement ∗p_geElement)

  Removes a group element from the elements vector.
- string getName ()

  Returns the name of this group.
- void setName (string p_szName)

  Sets the name of this group.
- bool compareElements (GroupElement ∗p_geElementA, GroupElement ∗p_geElementB)

  Compares two group elements by their target groups.
- vector< GroupElement ∗ > ∗ getElements ()

Returns the elements vector of this group.

- vector< string > * getEdges ()

  Returns the automata edges vector of this group.

4.6.1 Detailed Description

Group class used for Moore's minimizing algorithm.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Group::Group (FiniteStateAutomaton * *p_fsaAutomata*)

Standard constructor.

Gets the edges of the automaton and saves them to the automata edges vector.

Parameters

p_fsaAutomata	A reference to the automata this group belongs to.

Author

skowelek, fabiani

4.6.2.2 Group::Group (FiniteStateAutomaton * *p_fsaAutomata,* string *p_szName*)

Constructor.

Gets the edges of the automaton, saves them to the automata edges vector and sets the name of the group to the given name.

Parameters

p_fsaAutomata	A reference to the automata this group belongs to.
p_szName	Name for this group.

Author

skowelek, fabiani

4.6.3 Member Function Documentation

4.6.3.1 void Group::addElementToGroup (GroupElement * *p_geElement*)

Adds a group element to the elements vector.

Parameters

p_geElement	Group element to add to the vector.

Author

skowelek, fabiani

4.6.3.2 bool Group::compareElements (GroupElement * *p_geElementA*, GroupElement * *p_geElementB*)

Compares two group elements by their target groups.

Parameters

p_geElementA	First element of the comparison.
p_geElementB	Second element of the comparison.

Returns

True if the elements have the same target groups (in same order), false if not

Author

skowelek, fabiani

4.6.3.3 vector< string > * Group::getEdges ( )

Returns the automata edges vector of this group.

Returns

The automata edges vector of this group.

Author

skowelek, fabiani

4.6.3.4 vector< GroupElement * > * Group::getElements ( )

Returns the elements vector of this group.

Returns

The elements vector of this group.

Author

skowelek, fabiani

4.6.3.5 string Group::getName ( )

Returns the name of this group.

Returns

The name of this group.

Author

skowelek, fabiani

4.6.3.6 void Group::removeElementFromGroup (GroupElement * *p_geElement*)

Removes a group element from the elements vector.

Parameters

p_geElement	Group element to remove from the vector.

Author

skowelek, fabiani

4.6.3.7 void Group::setName (string *p_szName*)

Sets the name of this group.

Parameters

p_szName	The name of this group.

Author

skowelek, fabiani

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

- FSA_Group.hpp
- FSA_Group.cpp

4.7 GroupElement Class Reference

Class for elements stored in Group objects.

`#include <FSA_GroupElement.hpp>`

Public Member Functions

- GroupElement ()
  Standard constructor.
- GroupElement (State *p_stState)
  Creates a new group element and sets the element's state to the given state.
- void addGroupToTargetVector (string p_szGroupName)
  Adds a group (name) to the target groups vector of this element.
- void removeGroupFromTargetVector (string p_szGroupName)
  Removes a group (name) from the target groups vector of this element.
- State * getState ()
  Returns the state object of this element.
- void setState (State *p_stState)
  Sets the state object of this element to the given state.
- vector< string > * getTargetGroups ()
  Returns the target groups vector.
- void clearTargetGroups ()
  Clears the target groups vector of this element.

4.7.1 Detailed Description

Class for elements stored in Group objects.

Used in Moore's minimizing algorithm

4.7.2 Constructor & Destructor Documentation

4.7.2.1 GroupElement::GroupElement (State * *p_stState*)

Creates a new group element and sets the element's state to the given state.

Parameters

p_stState	State to use for the group element.

Author

skowelek, fabiani

4.7.3 Member Function Documentation

4.7.3.1 void GroupElement::addGroupToTargetVector (string *p_szGroupName*)

Adds a group (name) to the target groups vector of this element.

Parameters

p_szGroupName	Name of the group.

Author

skowelek, fabiani

4.7.3.2 void GroupElement::clearTargetGroups ( )

Clears the target groups vector of this element.

Author

skowelek, fabiani

4.7.3.3 State * GroupElement::getState ( )

Returns the state object of this element.

Returns

The state object of this element.

Author

skowelek, fabiani

4.7.3.4 vector< string > * GroupElement::getTargetGroups ( )

Returns the target groups vector.

Returns

The target groups vector.

Author

skowelek, fabiani

4.7.3.5 void GroupElement::removeGroupFromTargetVector (string *p_szGroupName*)

Removes a group (name) from the target groups vector of this element.

Parameters

p_szGroupName	Name of the group.

Author

skowelek, fabiani

4.7.3.6 void GroupElement::setState (State * *p_stState*)

Sets the state object of this element to the given state.

Parameters

p_stState	The state object to set the state to.

Author

skowelek, fabiani

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

- FSA_GroupElement.hpp
- FSA_GroupElement.cpp

4.8 Production Class Reference

Represents a Production as an element of a Context-Free Grammar.

`#include <RG_Production.h>`

Public Member Functions

- Production ()

  Constructor.
- ~Production ()

Destructor.

- void setSubstitution (Substitution *subs)

  A setter Method to set the Production's Substitution (the right Side).

- void setLeftSide (string s)

  A setter Method to set the Production's left Side.

- Substitution * getSubstitution ()

  A getter Method.

- void readProductionFromLine (string line, string productionArrow)

  called by the Reader

- void printProduction ()

  prints the Production

- string toString ()

  returns a string form of the Production

- string getLeftSide ()

  returns the left side of a Production

4.8.1 Detailed Description

Represents a Production as an element of a Context-Free Grammar.

When a Production is used in a context-free Grammar, it substitutes the left side with what is in the right Side. The structure of this class is based on this function.

4.8.2 Member Function Documentation

4.8.2.1 string Production::getLeftSide ( )

returns the left side of a Production

Returns

the leftSide of a Production

4.8.2.2 Substitution * Production::getSubstitution ( )

A getter Method.

Returns

a pointer on the Production's Substitution.

4.8.2.3 void Production::readProductionFromLine (string *line*, string *productionArrow*)

called by the Reader

Separates the read Production line into a left side string and a right side string.

Parameters

line	a string containing the whole Production (directly read from file).
productionArrow	the Separating Symbol between the left and right Side of a production

4.8.2.4 void Production::setLeftSide (string *s*)

A setter Method to set the Production's left Side.

Parameters

s	A Non-Terminal Symbol.

4.8.2.5 void Production::setSubstitution (Substitution * *subs*)

A setter Method to set the Production's Substitution (the right Side).

Parameters

subs	a pointer on a Substitution Object.

4.8.2.6 string Production::toString ( )

returns a string form of the Production

Returns

Production as a string

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

- RG_Production.h
- RG_Production.cpp

4.9 RegularExpression Class Reference

This class represents regular expressions.

`#include <RE_RegularExpression.hpp>`

Public Member Functions

- RegularExpression (RETreeNode *p_tR)

  Constructs a new regular expression from an existing regular expression tree.
- RegularExpression (string regex)

  Constructs a new regular expression from a string.
- void setTreeRoot (RETreeNode *p_tR)

  Changes the root node of the regular expression tree.
- RETreeNode * getTreeRoot ()

  Returns the root node of the regular expression tree.
- FiniteStateAutomaton * toFSA ()

  Converts this regular expression to a finite state automaton.
- Grammar * toRG ()

  Converts this regular expression to a regular grammar.
- string toString ()

  Returns the string representation of this regular expression.

Static Public Member Functions

- static bool isOperator (string s)

  Checks whether a given string is a valid operator in the regular expression syntax.

Static Public Attributes

- static const string re_orOp = "|"

  String representation of the boolean or operator.
- static const string re_andOp = "."

  String representation of the concatenation operator.
- static const string re_starOp = "*"

  String representation of the Kleene-star operator (preceding element occurs zero or more times)
- static const string re_lParen = "("

  String representation of the opening parenthesis.
- static const string re_rParen = ")"

  String representation of the closing parenthesis.

4.9.1 Detailed Description

This class represents regular expressions.

Regular expressions are stored in objects of this class. The underlying data structure is an expression tree. Operators are stored in nodes that have their operands as children. Literals are stored in the leafs of the expression tree.

Valid operators are:

- | - boolean or - Either the preceding or the following operand have to occur
- **.** - and/concatenation - The preceding element has to be followed by the next element.
- $*$ - star quantifier - The preceding element can occur zero or more times
- **(** - opening parenthesis - Starts a group, used to express precedence
- **)** - closing parenthesis - Closes a group, used to express precedence

Valid literals are all strings that do not contain spaces or operators.

The empty literal is represented by $<epsilon>$.

Author

Daniel Dreibrodt, Konstantin Steinmiller

See also

RETreeNode

4.9.2 Constructor & Destructor Documentation

4.9.2.1 RegularExpression::RegularExpression (RETreeNode * *p_tR*)

Constructs a new regular expression from an existing regular expression tree.

Parameters

p_tR	The regular expression tree root.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.9.2.2 RegularExpression::RegularExpression (string *regex*)

Constructs a new regular expression from a string.

Parameters

regex	The regular expression string.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.9.3 Member Function Documentation

4.9.3.1 RETreeNode * RegularExpression::getTreeRoot ( )

Returns the root node of the regular expression tree.

Returns

The root node of the regular expression tree.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.9.3.2 static bool RegularExpression::isOperator (string *s*) [inline, static]

Checks whether a given string is a valid operator in the regular expression syntax.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.9.3.3 void RegularExpression::setTreeRoot (RETreeNode * *p_tR*)

Changes the root node of the regular expression tree.

Parameters

p_tR	New root node of the regular expression tree.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.9.3.4 FiniteStateAutomaton * RegularExpression::toFSA ( )

Converts this regular expression to a finite state automaton.

The returned FSA is not guaranteed to be deterministic.

Returns

A finite state automaton representing this regular expression.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.9.3.5 Grammar * RegularExpression::toRG ( )

Converts this regular expression to a regular grammar.

First the regular expression is converted to a finite state automata. That automata is then converted to a regular grammar.

Returns

A regular grammar that is equivalent to this regular expression.

Author

Daniel Dreibrodt

4.9.3.6 string RegularExpression::toString ( )

Returns the string representation of this regular expression.

Returns

A string representing this regular expression.

Author

Daniel Dreibrodt

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

- RE_RegularExpression.hpp
- RE_RegularExpression.cpp

4.10 REReaderWriter Class Reference

Handles reading and writing of regular expressions.

`#include <RE_ReaderWriter.hpp>`

Static Public Member Functions

- static RegularExpression * read (string filename)

  Reads a file and parses the regular expression in it.
- static RegularExpression * parse (const char string[])

  Parses a string and returns the represented regular expression.
- static string writeToString (RegularExpression *re)

  Creates a string representation of the given regular expression.
- static void writeToFile (RegularExpression *re, const char *filename)

  Writes a string representation of the given regular expression into a file.

Friends

- class **RegularExpression**

4.10.1 Detailed Description

Handles reading and writing of regular expressions.

This class can read files and strings and builds RegularExpression objects corresponding to the regular expressions defined in the given files or strings.

It also can write regular expressions to strings and files.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.10.2 Member Function Documentation

4.10.2.1 RegularExpression * REReaderWriter::parse (const char *str[]*) [static]

Parses a string and returns the represented regular expression.

Author

Daniel Dreibrodt, Konstantin Steinmiller

Returns

The regular expression defined in the given string.

Parameters

str	The null-terminated string representing the regular expression.

4.10.2.2 RegularExpression * REReaderWriter::read (string *filename*) [static]

Reads a file and parses the regular expression in it.

The file must contain only a single line, which in turn contains the regular expression.

Parameters

filename	The path to the input file. The path is relative to the working directory, but can also be defined absolutely.

See also

RegularExpression

Author

Daniel Dreibrodt, Konstantin Steinmiller

Returns

The regular expression defined in the given file.

4.10.2.3 void REReaderWriter::writeToFile (RegularExpression * *re*, const char * *filename*) [static]

Writes a string representation of the given regular expression into a file.

Parameters

re	The regular expression.
filename	The path to the output file.

Author

Daniel Dreibrodt

4.10.2.4 string REReaderWriter::writeToString (RegularExpression * *re*) [static]

Creates a string representation of the given regular expression.

Parameters

re	The regular expression.

Returns

The string representation of the regular expression.

Author

Daniel Dreibrodt

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

- RE_ReaderWriter.hpp
- RE_ReaderWriter.cpp

4.11 RETreeNode Class Reference

Represents nodes in the regular expression tree.

`#include <RE_TreeNode.hpp>`

Public Member Functions

- RETreeNode (string c)

  Creates a new node in the regular expression tree.
- bool isOperator ()

  Checks whether the given node represents an operator.
- RETreeNode * getLeft ()

  Gets the left operand of this operator.
- RETreeNode * getRight ()

  Gets the right operand of this operator.
- void setLeft (RETreeNode *p_l)

  Sets the left operand of this operator.
- void setRight (RETreeNode *p_r)

  Sets the right operand of this operator.
- string getContent ()

  Gets the content of the node.
- void setContent (string c)

  Sets the content of the node.
- bool isEmpty ()

  Determines whether this tree node represents an empty regular expression.
- void simplify ()

  Removes all redundancies from the regular expression.
- RETreeNode * clone ()

  Creates a clone of this tree node.
- FiniteStateAutomaton * toFSA (int *labelNum)

  Generates a non-deterministic finite state automaton representing the regular expression tree with this node as its root.
- string toString ()

  Converts a regular expression tree to a string by performing an inorder tree walk.

4.11.1 Detailed Description

Represents nodes in the regular expression tree.

A node can either be an operator or a literal. Operator nodes link to one or two nodes which act as operands.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.11.2 Constructor & Destructor Documentation

4.11.2.1 RETreeNode::RETreeNode (string *c*)

Creates a new node in the regular expression tree.

Author

Daniel Dreibrodt, Konstantin Steinmiller

Parameters

c	The content of the node. This is either an operator or a literal.

4.11.3 Member Function Documentation

4.11.3.1 RETreeNode * RETreeNode::clone ( )

Creates a clone of this tree node.

Returns

A new tree node that represents the same regular expression tree as this node.

Author

Daniel Dreibrodt

4.11.3.2 string RETreeNode::getContent ( )

Gets the content of the node.

This is either an operator or a literal.

Author

Daniel Dreibrodt, Konstantin Steinmiller

Returns

Node content.

4.11.3.3 RETreeNode * RETreeNode::getLeft ( )

Gets the left operand of this operator.

Literals have no left operand.

Returns

Left operand.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.11.3.4 RETreeNode * RETreeNode::getRight ( )

Gets the right operand of this operator.

Literals have no right operand.

Returns

Right operand.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.11.3.5 bool RETreeNode::isEmpty ( )

Determines whether this tree node represents an empty regular expression.

Returns

Whether the regular expression represented by this node is empty.

Author

Daniel Dreibrodt

4.11.3.6 bool RETreeNode::isOperator ( )

Checks whether the given node represents an operator.

Author

Daniel Dreibrodt, Konstantin Steinmiller

Returns

Returns true only if a child node is present and the content of the node is a valid operator.

4.11.3.7 void RETreeNode::setContent (string *c*)

Sets the content of the node.

This can either be an operator or a literal. Note that you cannot change a node type by giving an operator node a literal value or vice versa.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.11.3.8 void RETreeNode::setLeft (RETreeNode * *p_l*)

Sets the left operand of this operator.

Literals can have no left operand.

Parameters

p_l	Left operand.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.11.3.9 void RETreeNode::setRight (RETreeNode * *p_r*)

Sets the right operand of this operator.

Literals and Kleene-Stars can have no right operand.

Parameters

p_r	Right operand.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.11.3.10 void RETreeNode::simplify ( )

Removes all redundancies from the regular expression.

So expressions like (<epsilon>)* or (<epsilon>|<epsilon>) will be changed to <epsilon>. Expressions like (A.<epsilon>) will be changed to A (if A is a literal, subtree equality is not yet checked). Also incomplete subtrees, like operator nodes without children, will be changed to empty nodes.

Author

Daniel Dreibrodt

4.11.3.11 FiniteStateAutomaton * RETreeNode::toFSA (int * *labelNum*)

Generates a non-deterministic finite state automaton representing the regular expression tree with this node as its root.

Parameters

labelNum	Pointer to the counter variable that is used for naming the states of the FSA.

Returns

A NDFSA representing the regular expression tree with this node as its root.

Author

Daniel Dreibrodt, Konstantin Steinmiller

4.11.3.12 string RETreeNode::toString ( )

Converts a regular expression tree to a string by performing an inorder tree walk.

Returns

The string representation of the regular expression specified by the given node.

Author

Daniel Dreibrodt, Konstantin Steinmiller

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

- RE_TreeNode.hpp
- RE_TreeNode.cpp

4.12 RGReaderWriter Class Reference

provides methods to read/write a Grammar from/to a text file

```
#include <RG_ReaderWriter.h>
```

Public Member Functions

- RGReaderWriter (string fileName)

  RGReaderWriter Constructor with the full path of the file to read from as a parameter.
- ~RGReaderWriter ()

  Destructor.
- Grammar * Read ()

  read a Grammar from a text file.
- void write (Grammar *g)

  write the Grammar g into a file
- void setFileName (string fileName)

  gives a new path of a new file to read/write from/to.

4.12.1 Detailed Description

provides methods to read/write a Grammar from/to a text file

4.12.2 Constructor & Destructor Documentation

4.12.2.1 RGReaderWriter::RGReaderWriter (string *filename*)

RGReaderWriter Constructor with the full path of the file to read from as a parameter.

Parameters

filename	full path of the file to read from.

4.12.3 Member Function Documentation

4.12.3.1 Grammar * RGReaderWriter::Read ( )

read a Grammar from a text file.

Returns

the read Grammar.

4.12.3.2 void RGReaderWriter::setFileName (string *fileName*)

gives a new path of a new file to read/write from/to.

Parameters

fileName	

4.12.3.3 void RGReaderWriter::write (Grammar * g)

write the Grammar g into a file

Before calling, set a new FileName for the reader if needed.

Parameters

g	the Grammar to write

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

- RG_ReaderWriter.h
- RG_ReaderWriter.cpp

4.13 State Class Reference

A state in a Finite State Automaton.

`#include <FSA_State.hpp>`

Public Member Functions

- State ()

  default Constructor for FSA_State.
- State (string p_szName)

  Constructor for FSA_State.
- State (string p_szName, bool p_bStartState, bool p_bFinalState)

  Constructor for FSA_State.
- void setStartState (bool p_bSetStartState)

  Sets the startState value of FSA_State.
- void setFinalState (bool p_bSetFinalState)

  Sets the finalState value of FSA_State.
- bool isStartState ()

  Gives the value of startState attribute.
- bool isFinalState ()

  Gives the value of finalState attribute.
- string output ()

  Output of one state.
- string getName ()

  gets the name of the State.
- int compare (State *state)

  compares two States.

Friends

- class **Transition**
- class **FiniteStateAutomaton**

4.13.1 Detailed Description

A state in a Finite State Automaton.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 State::State ( )

default Constructor for FSA_State.

4.13.2.2 State::State (string *p_szName*)

Constructor for FSA_State.

Parameters

p_szName	Name of state.

Returns

FSA_state.

Author

fabiani, andreasb

4.13.2.3 State::State (string *p_szName,* bool *p_bStartState,* bool *p_bFinalState*)

Constructor for FSA_State.

Parameters

p_szName	Name of state,
p_bStart-State	True if State is StartState,
p_bFinal-State	True if State is FinalState.

Returns

FSA_State.

Author

fabiani, andreasb

4.13.3 Member Function Documentation

4.13.3.1 int State::compare (State * *state*)

compares two States.

Author

Yacine Smaoui

4.13.3.2 string State::getName ( )

gets the name of the State.

Author

Yacine Smaoui

4.13.3.3 bool State::isFinalState ( )

Gives the value of finalState attribute.

Author

fabiani, andreasb

4.13.3.4 bool State::isStartState ( )

Gives the value of startState attribute.

Author

fabiani, andreasb

4.13.3.5 string State::output ( )

Output of one state.

Returns

szName Name of the state.

Author

fabiani, andreasb

4.13.3.6 void State::setFinalState (bool *p_bSetFinalState*)

Sets the finalState value of FSA_State.

Parameters

p_bSetFinal-State	Value for the finalState.

Author

fabiani, andreasb

4.13.3.7 void State::setStartState (bool *p_bSetStartState*)

Sets the startState value of FSA_State.

Parameters

p_bSetStart-State	Value for the startState.

Author

fabiani, andreasb

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

- FSA_State.hpp
- FSA_State.cpp

4.14 StateConverter Class Reference

Helper class used during conversion from NDA to DFA.

`#include <FSA_StateConverter.hpp>`

Public Member Functions

- StateConverter ()

  Standard constructor.
- StateConverter (State *p_stState)

  Constructor.
- StateConverter (State *p_stState, vector< State * > *p_vecReferencedStates)

  Constructor.
- StateConverter (string p_szStateName)

  Constructor.
- StateConverter (string p_szStateName, vector< State * > *p_vecReferencedStates)

  Constructor.
- void setReferencedStates (vector< State * > *p_vecReferencedStates)

  Sets the referenced states vector of a StateConverter to the given vector.
- void addReferencedState (State *p_szReferencedState)

  Adds a State to the referenced states vector of this StateConverter.
- void removeReferencedState (string p_szStateName)

  Removes a State of the referenced states vector of this StateConverter.
- void clearReferencedStates ()

  Clears the referenced states vector of this StateConverter.
- State * getConvertedState ()

  Returns the stConvertedState of this StateConverter.
- vector< State * > * getReferencedStates ()

  Returns the referenced states vector of this StateConverter.
- bool equalsReferencedStates (StateConverter *p_scStateConverter)

  Checks if the referenced states vector of the given StateConverter equals the one of this StateConverter.

Friends

- class **Transition**
- class **FiniteStateAutomaton**
- class **State**

4.14.1 Detailed Description

Helper class used during conversion from NDA to DFA.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 StateConverter::StateConverter (State * *p_stState*)

Constructor.

Creates a new StateConverter and sets the member stConvertedState to the given -State.

Parameters

p_stState	State to set stConvertedState to.

4.14.2.2 StateConverter::StateConverter (State * *p_stState*, vector< State * > * *p_vecReferencedStates*)

Constructor.

Creates a new StateConverter and sets the members to the given values.

Parameters

p_stState	State to set stConvertedState to.
p_vecReferencedStates	to set the referenced states vector to.

4.14.2.3 StateConverter::StateConverter (string *p_szStateName*)

Constructor.

Creates a new StateConverter and creates a new State for stConvertedState with the given name.

Parameters

p_szStateName	Name for the new stConvertedState.

4.14.2.4 StateConverter::StateConverter (string *p_szStateName*, vector< State * > * *p_vecReferencedStates*)

Constructor.

Creates a new StateConverter and sets the members to the given values.

Parameters

p_szStateName	String to use as name for the new stConvertedState.
p_vecReferencedStates	to set the referenced states vector to.

4.14.3 Member Function Documentation

4.14.3.1 void StateConverter::addReferencedState (State * *p_szReferencedState*)

Adds a State to the referenced states vector of this StateConverter.

Parameters

p_szReferencedState	State to add to the referenced states vector.

4.14.3.2 bool StateConverter::equalsReferencedStates (StateConverter * *p_scStateConverter*)

Checks if the referenced states vector of the given StateConverter equals the one of this StateConverter.

Parameters

p_scStateConverter	State to compare with.

Returns

True if the referenced states vector of both StateConverters are equal, false if not.

4.14.3.3 State * StateConverter::getConvertedState ( )

Returns the stConvertedState of this StateConverter.

Returns

The stConvertedState of this StateConverter.

4.14.3.4 vector< State * > * StateConverter::getReferencedStates ( )

Returns the referenced states vector of this StateConverter.

Returns

The referenced states vector of this StateConverter.

4.14.3.5 void StateConverter::removeReferencedState (string *p_szStateName*)

Removes a State of the referenced states vector of this StateConverter.

Parameters

p_szStateName	State to remove of the referenced states vector.

4.14.3.6 void StateConverter::setReferencedStates (vector< State * > * *p_vecReferencedStates*)

Sets the referenced states vector of a StateConverter to the given vector.

Parameters

p_vecReferencedStates	to set the referenced states vector to.

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

- FSA_StateConverter.hpp
- FSA_StateConverter.cpp

4.15 StatesPair Struct Reference

defines properties of a pair of States

`#include <FSA_TableFillingMinimizer.h>`

4.15.1 Detailed Description

defines properties of a pair of States

Author

Yacine Smaoui

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

- FSA_TableFillingMinimizer.h

4.16 Substitution Class Reference

defines the right side of a Production in a Grammar

`#include <RG_Substitution.h>`

Public Member Functions

- Substitution ()

  Contructor.
- ~Substitution ()

  Destructor.
- void decode (DynArray< string > referenceTerminals, DynArray< string > referenceNonTerminals)

  Decomposes the rawString into Terminals and Non-Terminals.
- void setRawString (string s)

  A setter method to set the RawString of the Substitution.
- string getRawString ()

  returns the Substitution as it's read from a file or given manually (a string form)
- DynArray< flaggedString > * getDecodedSubstitution ()

  returns a pointer on the decoded substitution.
- string toString ()

  converts the decoded substitution into a string (for example to be printed into a file..)
- flaggedString getSymbol (int index)

  returns the symbol index in the decoded substitution.
- int symbolIsTerminal (int index)

  checks if the symbol at the index "index" is a Terminal.
- string getSymbolstring (int index)

  returns the symbol at the index "index" in a string form.
- int getDecodedSubstitutionLength ()

  returns the length of the decodedSubstitution in terms of symbols.

4.16.1 Detailed Description

defines the right side of a Production in a Grammar

composed of: A RawString : the substitution in a string form without any editing. and a DecodedSubstitution: a DynArray containing the different Terminals and Non-Terminals composing the Substitution

4.16.2 Member Function Documentation

4.16.2.1 void Substitution::decode (DynArray< string > *referenceTerminals,* DynArray< string > *referenceNonTerminals*)

Decomposes the rawString into Terminals and Non-Terminals.

the Terminals and Non-Terminals arrays are given here as a reference to find out wich symbols in the Substitution are Terminals and wich are not

ATTENTION! : this method is to call as a last step, after reading/adding all Terminals, nonTerminals and productions of a grammar all it does is Take the right side of a production in a string form, and decompose it in a sequence of Terminals and nonTerminals.

Parameters

reference-Terminals	[in] the whole Terminal array of the Grammar
reference-Non-Terminals	[in] the whole Non-Terminal array of the Grammar

4.16.2.2 DynArray< flaggedString > * Substitution::getDecodedSubstitution ( )

returns a pointer on the decoded substitution.

Returns

4.16.2.3 int Substitution::getDecodedSubstitutionLength ( )

returns the length of the decodedSubstitution in terms of symbols.

Returns

4.16.2.4 string Substitution::getRawString ( )

returns the Substitution as it's read from a file or given manually (a string form)

Returns

rawString

4.16.2.5 flaggedString Substitution::getSymbol (int *index*)

returns the symbol index in the decoded substitution.

Parameters

index	

Returns

4.16.2.6 string Substitution::getSymbolstring (int *index*)

returns the symbol at the index "index" in a string form.

Parameters

index	

Returns

4.16.2.7 void Substitution::setRawString (string *s*)

A setter method to set the RawString of the Substitution.

Parameters

s	the Substitution in string form.

4.16.2.8 int Substitution::symbolIsTerminal (int *index*)

checks if the symbol at the index "index" is a Terminal.

Parameters

index	

Returns

4.16.2.9 string Substitution::toString ( )

converts the decoded substitution into a string (for example to be printed into a file..)

Returns

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

- RG_Substitution.h
- RG_Substitution.cpp

4.17 TableFillingFSAMinimizer Class Reference

a class providing methods to minimize an FSA with the table filling algorithm.

`#include <FSA_TableFillingMinimizer.h>`

Static Public Member Functions

- static void minimize (FiniteStateAutomaton *pAutomat)

  the main function to call to use the Table filling Algorithm for minimization steps in this implementation:

4.17.1 Detailed Description

a class providing methods to minimize an FSA with the table filling algorithm.

Author

Yacine Smaoui

4.17.2 Member Function Documentation

4.17.2.1 void TableFillingFSAMinimizer::minimize (FiniteStateAutomaton * *pAutomat*) [static]

the main function to call to use the Table filling Algorithm for minimization steps in this implementation:

1)initialization of the minimization table 2)filling the minimization table by finding the distinguishable pairs of states 3)merging the distinguishable states in one equivalent state

4) finally the automat is written in a text file to see result.

Parameters

pAutomat	the FSA to minimize

Author

Yacine Smaoui

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

- FSA_TableFillingMinimizer.h
- FSA_TableFillingMinimizer.cpp

4.18 Transition Class Reference

Transition in a Finite State Automaton.

`#include <FSA_Transition.hpp>`

Public Member Functions

- Transition (State *p_stBeginning, State *p_stFinish, string p_szEdge)

  Constructor FSA_Transition.
- string output ()

  Output of one transition.
- State * getBeginningState ()

  Getter for the BeginningState.
- State * getFinishState ()

  Getter for the FinishState.
- string getEdgeName ()

  Getter for the EdgeName.

Friends

- class **FiniteStateAutomaton**

4.18.1 Detailed Description

Transition in a Finite State Automaton.

4.18.2 Member Function Documentation

4.18.2.1 State * Transition::getBeginningState ( )

Getter for the BeginningState.

Returns

Statepointer to the BeginningState.

Author

skowelek

4.18.2.2 string Transition::getEdgeName ( )

Getter for the EdgeName.

Returns

Name of the edge.

Author

skowelek

4.18.2.3 State * Transition::getFinishState ( )

Getter for the FinishState.

Returns

Statepointer to the FinishState.

Author

skowelek

4.18.2.4 string Transition::output ( )

Output of one transition.

Returns

The transition as string.

Author

fabiani, andreasb

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

- FSA_Transition.hpp
- FSA_Transition.cpp

Chapter 5

File Documentation

5.1 FSA_EquivalenceChecker.cpp File Reference

Contains the implementation of the FSAEquivalenceChecker class.

```
#include "FSA_FiniteStateAutomaton.hpp"  #include "FSA_-
EquivalenceChecker.hpp"  #include <set>  #include <map> ×
#include <vector>
```

5.1.1 Detailed Description

Contains the implementation of the FSAEquivalenceChecker class.

5.2 FSA_EquivalenceChecker.hpp File Reference

Contains the definition of the FSAEquivalenceChecker class.

```
#include "FSA_FiniteStateAutomaton.hpp" #include <map>
```

Classes

- class FSAEquivalenceChecker

  Checks whether two automata represent the same regular language.

5.2.1 Detailed Description

Contains the definition of the FSAEquivalenceChecker class.

5.3 FSA_FiniteStateAutomaton.cpp File Reference

Contains the implementation of the FSA class.

```
#include "FSA_FiniteStateAutomaton.hpp"  #include "FSA_-
Group.hpp" #include "FSA_GroupElement.hpp" #include "RG_-
Grammar.h" #include <string> #include <cstring> #include
<iostream> #include <fstream> #include <vector> #include
<map>
```

5.3.1 Detailed Description

Contains the implementation of the FSA class.

5.4 FSA_FiniteStateAutomaton.hpp File Reference

Contains the definition of the FiniteStateAutomaton class.

```
#include <iostream> #include "FSA_State.hpp" #include "-
FSA_Transition.hpp"  #include "FSA_StateConverter.hpp" ×
#include <vector>  #include <sstream>  #include <map> ×
#include <list>
```

Classes

- class FiniteStateAutomaton

  Finite State Automaton data structure.

5.4.1 Detailed Description

Contains the definition of the FiniteStateAutomaton class.

5.5 FSA_FSAtoREConverter.cpp File Reference

Contains the implementation of the FSAtoREConverter class.

```
#include "FSA_FSAtoREConverter.hpp" #include "FSA_Transition.-
hpp" #include "FSA_State.hpp" #include <vector> #include
<map> #include <string> #include "RE_RegularExpression.-
hpp" #include "RE_TreeNode.hpp"
```

5.5.1 Detailed Description

Contains the implementation of the FSAtoREConverter class.

5.6 FSA_FSAtoREConverter.hpp File Reference

Contains the definition of the FSAtoREConverter class.

```
#include "RE_RegularExpression.hpp" #include "FSA_Finite-
StateAutomaton.hpp"
```

Classes

- class FSAtoREConverter

  Converts Finite State Automata to Regular Expressions.

5.6.1 Detailed Description

Contains the definition of the FSAtoREConverter class.

5.7 FSA_Group.cpp File Reference

Contains the implementation of the Group class.

```
#include "FSA_Group.hpp"
```

5.7.1 Detailed Description

Contains the implementation of the Group class.

5.8 FSA_Group.hpp File Reference

Contains the definition of the Group class used in Moore's minimizing algorithm.

```
#include <iostream> #include <vector> #include "FSA_-
FiniteStateAutomaton.hpp" #include "FSA_State.hpp" #include
"FSA_GroupElement.hpp"
```

Classes

- class Group

  Group class used for Moore's minimizing algorithm.

5.8.1 Detailed Description

Contains the definition of the Group class used in Moore's minimizing algorithm.

5.9 FSA_GroupElement.cpp File Reference

Contains the implementation of the GroupElement class.

`#include "FSA_GroupElement.hpp" #include <string>`

5.9.1 Detailed Description

Contains the implementation of the GroupElement class.

5.10 FSA_GroupElement.hpp File Reference

Contains the definition of the GroupElement class used in Moore's minimizing algorithm.

`#include <iostream> #include <vector> #include "FSA_State.hpp"`

Classes

- class GroupElement

  Class for elements stored in Group objects.

5.10.1 Detailed Description

Contains the definition of the GroupElement class used in Moore's minimizing algorithm.

5.11 FSA_State.cpp File Reference

Contains the implementation of the State class.

`#include "FSA_State.hpp"`

5.11.1 Detailed Description

Contains the implementation of the State class.

Author

fabiani

5.12 FSA_State.hpp File Reference

Contains the definition of the State class.

```
#include <iostream>
```

Classes

- class State

  A state in a Finite State Automaton.

5.12.1 Detailed Description

Contains the definition of the State class.

Author

fabiani

5.13 FSA_StateConverter.cpp File Reference

Contains the implementation of the StateConverter class.

```
#include "FSA_StateConverter.hpp"
```

5.13.1 Detailed Description

Contains the implementation of the StateConverter class.

Author

skowelek

5.14 FSA_StateConverter.hpp File Reference

Contains the definition of the StateConverter class.

```
#include <iostream>   #include <string>   #include "FSA-
_StateConverter.hpp"   #include "FSA_State.hpp"   #include
<vector>
```

Classes

- class StateConverter

  Helper class used during conversion from NDA to DFA.

5.14.1 Detailed Description

Contains the definition of the StateConverter class.

Author

skowelek

5.15 FSA_TableFillingMinimizer.cpp File Reference

implementation of the Table filling algorithm to minimize a finite states automaton

```
#include <cstdlib>  #include "FSA_TableFillingMinimizer.-
h"
```

5.15.1 Detailed Description

implementation of the Table filling algorithm to minimize a finite states automaton

Author

Yacine Smaoui

5.16 FSA_TableFillingMinimizer.h File Reference

implementation of the Table filling algorithm to minimize a finite states automaton

```
#include "FSA_FiniteStateAutomaton.hpp"
```

Classes

- struct StatesPair

  defines properties of a pair of States
- class TableFillingFSAMinimizer

  a class providing methods to minimize an FSA with the table filling algorithm.

5.16.1 Detailed Description

implementation of the Table filling algorithm to minimize a finite states automaton

Author

Yacine Smaoui

5.17 FSA_Transition.cpp File Reference

Contains the implementation of the Transition class.

```
#include "FSA_Transition.hpp"   #include "FSA_State.hpp" ×
#include <string> #include <cstring>
```

5.17.1 Detailed Description

Contains the implementation of the Transition class.

5.18 FSA_Transition.hpp File Reference

Contains the definition of the State class.

```
#include <iostream> #include "FSA_State.hpp"
```

Classes

- class Transition

  Transition in a Finite State Automaton.

5.18.1 Detailed Description

Contains the definition of the State class.

Author

fabiani

5.19 RE_ReaderWriter.cpp File Reference

Implementation of the reader and writer class for regular expressions.

```
#include "RE_ReaderWriter.hpp"    #include "RE_TreeNode.-
hpp" #include <cstring> #include <string> #include <iostream> ×
#include <fstream>
```

5.19.1 Detailed Description

Implementation of the reader and writer class for regular expressions.

Author

Daniel Dreibrodt, Konstantin Steinmiller

5.20 RE_ReaderWriter.hpp File Reference

Definition of the reader and writer class for regular expressions.

```
#include "RE_RegularExpression.hpp"
```

Classes

- class REReaderWriter

  Handles reading and writing of regular expressions.

5.20.1 Detailed Description

Definition of the reader and writer class for regular expressions.

Author

Daniel Dreibrodt, Konstantin Steinmiller

5.21 RE_RegularExpression.cpp File Reference

Implementation of the regular expression class.

```
#include "RE_RegularExpression.hpp"  #include "RE_Reader-
Writer.hpp" #include "FSA_FiniteStateAutomaton.hpp" #include
"RG_Grammar.h"
```

5.21.1 Detailed Description

Implementation of the regular expression class.

Author

Daniel Dreibrodt, Konstantin Steinmiller

5.22 RE_RegularExpression.hpp File Reference

Definition of the regular expression class.

```
#include <string> #include "RE_TreeNode.hpp" #include "FS-
A_FiniteStateAutomaton.hpp" #include "RG_Grammar.h"
```

Classes

- class RegularExpression

This class represents regular expressions.

5.22.1 Detailed Description

Definition of the regular expression class.

Author

Daniel Dreibrodt, Konstantin Steinmiller

5.23 RE_TreeNode.cpp File Reference

Implementation of the regular expression tree node class.

```
#include <stddef.h> #include <cstdio> #include <string>
#include <vector>  #include "RE_TreeNode.hpp"  #include
"RE_RegularExpression.hpp"     #include "FSA_FiniteState-
Automaton.hpp" #include "FSA_State.hpp"
```

5.23.1 Detailed Description

Implementation of the regular expression tree node class.

Author

Daniel Dreibrodt, Konstantin Steinmiller

5.24 RE_TreeNode.hpp File Reference

Definition of the regular expression tree node class.

```
#include <string>   #include "FSA_FiniteStateAutomaton.-
hpp"
```

Classes

- class RETreeNode

  Represents nodes in the regular expression tree.

5.24.1 Detailed Description

Definition of the regular expression tree node class.

Author

Daniel Dreibrodt, Konstantin Steinmiller

5.25 RG_DynArray.h File Reference

Definition and implementation of the DynArray class.

`#include <iostream> #include <cassert>`

Classes

- class DynArray< T >

  Array of dynamic size.

5.25.1 Detailed Description

Definition and implementation of the DynArray class.

Author

Yacine Smaoui, Florian Hemmelgarn

5.26 RG_Grammar.cpp File Reference

implementation of the Grammar class

`#include "RG_Grammar.h" #include "FSA_FSAtoREConverter.hpp" #include <stdlib.h> #include <sstream>`

5.26.1 Detailed Description

implementation of the Grammar class

Author

Yacine Smaoui, Florian Hemmelgarn

5.27 RG_Grammar.h File Reference

Definition of the Grammar class.

`#include <iostream> #include "RG_DynArray.h" #include "RG_Production.h" #include "FSA_FiniteStateAutomaton.hpp" #include "RE_RegularExpression.hpp"`

Classes

- class Grammar

  Represents a Grammar.

5.27.1 Detailed Description

Definition of the Grammar class.

Author

Yacine Smaoui, Florian Hemmelgarn

5.28 RG_Production.cpp File Reference

Implementation of the Production class.

`#include "RG_Production.h" #include <sstream>`

5.28.1 Detailed Description

Implementation of the Production class.

Author

Yacine Smaoui, Florian Hemmelgarn

5.29 RG_Production.h File Reference

Definition of the Production class.

`#include <iostream> #include "RG_Substitution.h"`

Classes

- class Production

  Represents a Production as an element of a Context-Free Grammar.

5.29.1 Detailed Description

Definition of the Production class.

Author

Yacine Smaoui, Florian Hemmelgarn

5.30 RG_ReaderWriter.cpp File Reference

Implementation of the Reader class.

```
#include "RG_ReaderWriter.h"  #include "RG_Production.h" ×
#include "RG_Substitution.h"  #include <fstream>  #include
<sstream> #include <iostream> #include <stdlib.h>
```

5.30.1 Detailed Description

Implementation of the Reader class.

Author

Yacine Smaoui, Florian Hemmelgarn

5.31 RG_ReaderWriter.h File Reference

Definition of the Reader class. Allows to read from a file.

```
#include "RG_Grammar.h"
```

Classes

- class RGReaderWriter

  provides methods to read/write a Grammar from/to a text file

5.31.1 Detailed Description

Definition of the Reader class. Allows to read from a file.

Author

Yacine Smaoui, Florian Hemmelgarn

5.32 RG_Substitution.cpp File Reference

Implementation of the Substitution class.

```
#include <stdlib.h> #include <iostream> #include <sstream> ×
#include "RG_Substitution.h"
```

5.32.1 Detailed Description

Implementation of the Substitution class.

Author

Yacine Smaoui, Florian Hemmelgarn

5.33 RG_Substitution.h File Reference

Definition of the Substitution class.

`#include <string>` `#include "RG_DynArray.h"`

Classes

- class Substitution

  defines the right side of a Production in a Grammar

5.33.1 Detailed Description

Definition of the Substitution class.

Author

Yacine Smaoui, Florian Hemmelgarn