My Project

Generated by Doxygen 1.8.13

Contents

Chapter 1

Example 1

The NFA interface has three most utilized methods which we will be using:

```
void addState(std::string state_name, bool is_end=false);
void addTransition(std::string source_name, std::string destination_name, char symbol);
bool match(std::string x);
```

Building the NFA

So let's say we want to build the NFA shown in the figure.

This NFA accepts languages given by the Regular Expression `(a|b)*abb`. We could use the Regexp class but let's say we want to build it manually.

Creating the NFA

We first start by creating the NFA. All of our objects are under the namespace `Automata`. So if we want to access the NFA class we have to use `Automata::NFA`. However, for this example we simply just remove this condition by using the namespace.

```
using namespace Automata;

...

NFA nfa("0"); // Declaration of the NFA with initial state "0"
```

This creates a NFA with a state named `0` as our entry state.

Adding states

Next we have to add all the states

```
nfa.addState("1");
nfa.addState("2");
nfa.addState("3");
nfa.addState("4");
nfa.addState("5");
...
nfa.addState("10", true);
```

Adding transitions

Now we have to add the transition from state to state.

```
// add the transitions
nfa.addTransition("0", "1", nfa.epsilon);
nfa.addTransition("0", "7", nfa.epsilon);

nfa.addTransition("1", "2", nfa.epsilon);
nfa.addTransition("1", "4", nfa.epsilon);

nfa.addTransition("2", "3", 'a');

nfa.addTransition("3", "6", nfa.epsilon);

...
```

The epsilon character is accessed via `nfa.epsilon` or `NFA::epsilon`. It is of type `char`.

Note: If we try to add a state with a non-existing name then it will throw an exception.

Matching a string

Finally we want to match a string against the NFA. For this we employ the use of our match method.

```
cout << "Match? " << nfa.match("abbababaabb") << endl;
```

With output:

```
./NFA
Match? 1
```

Chapter 2

What is it?

This is a Regex engine that matches a string like `abbccccaa` to a pattern like `a+b*(cc)*aa`. It does this by building a Non-Deterministic Finite Automata out of the given regular expression and recursively emulating the automata.

One will find that the code has been abstracted in such a way that it will be easy to understand. Albeit, there is still much to do.

Basic usage

```
#include <iostream>
#include "src/Regex/Regex.h"

using namespace std;

int main() {

    Regex::Regex regex("aaa(a|b)+c*");
    cout << "Match? " << regex.match("aaabbababacc") << endl;

    return 0;
}
```

Output:

```
~ $ Match?  1
```

How-to use

Please see the Examples/ directory for examples on how to use the code. Each subfolder will have an explanation.

Building

To build the project you just have to clone the repo using

`git clone https://github.com/carlosb/regexp-to-automata`

Navigate to the directory where you cloned:

`cd dir_where_you_cloned`

Type in:

```
cmake .
cd cmake-build-debug
make
```

And to execute:

`./MyRegex`

Notes on Building

This project is coded using C++11 syntax. This should be set in CMakeLists.txt if using CMake with the flag

```
set(CMAKE_CXX_STANDARD 11)
```

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

Chapter 6

Class Documentation

6.1 hashtable< Key, T, Hasher, KeyEqual >::const_iterator Class Reference

Public Types

- typedef const_iterator **self_type**
- typedef int **difference_type**
- typedef hash_entry_type & **reference**
- typedef hash_entry_type **value_type**
- typedef hash_entry_type * **pointer**

Public Member Functions

- **const_iterator** (hashtable const &table, size_t index, const_bucket_iterator const &entry)
- self_type **operator++** ()
- self_type **operator++** (int dummy)
- const reference **operator*** () const
- const_bucket_iterator **operator->** () const
- bool **operator==** (const self_type &rhs)
- bool **operator!=** (const self_type &rhs)

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

- src/Hashtable/Hashtable.h

6.2 Set< Key, Hasher, KeyEqual >::const_iterator Class Reference

Public Types

- typedef const_iterator **self_type**
- typedef int **difference_type**
- typedef Key & **reference**
- typedef Key **value_type**
- typedef Key * **pointer**

Public Member Functions

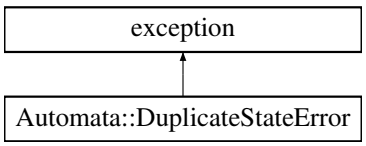
- **const_iterator** (const_hashtable_iterator const &hashtable_it)
- self_type **operator**++ ()
- self_type **operator**++ (int dummy)
- const value_type & **operator**∗ ()
- const pointer **operator->** ()
- bool **operator**== (const self_type &rhs)
- bool **operator!**= (const self_type &rhs)

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

- src/Set/Set.h

6.3 Automata::DuplicateStateError Class Reference

Inheritance diagram for Automata::DuplicateStateError:



Public Member Functions

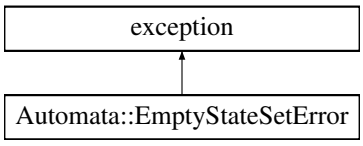
- **DuplicateStateError** (std::string name)
- virtual const char ∗ **what** () const throw ()

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

- src/Automata/AutomataErrors.h

6.4 Automata::EmptyStateSetError Class Reference

Inheritance diagram for Automata::EmptyStateSetError:



Public Member Functions

- virtual const char * **what** () const throw ()

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

- src/Automata/AutomataErrors.h

6.5 Automata::Transition::Hasher Struct Reference

Hasher to hash a transition.

`#include <Transition.h>`

Public Member Functions

- size_t **operator()** (Transition transition) const

Public Attributes

- std::hash< std::string > **h**

6.5.1 Detailed Description

Hasher to hash a transition.

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

- src/Automata/Transition.h

6.6 Automata::State::Hasher Class Reference

Functor which returns a `size_t` hash for the state.

`#include <State.h>`

Public Member Functions

- size_t **operator()** (State state) const

Public Attributes

- std::hash< std::string > **h**

6.6.1 Detailed Description

Functor which returns a `size_t` hash for the state.

This class serves as a way of hashing the state. It generates a hash of type `size_t` which a set or hashtable can then use.

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

- src/Automata/State.h

6.7 hashtable< Key, T, Hasher, KeyEqual > Class Template Reference

`#include <Hashtable.h>`

Classes

- class const_iterator
- class iterator

Public Types

- typedef hashtable< Key, T, Hasher, KeyEqual > **self_type**
- typedef std::pair< Key, T > **hash_entry_type**
- typedef std::list< hash_entry_type > **bucket_type**
- typedef std::vector< bucket_type > **index_table**
- typedef index_table::iterator **index_iterator**
- typedef bucket_type::iterator **bucket_iterator**
- typedef bucket_type::const_iterator **const_bucket_iterator**

Public Member Functions

- hashtable (const size_t buckets)

  Constructs a hashtable with specified number of buckets.
- iterator insert (Key const &key, T const &obj)

  Inserts an element using its key into hashtable.
- iterator find (Key const &key)

  Finds an element in the hashtable.
- const_iterator find (Key const &key) const

  Finds an element in the hashtable.
- void erase (Key const &key)

  Erases an object by its key.
- bool contains_key (Key const &key)

  Returns true if the key is found in the hashtable.
- size_t bucket_count () const

  Returns the number of buckets in the hashtable.
- iterator begin ()

Returns an iterator pointing to the first element of the table.

- iterator end ()

  Returns an iterator pointing to the end of the hashtable.
- const_iterator cbegin () const

  Returns a const iterator the beginning of hashtable. See `begin()`.
- const_iterator cend () const

  Returns a const iterator to the end of hashtable. See `end()`.
- bucket_type & at (size_t index)

  Returns the bucket at the specified index.
- const bucket_type & at (size_t index) const

  Returns a const reference to the specified bucket at index.
- T & at (Key const &key)

  Returns a reference to the object with the specified key.
- const T & at (Key const &key) const

  Returns a const reference to the object with the specified key.
- T & operator[] (Key const &key)

  Returns a reference to the object with the specified key.
- bucket_type operator[] (size_t index)

  Returns the bucket at the specified index.
- const T & operator[] (Key const &key) const

  Returns a const reference to the object with the specified key.
- const bucket_type & operator[] (size_t index) const

  Returns a const reference to the specified bucket at index.
- size_t count ()

  Returns the object count in the hashtable.
- size_t count () const

  Returns the object count in the hashtable.
- double load_factor ()

  Returns the load factor of the hashtable.
- self_type & operator= (self_type const &rhs)

  Overload of the operator =
- **hashtable** (const hashtable &table)

Friends

- class **iterator**
- class **const_iterator**

6.7.1 Detailed Description

template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
class hashtable< Key, T, Hasher, KeyEqual >

Description

This is a class implementing a hashtable à la STL.

We have to note that the default value for the Hasher is to utilize `std::hash` which overloads `operator()` and has a result of type `size_t`

For insertion of a hash_entry of type `T`, we expect that the object class for `T` has an appropiate and well behaved copy constructor.

6.7.2 Member Function Documentation

6.7.2.1 at() [1/4]

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
bucket_type& hashtable< Key, T, Hasher, KeyEqual >::at (
            size_t index )  [inline]
```

Returns the bucket at the specified index.

Parameters

index	Index of bucket

Returns

Bucket at specified index

6.7.2.2 at() [2/4]

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
const bucket_type& hashtable< Key, T, Hasher, KeyEqual >::at (
            size_t index ) const  [inline]
```

Returns a const reference to the specified bucket at index.

Parameters

index	Index of bucket

Returns

Const reference to bucket

6.7.2.3 at() [3/4]

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
T& hashtable< Key, T, Hasher, KeyEqual >::at (
            Key const & key )  [inline]
```

Returns a reference to the object with the specified key.

Parameters

key	Key of object

Returns

Reference to hashtable object

6.7.2.4 at() [4/4]

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
const T& hashtable< Key, T, Hasher, KeyEqual >::at (
            Key const & key ) const  [inline]
```

Returns a const reference to the object with the specified key.

Parameters

key	Key of object

Returns

Const reference to object

6.7.2.5 begin()

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
iterator hashtable< Key, T, Hasher, KeyEqual >::begin ( )  [inline]
```

Returns an iterator pointing to the first element of the table.

If no elements are in the hashtable, then it will return `end()`

Returns

`iterator` pointing to the first element

6.7.2.6 bucket_count()

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
size_t hashtable< Key, T, Hasher, KeyEqual >::bucket_count ( ) const  [inline]
```

Returns the number of buckets in the hashtable.

Returns

Number of buckets

6.7.2.7 cbegin()

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
const_iterator hashtable< Key, T, Hasher, KeyEqual >::cbegin ( ) const  [inline]
```

Returns a const iterator the beginning of hashtable. See `begin()`.

Returns

Const iterator pointing to the beginning of hashtable

6.7.2.8 cend()

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
const_iterator hashtable< Key, T, Hasher, KeyEqual >::cend ( ) const  [inline]
```

Returns a const iterator to the end of hashtable. See `end()`.

Returns

Const iterator pointing to the end of hashtable.

6.7.2.9 contains_key()

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
bool hashtable< Key, T, Hasher, KeyEqual >::contains_key (
            Key const & key )  [inline]
```

Returns true if the key is found in the hashtable.

This method looks to see if a specified key is in the hashtable. If the key is found, then return true. If the key isn't found, then return false.

Parameters

key	Key of object

Returns

`true` if key is found, `false` otherwise

6.7.2.10 count() [1/2]

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
size_t hashtable< Key, T, Hasher, KeyEqual >::count ( )  [inline]
```

Returns the object count in the hashtable.

Returns

Count of objects in hashtable

6.7.2.11 count() [2/2]

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
size_t hashtable< Key, T, Hasher, KeyEqual >::count ( ) const  [inline]
```

Returns the object count in the hashtable.

Returns

Count of objects in hashtable

6.7.2.12 end()

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
iterator hashtable< Key, T, Hasher, KeyEqual >::end ( )  [inline]
```

Returns an iterator pointing to the end of the hashtable.

The end of the hashtable is defined as being one position after the last element.

Returns

End of hashtable

6.7.2.13 erase()

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
void hashtable< Key, T, Hasher, KeyEqual >::erase (
            Key const & key )  [inline]
```

Erases an object by its key.

It the specified key isn't found then it has no effect

Parameters

key	key of object

6.7.2.14 find() [1/2]

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
iterator hashtable< Key, T, Hasher, KeyEqual >::find (
            Key const & key )  [inline]
```

Finds an element in the hashtable.

This method returns an iterator pointing to the requested object. If the specified key does not belong to an object the it will return end()

Parameters

key	key of object to find

Returns

an iterator pointing to the object, if object isn't found then return `end()`

Complexity

- Average $O(1)$
- Worst $O(n)$

6.7.2.15 find() [2/2]

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
const_iterator hashtable< Key, T, Hasher, KeyEqual >::find (
            Key const & key ) const  [inline]
```

Finds an element in the hashtable.

This is a const overload of the function `find(Key key)`. It returns a const iterator pointing to the requested object. In the given case the specified key does not belong to an object, then it will return `cend()`.

Parameters

key	key of object

Returns

a const iterator pointing to the object, if object isn't found then return cend()

- Average $O(1)$
- Worst $O(n)$

6.7.2.16 insert()

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
iterator hashtable< Key, T, Hasher, KeyEqual >::insert (
            Key const & key,
            T const & obj )  [inline]
```

Inserts an element using its key into hashtable.

Parameters

key	key of object
obj	object

Returns

iterator pointing to the inserted object

Complexity

- Average $O(1)$
- Worst $O(n)$

6.7.2.17 load_factor()

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
double hashtable< Key, T, Hasher, KeyEqual >::load_factor ( )  [inline]
```

Returns the load factor of the hashtable.

This value is calculated as follows: n/k

Where: n is the number of objects in the hashtable k is the number of buckets in the hashtable

Returns

Load factor of hashtable

6.7.2.18 operator=()

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
self_type& hashtable< Key, T, Hasher, KeyEqual >::operator= (
            self_type const & rhs )  [inline]
```

Overload of the operator =

This operator overwrites the left hand side value completely.

Parameters

rhs	Hashtable to be copied

Returns

Reference to new object

6.7.2.19 operator[]() [1/4]

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
T& hashtable< Key, T, Hasher, KeyEqual >::operator[] (
            Key const & key )  [inline]
```

Returns a reference to the object with the specified key.

Parameters

key	Key of object

Returns

Reference to object

6.7.2.20 operator[]() [2/4]

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
bucket_type hashtable< Key, T, Hasher, KeyEqual >::operator[] (
            size_t index )  [inline]
```

Returns the bucket at the specified index.

Parameters

index	Index of bucket

Returns

Bucket at specified index

6.7.2.21 operator[]() [3/4]

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
const T& hashtable< Key, T, Hasher, KeyEqual >::operator[] (
            Key const & key ) const  [inline]
```

Returns a const reference to the object with the specified key.

Parameters

key	Key of object

Returns

Const reference to object

6.7.2.22 operator[]() [4/4]

```
template<class Key, class T, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<↩
Key>>
const bucket_type& hashtable< Key, T, Hasher, KeyEqual >::operator[] (
            size_t index ) const  [inline]
```

Returns a const reference to the specified bucket at index.

Parameters

index	Index of bucket

Returns

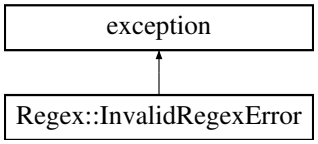
Bucket at specified index

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

- src/Hashtable/Hashtable.h

6.8 Regex::InvalidRegexError Class Reference

Inheritance diagram for Regex::InvalidRegexError:



Public Member Functions

- virtual const char * **what** () const throw ()

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

- src/Regex/RegexErrors.h

6.9 hashtable< Key, T, Hasher, KeyEqual >::iterator Class Reference

Public Types

- typedef iterator **self_type**
- typedef int **difference_type**
- typedef hash_entry_type & **reference**
- typedef hash_entry_type **value_type**
- typedef hash_entry_type * **pointer**

Public Member Functions

- **iterator** (hashtable &table, size_t index, bucket_iterator const &entry)
- self_type **operator**++ ()
- self_type **operator**++ (int dummy)
- reference **operator*** ()
- bucket_iterator **operator->** ()
- bool **operator**== (const self_type &rhs)
- bool **operator!**= (const self_type &rhs)

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

- src/Hashtable/Hashtable.h

6.10 Set< Key, Hasher, KeyEqual >::iterator Class Reference

Public Types

- typedef iterator **self_type**
- typedef int **difference_type**
- typedef Key & **reference**
- typedef Key **value_type**
- typedef Key * **pointer**

Public Member Functions

- **iterator** (hashtable_iterator const &hashtable_it)
- self_type **operator++** ()
- self_type **operator++** (int dummy)
- reference **operator*** ()
- pointer **operator->** ()
- bool **operator==** (const self_type &rhs)
- bool **operator!=** (const self_type &rhs)

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

- src/Set/Set.h

6.11 Regex::Lexer Class Reference

`#include <Lexer.h>`

Public Member Functions

- Lexer ()

  Default constructor.
- Lexer (std::string source)

  Constructor which takes a string as source.
- Token nextToken ()

  Returns the next token in the source.
- void setSource (std::string source)

  Sets the string to parse.

6.11.1 Detailed Description

Description

This lexer is meant to obtain tokens from a regular expression string. The variable symbol set is a hashtable where all the reserved symbols (like the alternation token "|") are stored.

Strictly speaking, we should also keep an "alphabet" but we just take it as everything we can type on a keyboard minus the symbol set.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Lexer() [1/2]

```
Regex::Lexer::Lexer ( )
```

Default constructor.

Will initialize the string to parse as an empty string

6.11.2.2 Lexer() [2/2]

```
Regex::Lexer::Lexer (
            std::string source )
```

Constructor which takes a string as source.

Parameters

source	String to parse

6.11.3 Member Function Documentation

6.11.3.1 nextToken()

```
Token Regex::Lexer::nextToken ( )
```

Returns the next token in the source.

Description

This method will consume the source input and return the next token determined by the given patterns in the addAutomata() method.

Returns

Next token lexed

6.11.3.2 setSource()

```
void Regex::Lexer::setSource (
            std::string source )
```

Sets the string to parse.

Parameters

source	String to parse

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

- src/Regex/Lexer.h
- src/Regex/Lexer.cpp

6.12 Automata::NFA Class Reference

This class models the behaviour of a non deterministic finite automaton.

`#include <NFA.h>`

Public Types

- typedef hashtable< std::string, State > **state_table_type**
- typedef Set< State, State::Hasher > **state_set_type**

Public Member Functions

- NFA (std::string start_state_name, size_t bucket_count=100)
  Instantiates the NFA with a starting state name.
- **NFA** (const NFA &nfa)
- ~NFA ()
  Destructor method for NFA.
- void addState (std::string state_name, bool is_end=false)
  Adds a state to the NFA.
- void addTransition (State *from, State *to, char symbol)
  Adds a state to the NFA.
- void addTransition (std::string from, std::string to, char symbol)
  Add transition from one state to other.
- NFA concatenate (NFA const &to_nfa) const
  Concatenates two automatas.
- NFA alternate (NFA const &to_nfa) const
  Alternates two automatas.
- NFA kleene () const
  Calculates the kleene closure of the automata.
- NFA kleene_plus () const
  Returns the kleene plus closure of the automata.
- NFA optional () const
  Returns an automata with an accepting language as optional.
- void setString (std::string str)
  Sets the string to parse.
- void advance ()
  Advances to the next set of states after advancing to the next character.
- state_set_type getCurrentStates ()

Gets current set of states the automaton is at.

- bool accepts ()

  Returns whether the automata is at an accepting state.
- State * getState (std::string name)

  Returns a pointer to the state identified by its name.
- State * initialState () const

  Returns a pointer to the initial state of the NFA.
- state_table_type & table ()

  Returns the table of states.
- const state_table_type & table () const

  Returns a const reference of the table of states.
- state_set_type & end_states ()

  Returns a reference to the set of end states of the automata.
- const state_set_type & end_states () const

  Returns a const reference to the set of end states of the automata.
- state_set_type epsilon_closure (State s)

  Returns the epsilon closure of state s.
- state_set_type epsilon_closure (state_set_type T)

  Returns the epsilon closure of a set of states T.
- state_set_type move (state_set_type T, char c)

  Returns a the set of states to which there is a move from T on symbol c.
- bool match (std::string x)

  Returns true if the string matches the nfa pattern.
- NFA & operator= (const NFA &rhs)

  Copies the NFA to another.
- void setInitialState (std::string name)

  Sets the initial state of the automata.

Public Attributes

- const std::string **id_string_**
- state_table_type stateTable_

  Table of states.
- state_set_type endStates_

  Set of end states.
- State * startState_

  Pointer to start state.
- std::string str_to_match

  String to match.
- state_set_type S_

  Set that contains the current states.
- std::string::iterator current_ptr_

  Iterator pointing to the current char on str_to_match.

Static Public Attributes

- static const char epsilon = '\x08'

  We take the convention that the escape character '' represents epsilon.

6.12.1 Detailed Description

This class models the behaviour of a non deterministic finite automaton.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17.

Description

This class defines the methods and attributes for a non deterministic finite automaton.

We keep a pointer to the start state as well as a table for the states. We identify a state by a name key, such as "s1" or "state3". It will be up to the user of the class to keep track of the names he/she gives to the states. This means that we have to make references to the states with their names

Please see the book: Compilers: Principles, Techniques and Tools (Aho et Al.)

6.12.2 Constructor & Destructor Documentation

6.12.2.1 NFA()

```
Automata::NFA::NFA (
            std::string start_state_name,
            size_t bucket_count = 100 )  [explicit]
```

Instantiates the NFA with a starting state name.

This constructor will instantiate the NFA given the name of the start state. It also takes the number of buckets which regulates whether the internal access to the elements will be $O(1)$ or not.

If the number of states is known beforehand, then you should set bucket count to be equal to the number of states.

Parameters

start_state_name	Name of initial state
bucket_count	Number of buckets which the state table will have

6.12.3 Member Function Documentation

6.12.3.1 accepts()

```
bool Automata::NFA::accepts ( )
```

Returns whether the automata is at an accepting state.

If any of the states in the current set of states is an end state, then the automata must be at an accepting state and therefore it accepts the string at that point.

Note: This does not mean it accepts the whole string.

6.12.3.2 addState()

```
void Automata::NFA::addState (
            std::string state_name,
            bool is_end = false )
```

Adds a state to the NFA.

Parameters

state_ name	Name of state to be added. Must be unique.
is_ end	true if state is end state, false otherwise

Implementation details

This method adds a state to the NFA by inserting it in the state table. If the state is an end state, then it adds it to the end_states set defined in the Variables section.

6.12.3.3 addTransition() [1/2]

```
void Automata::NFA::addTransition (
            State * from,
            State * to,
            char symbol )
```

Adds a state to the NFA.

Parameters

s	state to add

Implementation details

This method adds an already constructed state to the NFAAdd transition from one state to other

Parameters

from	Pointer to state from where the transition begins
to	Pointer to state where the transition ends
symbol	Character symbol required to make the transition

6.12.3.4 addTransition() [2/2]

```
void Automata::NFA::addTransition (
            std::string from,
            std::string to,
            char symbol )
```

Add transition from one state to other.

States can be identified by their name, so you can actually add transitions based on the names.

Parameters

from	Name of the state from where the transition begins
to	Name of the state to where the transition ends
symbol	Character symbol required to make the transition

6.12.3.5 advance()

```
void Automata::NFA::advance ( )
```

Advances to the next set of states after advancing to the next character.

Advances to the next character of the string that we are currently parsing. In other words, it sets the current states to the epsilon closure of the set move of the current states on the current symbol.

6.12.3.6 alternate()

```
NFA Automata::NFA::alternate (
            NFA const & to_nfa ) const
```

Alternates two automatas.

This method returns an automata that is the alternation of both given automatas.

Formally, if A and B are two NFAs which have respective languages $L(A)$ and $L(B)$ then this method returns a NFA $C = A|B$ which accepts the language $L(A)UL(B)$.

Complexity

Worst case $O(n^2 + m^2)$ Where:

- n is the number of states in A
- m is the number of states in B

Notes on the complexity

The complexity is actually really bad because we're reconstructing the whole "graph". This means we first have to add the states which takes $O(n)$. Then we have to add all transitions. In the worst case scenario we have at most kn^2 arrows coming out of each state. However, k is a constant which represents the number of symbols in the alphabet. Therefore, we have to do $O(n^2)$ operations for each graph.

Parameters

to_nfa	Right side automata. (AKA. automata B)

Returns

The alternation of two NFAs

6.12.3.7 concatenate()

```
NFA Automata::NFA::concatenate (
            NFA const & to_nfa ) const
```

Concatenates two automatas.

This method returns an automata that is the concatenation of both automatas. It merges the sets of states and creates a new set of states as well as it copies the transitions.

Formally, if A and B are two NFAs which have respective languages $L(A)$ and $L(B)$ then this method returns a NFA $C = AB$ which accepts the language $L(A)L(B)$.

Implementation Details

The automata will merge both sets of states using their respective keys. This means that if we have two states with equal names in both A and B, this will generate a collision and throw an exception.

Complexity

Worst case $O(n^2 + m^2)$ Where:

- n is the number of states in A
- m is the number of states in B

Notes on the complexity

The complexity is actually really bad because we're reconstructing the whole "graph". This means we first have to add the states which takes $O(n)$. Then we have to add all transitions. In the worst case scenario we have at most kn^2 arrows coming out of each state. However, k is a constant which represents the number of symbols in the alphabet. Therefore, we have to do $O(n^2)$ operations for each graph.

Parameters

to_nfa	Right side automata. (AKA. automata B)

Returns

The concatenation of two NFAs

6.12.3.8 end_states() [1/2]

```
NFA::state_set_type & Automata::NFA::end_states ( )
```

Returns a reference to the set of end states of the automata.

Returns

End states

6.12.3.9 end_states() [2/2]

```
const NFA::state_set_type & Automata::NFA::end_states ( ) const
```

Returns a const reference to the set of end states of the automata.

Returns

End states

6.12.3.10 epsilon_closure() [1/2]

```
NFA::state_set_type Automata::NFA::epsilon_closure (
            State s )
```

Returns the epsilon closure of state s.

Formally, this method returns the epsilon closure of state s. In other words, it returns the set of states reachable from state s on an epsilon transition (i.e. transition with epsilon as its symbol)

Parameters

s	State s

Returns

set of states reachable from state s on epsilon transition

6.12.3.11 epsilon_closure() [2/2]

```
NFA::state_set_type Automata::NFA::epsilon_closure (
            state_set_type T )
```

Returns the epsilon closure of a set of states T.

In other words, this will return the union of epsilon closures for each state s in T

Parameters

T	Set of states

Returns

union of epsilon closure for each state s in T

6.12.3.12 getCurrentStates()

```
NFA::state_set_type Automata::NFA::getCurrentStates ( )
```

Gets current set of states the automaton is at.

This method gets the current set of states the automaton is at after having advanced a character using the advance() method.

Implementation Details

If the advanced method hasn't been used yet then, formally, it will return the epsilon closure of the initial state.

Returns

the set of states that the automaton is at

6.12.3.13 getState()

```
State * Automata::NFA::getState (
            std::string name )
```

Returns a pointer to the state identified by its name.

Parameters

name	Name of state

Returns

a pointer to the state

6.12.3.14 initialState()

```
State * Automata::NFA::initialState ( ) const
```

Returns a pointer to the initial state of the NFA.

Returns

a pointer to the initial state of the NFA

6.12.3.15 kleene()

```
NFA Automata::NFA::kleene ( ) const
```

Calculates the kleene closure of the automata.

This method returns an automata which is the kleene closure of the automata itself.

Formally, if A is an automata with language $L(A)$, this method returns an automata V with language $L(V) = (L(A))*$.

Complexity

Worst case $O(n^2)$ Where:

- n is the number of states in A

Notes on the complexity

The complexity is actually really bad because we're reconstructing the whole "graph". This means we first have to add the states which takes $O(n)$. Then we have to add all transitions. In the worst case scenario we have at most kn^2 arrows coming out of each state. However, k is a constant which represents the number of symbols in the alphabet. Therefore, we have to do $O(n^2)$ operations for each graph.

References

- For more reference and a simple explanation you can consult wikipedias page on the subject at https://en.wikipedia.org/wiki/Kleene_star

Returns

Kleene star of automata

6.12.3.16 kleene_plus()

```
NFA Automata::NFA::kleene_plus ( ) const
```

Returns the kleene plus closure of the automata.

This method returns an automata which is the kleene closure of the automata itself.

Formally, if A is an automata with language $L(A)$, this method returns an automata V with language $L(V) = (L(A))+$.

Complexity

Worst case $O(n^2)$ Where:

- n is the number of states in A

Notes on the complexity

The complexity is actually really bad because we're reconstructing the whole "graph". This means we first have to add the states which takes $O(n)$. Then we have to add all transitions. In the worst case scenario we have at most kn^2 arrows coming out of each state. However, k is a constant which represents the number of symbols in the alphabet. Therefore, we have to do $O(n^2)$ operations for each graph.

References

- For more reference and a simple explanation you can consult wikipedias page on the subject at https://en.wikipedia.org/wiki/Kleene_star

Returns

Kleene plus of automata

6.12.3.17 match()

```
bool Automata::NFA::match (
            std::string x )
```

Returns true if the string matches the nfa pattern.

Note: It does not modify anything.

Parameters

x	String to match

Returns

true if string matches pattern, false otherwise

6.12.3.18 move()

```
NFA::state_set_type Automata::NFA::move (
            NFA::state_set_type T,
            char c )
```

Returns a the set of states to which there is a move from T on symbol c.

Formally, it returns a set of states to which there is a transition on symbol c from some state s in T

Parameters

T	Set of states
c	Symbol of transition

Returns

a set of states to which there is a transition on symbol c from some state s in T

6.12.3.19 optional()

```
NFA Automata::NFA::optional ( ) const
```

Returns an automata with an accepting language as optional.

Formally, if A is an automata with language $L = L(A)$ then this method returns an automata with an accepting language of $LU\{\epsilon\}$

Returns

Automata accepting either empty state or the language of the automata

6.12.3.20 setString()

```
void Automata::NFA::setString (
            std::string str )
```

Sets the string to parse.

This method sets the string to parse. We can then use advance() to advance to the next set of states the automaton is at.

Parameters

str	String to parse

6.12.3.21 table() [1/2]

```
NFA::state_table_type & Automata::NFA::table ( )
```

Returns the table of states.

This table type is of `NFA::state_table_type` so in case one would want to access the table you simply have to do:

```
NFA::state_table_type table = nfa.table()
```

Returns

the table of states

6.12.3.22 table() [2/2]

```
const NFA::state_table_type & Automata::NFA::table ( ) const
```

Returns a const reference of the table of states.

This table type is of `NFA::state_table_type` so in case one would want to access the table you simply have to do:

```
NFA::state_table_type table = nfa.ctable()
```

Returns

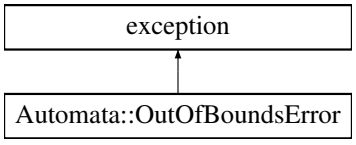
the table of states

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

- src/Automata/NFA.h
- src/Automata/NFA.cpp

6.13 Automata::OutOfBoundsError Class Reference

Inheritance diagram for Automata::OutOfBoundsError:



Public Member Functions

- **OutOfBoundsError** (std::string str)
- virtual const char * **what** () const throw ()

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

- src/Automata/AutomataErrors.h

6.14 Regex::Parser Class Reference

Public Member Functions

- Parser ()

  Constructor for the parser.
- ~Parser ()

  Destructor for the parser.
- bool parse (std::string regex)

  Method to parse the regex. Returns true if succesful, otherwise false. This method parses a regex specification string. If it returns true, then the parse completed succesfully.
- token_list tokenList ()

  Returns a list of tokens identified by the parser This method returns the list of tokens after they have been identified by the parser. These follow the order of the original string.
- Automata::NFA getBuiltNFA ()

6.14.1 Member Function Documentation

6.14.1.1 getBuiltNFA()

```
Automata::NFA Regex::Parser::getBuiltNFA ( )
```

Gets the associated NFA that accepts the same language as the regex pattern

Returns

Associated NFA

6.14.1.2 parse()

```
bool Regex::Parser::parse (
            std::string regex )
```

Method to parse the regex. Returns true if succesful, otherwise false. This method parses a regex specification string. If it returns true, then the parse completed succesfully.

Parameters

regex	Regular expression specification pattern

Returns

True if parse is succesful, false otherwise

6.14.1.3 tokenList()

```
std::vector< Token > Regex::Parser::tokenList ( )
```

Returns a list of tokens identified by the parser This method returns the list of tokens after they have been identified by the parser. These follow the order of the original string.

Returns

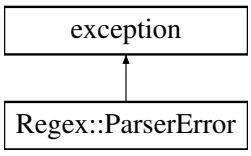
List of tokens

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

- src/Regex/Parser.h
- src/Regex/Parser.cpp

6.15 Regex::ParserError Class Reference

Inheritance diagram for Regex::ParserError:



Public Member Functions

- virtual const char * **what** () const throw ()

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

- src/Regex/RegexErrors.h

6.16 PearsonHasher8 Class Reference

An implementation of a Pearson hashtable using our template class hashtable.

`#include <PearsonHashtable8.h>`

Public Member Functions

- PearsonHasher8 ()
  Constructor for a PearsonHasher.
- virtual size_t operator() (std::string const &key) const
  Function which returns the hash using the Pearson hashing method.

Static Public Attributes

- static const size_t table_size_ = 256
  Table size.

6.16.1 Detailed Description

An implementation of a Pearson hashtable using our template class hashtable.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17.

Description

A PearsonHashtable8 is a Hashtable which uses the Pearson Hashing function to obtain the key. This particular case is an 8-bit implementation, which means we only have 256 (0-255) available keys.

Uses of this class could be for a table of reserved words and symbols of a compiler.

6.16.2 Member Function Documentation

6.16.2.1 operator()()

```
virtual size_t PearsonHasher8::operator() (
            std::string const & key ) const  [inline], [virtual]
```

Function which returns the hash using the Pearson hashing method.

Parameters

key	Key to hash

Returns

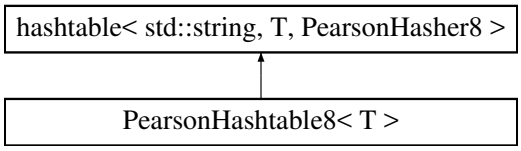
Hash of key

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

- src/Hashtable/PearsonHashtable8.h

6.17 PearsonHashtable8< T > Class Template Reference

Inheritance diagram for PearsonHashtable8< T >:



Additional Inherited Members

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

- src/Hashtable/PearsonHashtable8.h

6.18 Regex::Regex Class Reference

Public Member Functions

- **Regex** (std::string pattern)
- void **setPattern** (std::string pattern)
- bool **match** (std::string str)

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

- src/Regex/Regex.h
- src/Regex/Regex.cpp

6.19 Regex Class Reference

Regular expression engine clcass.

`#include <Regex.h>`

Public Types

- enum {
  TOK_EOF = -1, **TOK_NONE** = 0, **TOK_LPAREN**, **TOK_RPAREN**,
  TOK_KLEENE, **TOK_ALTER**, **TOK_CONCAT**, **TOK_QMARK**,
  TOK_PLUS, **TOK_CHAR**, **TOK_SPACE**, **TOK_ESCAPE_SEQUENCE** }

  This type aids in defining the ids for the Token Tags.

6.19.1 Detailed Description

Regular expression engine clcass.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

4/18/17.

Description

It parses regular expressions.

TODO

Many many things.

6.19.2 Member Enumeration Documentation

6.19.2.1 anonymous enum

`anonymous enum`

This type aids in defining the ids for the Token Tags.

Description

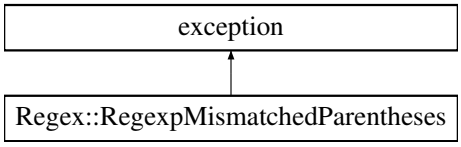
In order to get unique ids, we use an enum structure to assign each of the tokens an id.

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

- src/Regex/Regex.h
- src/Regex/TokenDecls.h

6.20 Regex::RegexpMismatchedParentheses Class Reference

Inheritance diagram for Regex::RegexpMismatchedParentheses:



Public Member Functions

- **RegexpMismatchedParentheses** (std::string s)
- virtual const char * **what** () const throw ()

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

- src/Regex/RegexErrors.h

6.21 Set< Key, Hasher, KeyEqual > Class Template Reference

`#include <Set.h>`

Classes

- class const_iterator
- class iterator

Public Member Functions

- **Set** (size_t bucket_count)
- **Set** (const Set &set)
- iterator insert (Key element)

  Inserts element into set.
- void remove (Key element)

  Removes specified element from set.
- iterator find (Key element)

  Finds element in set.
- const_iterator find (Key element) const

  Finds element in set.
- bool contains (Key element)

  Returns true if element is contained in set, false otherwise.
- self_type Union (self_type const &S)

  Returns the union of the set with another set S.
- self_type Intersection (self_type const &S)

  Returns the intersection of the set with another set S.
- self_type Difference (self_type const &right)

Returns the difference of the set with another set S.

- iterator begin ()

  Returns iterator pointing to the first element of the set.

- iterator end ()

  Returns iterator pointing to the end of the set.

- const_iterator cbegin () const

  Returns const iterator pointing to the first element of the set.

- const_iterator cend () const

  Returns const iterator pointing to the end of the set.

- size_t count () const

  Returns the number of elements in the set.

- size_t bucket_count () const

  Returns the number of buckets that the set uses.

- bool empty ()

  Returns true if the set is empty, false otherwise.

- self_type & operator= (const self_type &rhs)

  Overloaded copy operator.

Friends

- class **iterator**

6.21.1 Detailed Description

template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
class Set< Key, Hasher, KeyEqual >

Description

The class models the behaviour and notion of a set. The main methods are:

(Assuming an element access has average complexity of $O(1)$)

- Union - $O(n + m)$
- Intersection - $O(n)$
- Difference - $O(n)$

6.21.2 Member Function Documentation

6.21.2.1 begin()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
iterator Set< Key, Hasher, KeyEqual >::begin ( )  [inline]
```

Returns iterator pointing to the first element of the set.

If no elemnent is in set, then this will return `end()`.

Returns

Iterator pointing to first element in set

6.21.2.2 bucket_count()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
size_t Set< Key, Hasher, KeyEqual >::bucket_count ( ) const  [inline]
```

Returns the number of buckets that the set uses.

Returns

Number of buckets used internally by the set

6.21.2.3 cbegin()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
const_iterator Set< Key, Hasher, KeyEqual >::cbegin ( ) const  [inline]
```

Returns const iterator pointing to the first element of the set.

See `begin()` for a better understanding of what constitues the beginninng of a set.

Returns

Const iterator to the beginning of set.

6.21.2.4 cend()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
const_iterator Set< Key, Hasher, KeyEqual >::cend ( ) const  [inline]
```

Returns const iterator pointing to the end of the set.

See `end()`.

Returns

Const iterator pointing to the end of set.

6.21.2.5 contains()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
bool Set< Key, Hasher, KeyEqual >::contains (
            Key element )  [inline]
```

Returns true if element is contained in set, false otherwise.

Parameters

element	Element in question

Returns

True if element is in set, else false

6.21.2.6 count()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
size_t Set< Key, Hasher, KeyEqual >::count ( ) const  [inline]
```

Returns the number of elements in the set.

Returns

Number of elements in set

6.21.2.7 Difference()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
self_type Set< Key, Hasher, KeyEqual >::Difference (
            self_type const & right )  [inline]
```

Returns the difference of the set with another set S.

Parameters

right	the right operand of the difference

Returns

the difference of this set minus the second set

6.21.2.8 empty()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
bool Set< Key, Hasher, KeyEqual >::empty ( )  [inline]
```

Returns true if the set is empty, false otherwise.

Returns

True if the set is empty, else false

6.21.2.9 end()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
iterator Set< Key, Hasher, KeyEqual >::end ( )  [inline]
```

Returns iterator pointing to the end of the set.

Returns

Iterator pointing to the end of set

6.21.2.10 find() [1/2]

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
iterator Set< Key, Hasher, KeyEqual >::find (
            Key element )  [inline]
```

Finds element in set.

Parameters

element	Element to find

Returns

Iterator to found element

If no element is found then this will return `end()`.

6.21.2.11 find() [2/2]

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
const_iterator Set< Key, Hasher, KeyEqual >::find (
            Key element ) const  [inline]
```

Finds element in set.

Parameters

element	Element to find

Returns

Const iterator to found element

6.21.2.12 insert()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
iterator Set< Key, Hasher, KeyEqual >::insert (
            Key element )  [inline]
```

Inserts element into set.

Parameters

element	Element to be inserted

Returns

6.21.2.13 Intersection()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
self_type Set< Key, Hasher, KeyEqual >::Intersection (
            self_type const & S )  [inline]
```

Returns the intersection of the set with another set S.

Complexity

$O(n)$ where n is the number of elements in set S

Parameters

S	second set

Returns

intersection of both sets

6.21.2.14 operator=()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
self_type& Set< Key, Hasher, KeyEqual >::operator= (
            const self_type & rhs )  [inline]
```

Overloaded copy operator.

Parameters

rhs	Set to be copied

Returns

Reference to new set

6.21.2.15 remove()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
void Set< Key, Hasher, KeyEqual >::remove (
            Key element )  [inline]
```

Removes specified element from set.

Parameters

element	Element to be removed

6.21.2.16 Union()

```
template<class Key, class Hasher = std::hash<Key>, class KeyEqual = std::equal_to<Key>>
self_type Set< Key, Hasher, KeyEqual >::Union (
            self_type const & S )  [inline]
```

Returns the union of the set with another set S.

Complexity

$O(n+m)$ where n and m are both sets' respective sizes

Parameters

S	second set

Returns

union of both sets

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

- src/Set/Set.h

6.22 Automata::State Class Reference

A class to represent a state of a finite automata.

`#include <State.h>`

Classes

- class Hasher

  Functor which returns a `size_t` hash for the state.

Public Types

- typedef Set< Transition, Transition::Hasher > transition_set_type

  Set of transitions.

Public Member Functions

- State (std::string name, bool is_end=false, size_t bucket_count=100)

  Constructs a state given a unique name and whether the state is final or not.
- ~State ()

  Destructor for state.
- void addTransition (State *destination, char symbol)

  Adds a transition from this state to the destination on a given symbol.
- void setEnd (bool is_end)

  Sets whether the state is final or not.
- void setName (std::string name)

  Sets the name of the state.
- std::string name () const

  Returns the name of the state.
- bool isEnd () const

  Returns whether the state is final or not.
- transition_set_type & transition_set ()

  Returns a reference to the transition set.
- transition_set_type const & transition_set () const

  Returns a const reference to the transition set.

6.22.1 Detailed Description

A class to represent a state of a finite automata.

Description

This class models the behaviour of the state of a finite automata and assigns each one a name. A state is composed of:

- A name
- A set of transitions (edges connecting to other states)
- An attribute to indicate whether the state is final.

We make the distinction that each state has a unique name. That is to say we take for granted that the state will have a name like: "s1" "state3" "whateveryouwant"

Adding an indentical transition has no effect on the set and therefore isn't added.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 State()

```
Automata::State::State (
            std::string name,
            bool is_end = false,
            size_t bucket_count = 100 )
```

Constructs a state given a unique name and whether the state is final or not.

Parameters

name	Name of state
is_ end	`true` if the state is final, `false` otherwise
bucket_ count	Number of buckets to be used for a transition set

6.22.3 Member Function Documentation

6.22.3.1 addTransition()

```
void Automata::State::addTransition (
            State * destination,
            char symbol )
```

Adds a transition from this state to the destination on a given symbol.

Parameters

destination	Destination state
symbol	Symbol required to move from this state to destination

6.22.3.2 isEnd()

```
bool Automata::State::isEnd ( ) const
```

Returns whether the state is final or not.

Returns

`true` if final, `false` otherwise

6.22.3.3 name()

```
std::string Automata::State::name ( ) const
```

Returns the name of the state.

Returns

Name of state

6.22.3.4 setEnd()

```
void Automata::State::setEnd (
            bool is_end )
```

Sets whether the state is final or not.

Parameters

is_end	`true` if state is final, `false` otherwise

6.22.3.5 setName()

```
void Automata::State::setName (
            std::string name )
```

Sets the name of the state.

Parameters

name	Name of state

6.22.3.6 transition_set() [1/2]

```
State::transition_set_type & Automata::State::transition_set ( )
```

Returns a reference to the transition set.

Returns

Reference to the transition set

6.22.3.7 transition_set() [2/2]

```
State::transition_set_type const & Automata::State::transition_set ( ) const
```

Returns a const reference to the transition set.

Returns

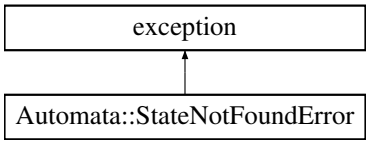
Const reference to the transition set

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

- src/Automata/State.h
- src/Automata/State.cpp

6.23 Automata::StateNotFoundError Class Reference

Inheritance diagram for Automata::StateNotFoundError:



Public Member Functions

- **StateNotFoundError** (std::string name)
- virtual const char * **what** () const throw ()

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

- src/Automata/AutomataErrors.h

6.24 Regex::Token::Tag Class Reference

Class to represent the different token classes.

`#include <Token.h>`

Public Member Functions

- Tag ()
  Empty constructor.
- Tag (int id, std::string name)
- **Tag** (const Tag &tag)
- int id () const
  Returns the id of tag.
- std::string name () const
  Returns the name of the tag.
- bool **operator==** (Tag const &rhs)
- const bool **operator==** (Tag const &rhs) const
- bool **operator!=** (Tag const &rhs)
- const bool **operator!=** (Tag const &rhs) const
- Tag & operator= (Tag const &rhs)
  Overloaded assignment operator.

Friends

- std::ostream & **operator<<** (std::ostream &os, Tag const &obj)

6.24.1 Detailed Description

Class to represent the different token classes.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 Tag()

```
Regex::Token::Tag::Tag (
            int id,
            std::string name )  [inline]
```

Constructor to instantiate a tag for a token

Parameters

id	Id of token class
name	Name of class

6.24.3 Member Function Documentation

6.24.3.1 id()

```
int Regex::Token::Tag::id ( ) const  [inline]
```

Returns the id of tag.

Returns

Id of tag

6.24.3.2 name()

```
std::string Regex::Token::Tag::name ( ) const  [inline]
```

Returns the name of the tag.

Returns

Name of tag

6.24.3.3 operator=()

```
Tag& Regex::Token::Tag::operator= (
            Tag const & rhs )  [inline]
```

Overloaded assignment operator.

Parameters

rhs	Right hand side

Returns

Reference to new Tag

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

- src/Regex/Token.h

6.25 Regex::Token Class Reference

Token class to represent a Token during lexing of the Regular Expression grammar.

`#include <Token.h>`

Classes

- class Tag
  Class to represent the different token classes.

Public Member Functions

- Token ()
  Empty constructor for Token.
- Token (Tag tag, std::string l)
- Tag tag () const
  Returns tag of token.
- std::string lexeme ()
  Returns associated lexeme of token.
- size_t length ()
  Returns length of lexeme.
- Token & operator= (Token const &rhs)
  Overloaded assignment operator.

Friends

- std::ostream & **operator**<< (std::ostream &os, Token const &obj)

6.25.1 Detailed Description

Token class to represent a Token during lexing of the Regular Expression grammar.

Description

This is a template class that manipulates "Token" types during parsing. An object of type Token stores the lexeme and has an attribute tag to indicate which "class" the token belongs to. A representation could be this:

Token < relop, "<=" >

6.25.2 Constructor & Destructor Documentation

6.25.2.1 Token()

```
Regex::Token::Token (
            Tag tag,
            std::string l )  [inline]
```

Constructor to instatiate a Token with its tag and lexeme

Parameters

tag	Corresponding tag for the token
l	Lexeme

6.25.3 Member Function Documentation

6.25.3.1 length()

```
size_t Regex::Token::length ( )  [inline]
```

Returns length of lexeme.

Returns

Length of lexeme

6.25.3.2 lexeme()

```
std::string Regex::Token::lexeme ( )  [inline]
```

Returns associated lexeme of token.

Returns

Lexeme

6.25.3.3 operator=()

```
Token& Regex::Token::operator= (
            Token const & rhs )  [inline]
```

Overloaded assignment operator.

Parameters

rhs	Right hand side

Returns

Reference to new token

6.25.3.4 tag()

```
Tag Regex::Token::tag ( ) const  [inline]
```

Returns tag of token.

Returns

Token tag

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

- src/Regex/Token.h

6.26 Automata::Transition Class Reference

Class which represents a transition from one state to other.

`#include <Transition.h>`

Classes

- struct Hasher
  Hasher to hash a transition.

Public Member Functions

- Transition (State *source, State *destination, char symbol)
- ~Transition ()
  Destructor for transition.
- char symbol () const
  Returns the symbol.
- State * source () const
  Returns the source state.
- State * destination () const
  Returns destination state.

6.26.1 Detailed Description

Class which represents a transition from one state to other.

Description

An object of the type Transition is meant to be inside an object of the type State. A transition can be thought of as an edge connecting a state to another. The transition indicates:

- The source state
- The required symbol to be consumed
- The destination state

6.26.2 Constructor & Destructor Documentation

6.26.2.1 Transition()

```
Automata::Transition::Transition (
            State * source,
            State * destination,
            char symbol )
```

for transition

Parameters

source	Source state
destination	Destination state
symbol	Symbol on which to move from source state to the destination state

6.26.3 Member Function Documentation

6.26.3.1 destination()

```
State * Automata::Transition::destination ( ) const
```

Returns destination state.

Returns

Destination state

6.26.3.2 source()

```
State * Automata::Transition::source ( ) const
```

Returns the source state.

Returns

Source state

6.26.3.3 symbol()

```
char Automata::Transition::symbol ( ) const
```

Returns the symbol.

Returns

Symbol

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

- src/Automata/Transition.h
- src/Automata/Transition.cpp

Chapter 7

File Documentation

7.1 src/Automata/NFA.cpp File Reference

```
#include <iostream>
#include "AutomataErrors.h"
#include "NFA.h"
```

7.1.1 Detailed Description

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17

Description

This is the .cpp file which contains the implementation for all the methods declared in the header file NFA.h

TODO

Nothing for the moment.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17

Description

This is the .cpp file which contains the implementation for all the methods declared in the header file Stae.h

TODO

Nothing for the moment.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17

Description

This is the .cpp file which contains the implementation for all the methods declared in the header file Transition.h

TODO

Nothing for the moment.

7.2 src/Automata/NFA.h File Reference

Header file which contains the errors associated with the construction of automatas.

```
#include "State.h"
#include "../Hashtable/Hashtable.h"
```

Classes

- class Automata::NFA

  This class models the behaviour of a non deterministic finite automaton.

7.2.1 Detailed Description

Header file which contains the errors associated with the construction of automatas.

Header file for the class NFA. This file contains declarations of methods and member variables.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17.

Description

This file contains various classes which are exceptions that will be thrown when an error occurs during the construction of NFAs.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17.

Description

This file contains the declarations for all the methods and member variables of class NFA. For the majority of the interface, we have based ourselves on the book:

- Compilers: Principles, Techniques and Tools (Aho et Al.)

TODO

- Review if we need to split this class into two classes BasicNFA and CompositeNFA. First one permits access to methods such as `addTransition()` and `addState()`. The second type is the result type of applying operations such as concatenation and union to two automatas. Because we lose the ability to reference states solely on the name, this should make sense.

7.3 src/Automata/State.h File Reference

Header file for the class State.

```
#include <string>
#include <vector>
#include "../Set/Set.h"
#include "Transition.h"
```

Classes

- class Automata::State

  A class to represent a state of a finite automata.
- class Automata::State::Hasher

  Functor which returns a `size_t` hash for the state.

Functions

- std::ostream & **Automata::operator**<< (std::ostream &os, const State &obj)
- bool **Automata::operator**== (State const &lhs, State const &rhs)

7.3.1 Detailed Description

Header file for the class State.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17

Description

It contains only the declarations for the methods and member variables of the class State

TODO

- THIS CLASS LEAKS MEMORY
- THERE IS A POINTER NOT BEING ALLOCATED

7.4 src/Automata/Transition.h File Reference

Header file for the Transition class.

```
#include <string>
```

Classes

- class Automata::Transition

  Class which represents a transition from one state to other.
- struct Automata::Transition::Hasher

  Hasher to hash a transition.

Functions

- bool **Automata::operator**== (Transition const &lhs, Transition const &rhs)

7.4.1 Detailed Description

Header file for the Transition class.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17.

TODO:

- THIS CLASS LEAKS MEMORY
- THERE IS A POINTER NOT BEING ALLOCATED

7.5 src/Hashtable/Hashtable.h File Reference

```
#include <string>
#include <vector>
#include <list>
```

Classes

- class hashtable< Key, T, Hasher, KeyEqual >
- class hashtable< Key, T, Hasher, KeyEqual >::iterator
- class hashtable< Key, T, Hasher, KeyEqual >::const_iterator

7.5.1 Detailed Description

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17.

This header file has been compiled for C++11.

TODO

Nothing for the moment.

7.6 src/Hashtable/PearsonHashtable8.h File Reference

```
#include <iostream>
#include <string>
#include "Hashtable.h"
```

Classes

- class PearsonHasher8
  An implementation of a Pearson hashtable using our template class hashtable.
- class PearsonHashtable8< T >

7.6.1 Detailed Description

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17.

TODO

Nothing for the moment.

7.7 src/Regex/AutomataDecls.h File Reference

Header file containing useful automatas.

```
#include "../Automata/NFA.h"
```

Functions

- Automata::NFA getKleeneStarNFA ()
  Get an automata which parses a kleene star operator.
- Automata::NFA getAlternationNFA ()
  Get an automata which parses an alternation.
- Automata::NFA getQMarkNFA ()
  Get an automata which parses a question mark.
- Automata::NFA getLParenNFA ()
  Get an automata which parses a left parentheses.
- Automata::NFA getRParenNFA ()
  Get an automata which parses a right parentheses.
- Automata::NFA getKleenePlusNFA ()
  Get an automata which parses a right parentheses.
- Automata::NFA getAlphaCharNFA ()
  Get an automata which parses an alphanumeric character.
- Automata::NFA getAsciiNFA (char from, char to)
  Get an automata which parses an ascii character.
- Automata::NFA getWhiteSpaceNFA ()
  Get an automata which parses whitespace.

Variables

- const Automata::NFA KLEENE_NFA = getKleeneStarNFA()
  Accepts Kleene Stars.
- const Automata::NFA ALTERNATION_NFA = getAlternationNFA()
  Accepts Alternation operator.
- const Automata::NFA QMARK_NFA = getQMarkNFA()
  Accepts Question Mark operator.
- const Automata::NFA LPAREN_NFA = getLParenNFA()
  Accepts Left Parentheses.
- const Automata::NFA RPAREN_NFA = getRParenNFA()
  Accepts Right Parentheses.
- const Automata::NFA ALPHA_CHAR_NFA = getAlphaCharNFA()
  Accepts alphanumeric characters.
- const Automata::NFA ASCII_NFA = getAsciiNFA(33, 126)
  Accepts ascii characters in the range 33 ('!') to 126 ('~') inclusive.
- const Automata::NFA SPACE_NFA = getWhiteSpaceNFA()
  Accepts whitespace.
- const Automata::NFA KLEENE_PLUS_NFA = getKleenePlusNFA()
  Accepts the Kleene Plus operator.

7.7.1 Detailed Description

Header file containing useful automatas.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

4/11/17.

Description

This header file was created with the intention of providing useful automatas to lex a regular expression string input in the Lexer class. It contains automatas which parse expressions such as escape sequences and operators.

TODO

Nothing for the moment.

7.7.2 Function Documentation

7.7.2.1 getAlphaCharNFA()

```
Automata::NFA getAlphaCharNFA ( )
```

Get an automata which parses an alphanumeric character.

Returns

NFA parsing an alphanumeric character

7.7.2.2 getAlternationNFA()

```
Automata::NFA getAlternationNFA ( )
```

Get an automata which parses an alternation.

Returns

NFA parsing alternation

7.7.2.3 getAsciiNFA()

```
Automata::NFA getAsciiNFA (
            char from,
            char to )
```

Get an automata which parses an ascii character.

The range is inclusive on both sides

Parameters

from	Starting ascii character
to	Ending ascii character

Returns

NFA parsing an ascii character

7.7.2.4 getKleenePlusNFA()

```
Automata::NFA getKleenePlusNFA ( )
```

Get an automata which parses a right parentheses.

Returns

NFA parsing right parentheses

7.7.2.5 getKleeneStarNFA()

```
Automata::NFA getKleeneStarNFA ( )
```

Get an automata which parses a kleene star operator.

Returns

NFA parsing kleene star

7.7.2.6 getLParenNFA()

```
Automata::NFA getLParenNFA ( )
```

Get an automata which parses a left parentheses.

Returns

NFA parsing left parentheses

7.7.2.7 getQMarkNFA()

```
Automata::NFA getQMarkNFA ( )
```

Get an automata which parses a question mark.

Returns

NFA parsing question mark

7.7.2.8 getRParenNFA()

```
Automata::NFA getRParenNFA ( )
```

Get an automata which parses a right parentheses.

Returns

NFA parsing right parentheses

7.7.2.9 getWhiteSpaceNFA()

```
Automata::NFA getWhiteSpaceNFA ( )
```

Get an automata which parses whitespace.

Returns

NFA parsing a whitespace

7.8 src/Regex/Lexer.h File Reference

Header file for the Lexer class.

```
#include "../Hashtable/Hashtable.h"
#include "Token.h"
#include "../Automata/NFA.h"
#include <string>
#include <vector>
#include <functional>
```

Classes

- class Regex::Lexer

7.8.1 Detailed Description

Header file for the Lexer class.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17.

TODO: Nothing for the moment.

7.9 src/Regex/Parser.cpp File Reference

File containing the implementation of the class Parser.

```
#include <iostream>
#include "Parser.h"
#include "TokenDecls.h"
#include "RegexErrors.h"
```

7.9.1 Detailed Description

File containing the implementation of the class Parser.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

4/13/17.

TODO

- Add the prediction table to the doc

7.10 src/Regex/Parser.h File Reference

Header file for the Parser class of the Regex.

```
#include <string>
#include <vector>
#include <stack>
#include <queue>
#include "Lexer.h"
```

Classes

- class Regex::Parser

7.10.1 Detailed Description

Header file for the Parser class of the Regex.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

4/13/17.

Description

This file contains the class which parses the regex in order to build the automata and check for syntactic correctness.

TODO

Nothing for the moment.

7.11 src/Regex/Regex.h File Reference

```
#include <string>
#include "../Automata/NFA.h"
#include "Parser.h"
```

Classes

- class Regex::Regex

7.11.1 Detailed Description

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

4/18/17.

7.12 src/Regex/Token.h File Reference

Header file for the class Token.

```
#include <string>
```

Classes

- class Regex::Token
  Token class to represent a Token during lexing of the Regular Expression grammar.
- class Regex::Token::Tag
  Class to represent the different token classes.

7.12.1 Detailed Description

Header file for the class Token.

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

: 3/22/17.

TODO: Nothing for the moment.

7.13 src/Regex/TokenDecls.h File Reference

```
#include "Token.h"
```

7.13.1 Detailed Description

Author

Carlos Brito (carlos.brito524@gmail.com)

Date

3/22/17

Description

This file contains the declaration of tokens and token tags.

- We have instanced some of the fixed tokens which we already know will appear like LPAREN and KLEENE_STAR.
- We have instanced token tags which will aid in creating new tokens.

TODO

Nothing for the moment.

7.14 src/Set/Set.h File Reference

This header file contains the class declarations and definitions for Set.

```
#include <functional>
#include "../Hashtable/Hashtable.h"
```

Classes

- class Set< Key, Hasher, KeyEqual >
- class Set< Key, Hasher, KeyEqual >::iterator
- class Set< Key, Hasher, KeyEqual >::const_iterator

7.14.1 Detailed Description

This header file contains the class declarations and definitions for Set.

Details

Author: Carlos Brito (carlos.brito524@gmail.com) Date: 3/22/17.

Description

This file contains the definitions for a class Set. The file contains the definitions for the `iterator` and `const_iterator` classes which are embedded into the class Set.

TODO

- Implement operator=
- Implement bool SubsetOf(self_type left)

Index