RNA Stem Loop Visualizer

Generated by Doxygen 1.8.4

Thu Jan 16 2014 23:17:44

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

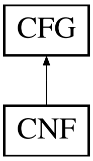
Class Documentation

3.1 CFG Class Reference

Class representing a context free grammar.

`#include <CFG.h>`

Inheritance diagram for CFG:



Public Member Functions

- CFG (const std::set< char > &terminals, const std::set< char > &variables, const std::multimap< char, SymbolString > &productions, const char &startsymbol)

  Constructor.
- virtual ~CFG ()

  Destructor.
- std::set< SymbolString > bodies (const char &v) const

  Get the set of bodies with the passed variable as head. For example if there are productions of the form A -> "a" and A -> "aA", then this will returns {"a", "aA"}.
- std::set< char > nullable () const

  Get all the nullable variables.
- void eleminateEpsilonProductions ()

  Eleminate epsilon productions. That is, eleminate productions of the form A -> , but doing so that the CFG still accepts the same language with epsilon (empty string excluded).
- std::set< std::pair< char, char > > units () const

  Get all the unit pairs of this CFG.
- void eleminateUnitProductions ()

  Eleminate unit productions. That is, eleminate productions of the form A -> B. But doing so that it does not affect the language of this CFG.
- std::set< char > generating () const

  Get all the generating symbols.
- std::set< char > reachable () const

  Get all the reachable symbols.

- void eleminateUselessSymbols ()

  Eleminate useless symbols. But doing so that is does not affect the language of this CFG.
- void cleanUp ()

  Clean up CFG, that is, eleminate epsilon productions, useless symbols and unit productions IN SAFE ORDER. This comes in handy for converting to CNF (Chomsky Normal Form). Also: removes all variables which don't have any production rules at all.
- std::set< char > **getTerminals** ()
- std::set< char > **getVariables** ()
- std::multimap< char, SymbolString > **getProductions** ()
- char **getStartsymbol** ()

Protected Attributes

- std::set< char > fTerminals

  The set of terminal symbols.
- std::set< char > fVariables

  The set of variables.
- std::multimap< char, SymbolString > fProductions

  The set of production rules.
- char **fStartSymbol**

3.1.1 Detailed Description

Class representing a context free grammar.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 CFG::CFG (const std::set< char > & *terminals,* const std::set< char > & *variables,* const std::multimap< char, SymbolString > & *productions,* const char & *startsymbol*)

Constructor.

Parameters

terminals	A set containing the terminals of the CFG.
variables	A set containing the variables of the CFG.
productions	A multimap that maps any symbol from the set of variables to a (possibly empty) SymbolString (which contains symbols from either the set of terminals or either the set of variables.
startsymbol	The startsymbol for the CFG.

Precondition

- The set of variables and the set of terminals are disjoints.
- The production rule is valid: the head consist of exactly one symbol that is in the set of the variables and the body must be empty or consisting of symbols that is either in the set of variables or in the set of terminals.
- The starting symbol must be a member of the set of variables.

Exceptions

std::invalid_argument	One of the preconditions were not met.

3.1.3 Member Function Documentation

3.1.3.1 std::set< SymbolString > CFG::bodies (const char & *v*) const

Get the set of bodies with the passed variable as head. For example if there are productions of the form A -> "a" and A -> "aA", then this will returns {"a", "aA"}.

Parameters

v	The variable representing the head of the production rules.

Returns

The set of SymbolString representing the body of the production rules whose head is the passed variable.

Precondition

- The passed variable must be in the set of the variables.

Exceptions

std::invalid_argument	The precondition were not satisfied.

3.1.3.2 void CFG::cleanUp ( )

Clean up CFG, that is, eleminate epsilon productions, useless symbols and unit productions IN SAFE ORDER. This comes in handy for converting to CNF (Chomsky Normal Form). Also: removes all variables which don't have any production rules at all.

Postcondition

The production rules doesn't contain any nullable symbols.
The production rules doesn't contain any useless symbols.
The CFG has only unit pairs of the form (A, A) for each A is a variable.

3.1.3.3 void CFG::eleminateEpsilonProductions ( )

Eleminate epsilon productions. That is, eleminate productions of the form A -> , but doing so that the CFG still accepts the same language with epsilon (empty string excluded).

Postcondition

The production rules doesn't contain any nullable symbols.

3.1.3.4 void CFG::eleminateUnitProductions ( )

Eleminate unit productions. That is, eleminate productions of the form A -> B. But doing so that it does not affect the language of this CFG.

Note

The algorithm only works if there is no cycle of unit productions. That is, unit pairs of the form A -> B, B -> C and C -> A. If that's the case, an exception will be thrown.

Exceptions

std::runtime_error	When there are cyclic unit pairs.

Postcondition

The CFG has only unit pairs of the form (A, A) for each A is a variable.

3.1.3.5 void CFG::eleminateUselessSymbols ( )

Eleminate useless symbols. But doing so that is does not affect the language of this CFG.

Postcondition

The production rules doesn't contain any useless symbols.
The CFG still accepts the same language.

3.1.3.6 std::set< char > CFG::generating ( ) const

Get all the generating symbols.

Returns

The set of generating symbols.

3.1.3.7 std::set< char > CFG::nullable ( ) const

Get all the nullable variables.

Returns

The set of all nullable variables.

3.1.3.8 std::set< char > CFG::reachable ( ) const

Get all the reachable symbols.

Returns

The set of all reachable symbols.

3.1.3.9 std::set< std::pair< char, char > > CFG::units ( ) const

Get all the unit pairs of this CFG.

Returns

The set of all unit pairs.

Exceptions

std::runtime_error	When there are cyclic unit pairs.

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

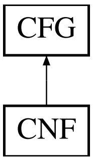
- CFG.h
- CFG.cpp

3.2 CNF Class Reference

The class CNF (Chomsky Normal Form), this is actually a CFG (Context Free Grammar) but with production rules of the form:

```
#include <CNF.h>
```

Inheritance diagram for CNF:



Public Member Functions

- CNF (const std::set< char > &terminals, const std::set< char > &variables, const std::multimap< char, SymbolString > &productions, const char &start)

  Constructor, initialize all datamembers. This will construct production rules based upon the rules of the CFG cleaned up variant and satisfying the conditions imposed by the CNF.
- bool CYK (const std::string &terminalstring) const

  Check whether the terminalstring is in the language of this CNF by using the CYK algorithm.

Additional Inherited Members

3.2.1 Detailed Description

The class CNF (Chomsky Normal Form), this is actually a CFG (Context Free Grammar) but with production rules of the form:

- A –> BC (with A, B and C variables) or
- A –> a (with A a variable and a a terminal)

3.2.2 Constructor & Destructor Documentation

3.2.2.1 CNF::CNF (const std::set< char > & *terminals*, const std::set< char > & *variables*, const std::multimap< char, SymbolString > & *productions*, const char & *start*)

Constructor, initialize all datamembers. This will construct production rules based upon the rules of the CFG cleaned up variant and satisfying the conditions imposed by the CNF.

Note

> You can still use the CFG methods as if it was only cleaned up (thus you can use the same variables for getting the bodies and such).

Parameters

terminals	The set of terminal symbols.
variables	The set of non-terminal symbols.
productions	The set of production rules (which is actually a std::map where each non-terminal symbol maps to a string consisting of terminals and variables.
start	The start symbol.

Postcondition

- The CFG methods will produce the same result as if it was already cleaned up without unwanted side-effects.

3.2.3 Member Function Documentation

3.2.3.1 bool CNF::CYK (const std::string & *terminalstring*) const

Check whether the terminalstring is in the language of this CNF by using the CYK algorithm.

Parameters

terminalstring	The string to be checked whether this is in the language of the CFG.

Returns

True if the terminalstring is in the language of this CFG, false if not.

Precondition

- The string passed consists only of terminal symbols in the set of terminals of this CFG.

Exceptions

std::invalid_argument	if the string passed is not a valid terminal string (that is, not consisting of terminal symbols).

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

- CNF.h
- CNF.cpp

3.3 LLP::LLParser Class Reference

Class representing an LL Parser.

```
#include <LLParser.h>
```

Public Member Functions

- LLParser (const std::set< char > &CFGTerminals, const std::set< char > &CFGVariables, const std::multimap< char, SymbolString > &CFGProductions, const char &CFGStartsymbol, const unsigned int lookahead)

  Constructor, constructs an LL Parser from the given elements of a context free grammar.
- LLParser (const CFG &grammar, const unsigned int lookahead)

  Constructor, constructs an LL Parser for the given context free grammar.
- bool process (const std::string &input) const

Process an input string through the LL Parser.

- virtual ∼LLParser ()

  Destructor.

Friends

- std::ostream & operator<< (std::ostream &stream, const LLParser &obj)

  Prints the parse table to the given output stream.

3.3.1 Detailed Description

Class representing an LL Parser.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 LLP::LLParser::LLParser (const std::set< char > & *CFGTerminals,* const std::set< char > & *CFGVariables,* const std::multimap< char, SymbolString > & *CFGProductions,* const char & *CFGStartsymbol,* const unsigned int *lookahead*)

Constructor, constructs an LL Parser from the given elements of a context free grammar.

Parameters

CFGTerminals	A set containing the terminals of the CFG
CFGVariables	A set containing the variables of the CFG
CFGProductions	A multimap that maps a variable to an symbolString
CFGStartsymbol	The startsymbol for the CFG
lookahead	The size of the lookahead (k)

Exceptions

invalid_argument	Throws this exception when the Parser can't be contructed

3.3.2.2 LLP::LLParser::LLParser (const CFG & *grammar,* const unsigned int *lookahead*)

Constructor, constructs an LL Parser for the given context free grammar.

Parameters

grammar	A context free grammar as a base for this LL Parser
lookahead	The size of the lookahead (k)

Exceptions

invalid_argument	Throws this exception when the Parser can't be contructed

3.3.3 Member Function Documentation

3.3.3.1 bool LLP::LLParser::process (const std::string & *input*) const

Process an input string through the LL Parser.

Parameters

input	The string to be processed by the LL Parser

Returns

A bool telling if the Parser accepted

3.3.4 Friends And Related Function Documentation

3.3.4.1 std::ostream& operator<< (std::ostream & *stream*, const **LLParser** & *obj*) [friend]

Prints the parse table to the given output stream.

Parameters

stream	The output stream
obj	The parser

Returns

The output stream.

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

- LLParser.h
- LLParser.cpp

3.4 LLP::LLTable Class Reference

Class representing an LL Parse Table.

```
#include <LLParser.h>
```

Public Member Functions

- LLTable (const std::set< char > &CFGTerminals, const std::set< char > &CFGVariables, const std::multimap< char, SymbolString > &CFGProductions, const unsigned int dimension)

  Constructor, constructs an LL Parse Table from the given elements of a context free grammar.
- LLTable (const CFG &grammar, const unsigned int dimension)

  Constructor, constructs an LL Parse Table for the given context free grammar.
- SymbolString process (const char &topStack, const SymbolString &remainingInput) const

  Process one input symbol of the remaining input string.
- virtual ∼LLTable ()

  Destructor.
- std::string toString (const std::set< char > &CFGTerminals, const std::set< char > &CFGVariables) const

  Returns a string representation of the parse table.

3.4.1 Detailed Description

Class representing an LL Parse Table.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 LLP::LLTable::LLTable (const std::set< char > & *CFGTerminals*, const std::set< char > & *CFGVariables*, const std::multimap< char, SymbolString > & *CFGProductions*, const unsigned int *dimension*)

Constructor, constructs an LL Parse Table from the given elements of a context free grammar.

Parameters

CFGTerminals	A set containing the terminals of the CFG
CFGVariables	A set containing the variables of the CFG
CFGProductions	A multimap that maps a variable to an symbolString
dimension	The dimension of the table, thus the size of the lookahead (k)

Exceptions

invalid_argument	Throws this exception when the Table can't be contructed

3.4.2.2 LLP::LLTable::LLTable (const CFG & *grammar,* const unsigned int *dimension*)

Constructor, constructs an LL Parse Table for the given context free grammar.

Parameters

grammar	A context free grammar as a base for this LL Parser
dimension	The dimension of the table, thus the size of the lookahead (k)

Exceptions

invalid_argument	Throws this exception when the Table can't be contructed

3.4.3 Member Function Documentation

3.4.3.1 SymbolString LLP::LLTable::process (const char & *topStack,* const SymbolString & *remainingInput*) const

Process one input symbol of the remaining input string.

Parameters

topStack	The variable at the top of the stack
remainingInput	The remaining part of the input string

Precondition

topStack is an element of CFGVariables

Exceptions

runtime_error	Throws this exception when the remaining input string results in an error

Returns

The right side of the used production rule.

3.4.3.2 std::string LLP::LLTable::toString (const std::set< char > & *CFGTerminals,* const std::set< char > & *CFGVariables*) const

Returns a string representation of the parse table.

Parameters

CFGTerminals	A set containing the terminals of the CFG
CFGVariables	A set containing the variables of the CFG

Returns

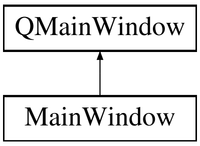
String representation.

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

- LLParser.h
- LLParser.cpp

3.5 MainWindow Class Reference

Inheritance diagram for MainWindow:



Public Member Functions

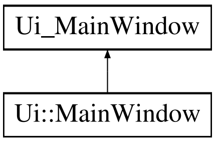
- **MainWindow** (QWidget ∗parent=0)

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

- mainwindow.h
- mainwindow.cpp

3.6 Ui::MainWindow Class Reference

Inheritance diagram for Ui::MainWindow:



Additional Inherited Members

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

- ui_mainwindow.h

3.7 PDA Class Reference

Class representing a PDA.

```
#include <PDA.h>
```

Public Member Functions

- PDA (const std::set< char > &alphabetPDA, const std::set< char > &alphabetStack, const PDAFinal &PDAending)

  Constructor.
- PDA (CFG cfg)

  Constructor.
- PDA (const std::string &fileName)

  Constructor.
- bool addState (const PDAState &state)

  Add a new state to the PDA.
- bool addState (const PDAState &state, const bool &isStarting)

  Add a new state to the PDA.
- bool addTransition (PDATransition transition)

  Add a new transition to the PDA.
- bool process (std::string input)

  Process an input string through the PDA.
- bool toDotFile (std::string fileName)

  Store an PDA in a dot file.

Friends

- std::ostream & operator<< (std::ostream &out, PDA pda)

  << overloading

3.7.1 Detailed Description

Class representing a PDA.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 PDA::PDA (const std::set< char > & *alphabetPDA,* const std::set< char > & *alphabetStack,* const PDAFinal & *PDAending*)

Constructor.

Parameters

alphabetPDA	A set containing characters representing the alphabet of the PDA
alphabetStack	A set containing characters representing the alphabet of the stack
PDAending	The type of PDA(STACK: PDA is final with empty stack, STATE: PDA is final when it reaches an empty state)

3.7.2.2 PDA::PDA (CFG *cfg*)

Constructor.

Parameters

cfg	A Context Free Grammar to be transformed to a PDA

3.7.2.3 PDA::PDA (const std::string & *fileName*)

Constructor.

Parameters

fileName	A XML file containing info about the PDA

3.7.3 Member Function Documentation

3.7.3.1 bool PDA::addState (const PDAState & *state*)

Add a new state to the PDA.

Parameters

state	An PDAState object representing an PDA state

Returns

A bool telling if the state is added or not

3.7.3.2 bool PDA::addState (const PDAState & *state,* const bool & *isStarting*)

Add a new state to the PDA.

Parameters

state	A PDAState object representing an PDA state
isStarting	A bool describing if the state is the start state

Returns

A bool telling if the state is added or not

3.7.3.3 bool PDA::addTransition (PDATransition *transition*)

Add a new transition to the PDA.

Parameters

transition	A PDATransition object representing an PDA transition

Returns

A bool telling if the transition is added or not

3.7.3.4 bool PDA::process (std::string *input*)

Process an input string through the PDA.

Parameters

input	The string to be processed by the PDA

Returns

A bool telling if the PDA ended in a final state or empty stack

3.7.3.5 bool PDA::toDotFile (std::string *fileName*)

Store an PDA in a dot file.

Parameters

fileName	The file to write to

A bool telling if creating and writing the file was succesfull

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

- PDA.h
- PDA.cpp

3.8 PDAID Class Reference

Class representing a PDA Instantenious Description.

`#include <PDA.h>`

Public Member Functions

- PDAID (const std::string &input, PDAState *currentState, const std::stack< char > stack)

  Constructor.
- void step (const std::string &input, PDAState *currentState, const std::stack< char > stack)

  Process the ID according to one transition for one step.
- bool isAccepted (PDAFinal pdaType)

  Check if this ID will be accepted by the PDA.
- std::string getInput ()

  get the input of the ID
- PDAState * getState ()

  get the state of the ID
- std::stack< char > getStack ()

  get the stack of the ID

Friends

- std::ostream & operator<< (std::ostream &out, PDAID id)

  << overloading

3.8.1 Detailed Description

Class representing a PDA Instantenious Description.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 PDAID::PDAID (const std::string & *input*, PDAState $*$ *currentState*, const std::stack$<$ char $>$ *stack*)

Constructor.

Parameters

input	The input string
currentState	The pointer to state where the ID starts
stack	The stack at this moment

3.8.3 Member Function Documentation

3.8.3.1 std::string PDAID::getInput ( ) [inline]

get the input of the ID

Returns

string

3.8.3.2 std::stack<char> PDAID::getStack ( ) [inline]

get the stack of the ID

Returns

stack with chars

3.8.3.3 PDAState* PDAID::getState ( ) [inline]

get the state of the ID

Returns

PDAState pointer

3.8.3.4 bool PDAID::isAccepted (PDAFinal *pdaType*)

Check if this ID will be accepted by the PDA.

Parameters

pdaType	Type of PDA(State or Stack)

Returns

bool telling if the ID is accepted

3.8.3.5 void PDAID::step (const std::string & *input,* PDAState * *currentState,* const std::stack< char > *stack*)

Process the ID according to one transition for one step.

Parameters

to	Pointer to next state
inputSymbol	Character accompanied with this transition
topStack	Character that should be at the top of the stack after the transition

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

- PDA.h
- PDA.cpp

3.9 PDAState Class Reference

Class representing a state from a PDA.

```
#include <PDA.h>
```

Public Member Functions

- PDAState (std::string name)

  Constructor.
- PDAState (std::string name, bool isFinal)

  Constructor.
- bool isFinal () const

  Check if a state is final.
- std::string getName () const

  Get the name of the state.
- bool operator== (const PDAState &other)

  == operator overloading

Friends

- std::ostream & operator<< (std::ostream &out, PDAState state)

  << overloading

3.9.1 Detailed Description

Class representing a state from a PDA.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 PDAState::PDAState (std::string *name*)

Constructor.

Parameters

name	The name of the PDAState

3.9.2.2 PDAState::PDAState (std::string *name,* bool *isFinal*)

Constructor.

Parameters

name	The name of the PDAState
isFinal	Bool which tells if this state is final or not

3.9.3 Member Function Documentation

3.9.3.1 std::string PDAState::getName ( ) const

Get the name of the state.

Returns

A string representing the name

3.9.3.2 bool PDAState::isFinal ( ) const

Check if a state is final.

Returns

A bool telling if it's true or not

3.9.3.3 bool PDAState::operator== (const **PDAState** & *other*)

== operator overloading

Returns

Bool telling if the two states are equal

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

- PDA.h
- PDA.cpp

3.10 PDATransition Class Reference

Class representing a transition from a PDA.

```
#include <PDA.h>
```

Public Member Functions

- PDATransition (PDAState ∗from, PDAState ∗to, const char &input, const char &stackTop, const PDAStackOperation &stackOperation)

  Constructor.
- PDATransition (PDAState ∗from, PDAState ∗to, const char &input, const char &stackTop, const PDAStackOperation &stackOperation, const char &stackPush)

  Constructor.
- PDATransition (PDAState ∗from, PDAState ∗to, const char &input, const char &stackTop, const PDAStackOperation &stackOperation, const std::vector< char > &stackPush)

  Constructor.

- bool operator== (const PDATransition &other)

    == operator overloading
- void stackOperation (std::stack< char > &in)

    Change a stack based upon the data in the transition.
- PDAState ∗ getFrom ()

    get the from state of the transition
- PDAState ∗ getTo ()

    get the to state of the transition
- char getInputSymbol ()

    get the input symbol of the transition
- char getTopStack ()

    get the symbol on the top of the stack in this transition
- PDAStackOperation getStackOperation ()

    get the stack operation of the transition
- std::vector< char > getPushStack ()

    get the characters which are popped on the stack during a push operation
- void setFrom (PDAState ∗from)

    change the from state in the transition
- void setTo (PDAState ∗to)

    change the to state in the transition

Friends

- std::ostream & operator<< (std::ostream &out, PDATransition transition)

    << overloading

3.10.1 Detailed Description

Class representing a transition from a PDA.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 PDATransition::PDATransition (**PDAState** ∗ *from,* **PDAState** ∗ *to,* const char & *input,* const char & *stackTop,* const PDAStackOperation & *stackOperation*)

Constructor.

Parameters

from	A pointer to the PDAState where this transition is coming from
to	A pointer to the PDAState where this transition is going to
input	The input symbol for the transition
stackTop	Define what should be on the top of the stack when processing this transition
stackOperation	Should the stack be popped, pushed or stay as it is

3.10.2.2 PDATransition::PDATransition (**PDAState** ∗ *from,* **PDAState** ∗ *to,* const char & *input,* const char & *stackTop,* const PDAStackOperation & *stackOperation,* const char & *stackPush*)

Constructor.

Parameters

from	A pointer to the PDAState where this transition is coming from
to	A pointer to the PDAState where this transition is going to
input	The input symbol for the transition
stackTop	Define what should be on the top of the stack when processing this transition
stackOperation	Should the stack be popped, pushed or stay as it is
stackPush	The character that should be pushed to the stack during the transition

3.10.2.3 PDATransition::PDATransition (PDAState ∗ *from*, PDAState ∗ *to*, const char & *input*, const char & *stackTop*, const PDAStackOperation & *stackOperation*, const std::vector< char > & *stackPush*)

Constructor.

Parameters

from	A pointer to the PDAState where this transition is coming from
to	A pointer to the PDAState where this transition is going to
input	The input symbol for the transition
stackTop	Define what should be on the top of the stack when processing this transition
stackOperation	Should the stack be popped, pushed or stay as it is
stackPush	The character vector that should be pushed to the stack during the transition

3.10.3 Member Function Documentation

3.10.3.1 PDAState∗ PDATransition::getFrom ( ) [inline]

get the from state of the transition

Returns

PDAState pointer

3.10.3.2 char PDATransition::getInputSymbol ( ) [inline]

get the input symbol of the transition

Returns

char

3.10.3.3 std::vector<char> PDATransition::getPushStack ( ) [inline]

get the characters which are popped on the stack during a push operation

Returns

vector with chars

3.10.3.4 PDAStackOperation PDATransition::getStackOperation ( ) [inline]

get the stack operation of the transition

Returns

PDAStackOperation

3.10.3.5 PDAState* PDATransition::getTo ( ) [inline]

get the to state of the transition

Returns

PDAState pointer

3.10.3.6 char PDATransition::getTopStack ( ) [inline]

get the symbol on the top of the stack in this transition

Returns

char

3.10.3.7 bool PDATransition::operator== (const PDATransition & *other*)

== operator overloading

Returns

Bool telling if the two transitions are equal

3.10.3.8 void PDATransition::setFrom (PDAState * *from*) [inline]

change the from state in the transition

Parameters

from	PDAState pointer to the state

3.10.3.9 void PDATransition::setTo (PDAState * *to*) [inline]

change the to state in the transition

Parameters

from	PDAState pointer to the state

3.10.3.10 void PDATransition::stackOperation (std::stack< char > & *in*)

Change a stack based upon the data in the transition.

Parameters

in	A stack with chars representing the stack in the PDA. Be careful! The stack is given by reference

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

- PDA.h
- PDA.cpp

3.11 LLP::RNAParser Class Reference

Class representing an RNA Parser based on an LLParser.

```
#include <LLParser.h>
```

Static Public Member Functions

- static bool parse (std::string input, unsigned int stemsize)

  Parses the given string. Checks if the given RNA string is a vallid stemloop of the given size.
- static unsigned int parse (const std::string input)

  Parses the given string. Checks if the given RNA string is a vallid stemloop.
- static unsigned int parse (const std::string &input, unsigned int &b_stemsize, unsigned int &b_begin, unsigned int &b_end, unsigned int begin=0, unsigned int end=0)

  Parses the given string.
- static bool isElement (char c)

  Indicates whether the given character is a vallid RNA-element.

3.11.1 Detailed Description

Class representing an RNA Parser based on an LLParser.

3.11.2 Member Function Documentation

3.11.2.1 bool LLP::RNAParser::isElement (char *c*) [static]

Indicates whether the given character is a vallid RNA-element.

Parameters

c	Possible RNA-element

Returns

True if 'c' is a vallid RNA-element

3.11.2.2 bool LLP::RNAParser::parse (std::string *input,* unsigned int *stemsize*) [static]

Parses the given string. Checks if the given RNA string is a vallid stemloop of the given size.

Parameters

input	RNA string
stemSize	The size of the stem

Returns

True if 'input' is a vallid RNA string with stem of size 'stemSize'

3.11.2.3 unsigned int LLP::RNAParser::parse (const std::string *input*) [static]

Parses the given string. Checks if the given RNA string is a vallid stemloop.

Parameters

input	RNA string

Returns

If vallid the stemsize of the RNA-string, else 0

3.11.2.4 unsigned int LLP::RNAParser::parse (const std::string & *input*, unsigned int & *b_stemsize*, unsigned int & *b_begin*, unsigned int & *b_end*, unsigned int *begin* = `0`, unsigned int *end* = `0`) `[static]`

Parses the given string.

Parameters

input	RNA string
b_stemsize	the size of the best founded stemloop
b_begin	the begin of the best founded stemloop
b_end	the end of the best founded stemloop
begin	indicates the possible begin of a stemloop
end	indicates the possible end of a stemloop (iterator like)

Returns

If vallid the stemsize of the RNA-string, else 0

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

- LLParser.h
- LLParser.cpp

3.12 RNAString Class Reference

Public Member Functions

- RNAString ()

  default constructor
- RNAString (Tape tape)

  Constructor using the information stored in a Tape.
- char getLetterAt (int i) const

  Gets the i'th letter (nucleotide) in the string.
- char getLoopSignAt (int i) const

  Gets the i'th loop sign (part of stem or loop?) in the string.
- void push_front (char nucl)

  add character in front of loop
- void push_back (char nucl)

  add character in back of loop
- int getSize () const

  gets the size of the string
- int getLoopStartIndex () const

  Gets the index of the first nucleotide in the loop.
- int getLoopEndIndex () const

  Gets the index of the first nucleotide in the loop.

- int getStemSize () const

  Gets the size of the stem of the stemloop.
- std::string getString () const

  get the RNAString as a std::string;

Friends

- std::ostream & **operator**<< (std::ostream &os, RNAString)

3.12.1 Constructor & Destructor Documentation

3.12.1.1 RNAString::RNAString (Tape *tape*)

Constructor using the information stored in a Tape.

Parameters

tape	The tape

3.12.2 Member Function Documentation

3.12.2.1 char RNAString::getLetterAt (int *i*) const

Gets the i'th letter (nucleotide) in the string.

i Index of the nucleotide

Returns

The nucleotide (as a char)

3.12.2.2 int RNAString::getLoopEndIndex ( ) const

Gets the index of the first nucleotide in the loop.

Returns

the index

3.12.2.3 char RNAString::getLoopSignAt (int *i*) const

Gets the i'th loop sign (part of stem or loop?) in the string.

i Index of the nucleotide

Returns

0 if loop, X if stem, ? if neither

3.12.2.4 int RNAString::getLoopStartIndex ( ) const

Gets the index of the first nucleotide in the loop.

Returns

the index

3.12.2.5 int RNAString::getSize ( ) const

gets the size of the string

Returns

size of the string

3.12.2.6 int RNAString::getStemSize ( ) const

Gets the size of the stem of the stemloop.

Returns

the size

3.12.2.7 std::string RNAString::getString ( ) const

get the RNAString as a std::string;

Returns

A std::string

3.12.2.8 void RNAString::push_back (char *nucl*)

add character in back of loop

Parameters

nucl	Nucleotide type

3.12.2.9 void RNAString::push_front (char *nucl*)

add character in front of loop

Parameters

nucl	Nucleotide type

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

- RNAString.h
- RNAString.cpp

3.13 RNAVisualizer Class Reference

Public Member Functions

- void **visualize** (const std::string &sequence, const unsigned int &stemsize, const unsigned int &loopstart, const unsigned int &loopend)

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

- RNAVisualizer.h
- RNAVisualizer.cpp

3.14 Tape Class Reference

Class representing the tape for a Turing Machine.

```
#include <Turing.h>
```

Public Member Functions

- Tape (const std::string &input, char blank, int trackCount)

  Constructor.
- std::vector< char > **getSymbolsAtHead** () const
- void replaceSymbolsAtHead (const std::vector< char > &symbols)

  Replaces symbol(s) at given position by given symbol(s)
- void moveHead (Direction dir)

  Move head one spot.
- void resetHead ()

  Moves the head to the very first nonblank character.

Friends

- std::ostream & operator<< (std::ostream &output, const Tape &T)

  output overload

3.14.1 Detailed Description

Class representing the tape for a Turing Machine.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 Tape::Tape (const std::string & *input*, char *blank*, int *trackCount*)

Constructor.

Parameters

input	String to write to tape
blank	Blank symbol
trackCount	number of tracks on the tape

3.14.3 Member Function Documentation

3.14.3.1 void Tape::moveHead (Direction *dir*)

Move head one spot.

Parameters

dir	Left or right

3.14.3.2 void Tape::replaceSymbolsAtHead (const std::vector< char > & *symbols*)

Replaces symbol(s) at given position by given symbol(s)

Parameters

symbol	The symbol(s) to be written to tape

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

- Turing.h
- Turing.cpp

3.15 TMID Class Reference

Class representing Turing Machine Instantaneous Description.

```
#include <Turing.h>
```

Public Member Functions

- TMID (const std::string &input, StatePtr startState, char blank, int trackCount)

  Constructor.
- std::pair< StatePtr,
  std::vector< char > > getStateAndSymbols () const

  Gets the symbol at the current head position and the current state of the ID.
- void step (StatePtr to, const std::vector< char > &write, Direction dir)

  Processes the ID according to one transition for one step.
- const Tape & getTape () const

  Gets the full Tape.

Friends

- std::ostream & operator<< (std::ostream &output, const TMID &ID)

  output overload

3.15.1 Detailed Description

Class representing Turing Machine Instantaneous Description.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 TMID::TMID (const std::string & *input*, StatePtr *startState*, char *blank*, int *trackCount*)

Constructor.

Parameters

input	The input string
state	The start state
blank	The blank symbol for the tape
trackCount	Number of tracks on the tape

3.15.3 Member Function Documentation

3.15.3.1 std::pair< StatePtr, std::vector< char > > TMID::getStateAndSymbols ( ) const

Gets the symbol at the current head position and the current state of the ID.

Returns

Pair of the symbol and the state

3.15.3.2 const Tape & TMID::getTape ( ) const

Gets the full Tape.

Returns

the full Tape

3.15.3.3 void TMID::step (StatePtr *to*, const std::vector< char > & *write*, Direction *dir*)

Processes the ID according to one transition for one step.

Parameters

to	Pointer to next state
write	Symbol to be written to tape
dir	Direction to move head in

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

- Turing.h
- Turing.cpp

3.16 TuringMachine Class Reference

Class representing a Turing Machine.

```
#include <Turing.h>
```

Public Member Functions

- TuringMachine ()

  Constructor (will construct TM with empty alphabets, empty state, final state and transition sets and no blank symbol or start state)
- TuringMachine (const std::set< char > &alphabetTuring, const std::set< char > &alphabetTape, char tapeBlank)

  Constructor.
- bool addState (const std::string &state, bool isStarting=0, bool isFinal=0, const std::vector< char > &storage=std::vector< char >())

  Adds a new state to the Turing Machine.
- bool addTransition (const std::string &from, const std::string &to, char read, char write, Direction dir, const std::vector< char > &fromStorage=std::vector< char >(), const std::vector< char > &toStorage=std::vector< char >())

  Add a new transition to the Turing Machine (single track only)

- bool addTransition (const std::string &from, const std::string &to, const std::vector< char > &read, const std::vector< char > &write, Direction dir, const std::vector< char > &fromStorage=std::vector< char >(), const std::vector< char > &toStorage=std::vector< char >())

  Add a new transition to the Turing Machine (multitrack supported)
- bool indicateStartState (const std::string &name, const std::vector< char > &storage=std::vector< char >())

  Indicates start state pointer after adding the states (so the state has to be in the TM first!). Useful for XML with state storage.
- bool indicateAcceptingState (const std::string &name, const std::vector< char > &storage=std::vector< char >())

  Indicates an accepting state after adding the states (so the state has to be in the TM first!). Useful for XML with state storage.
- bool process (const std::string &input) const

  Processes an input string through the Turing Machine.
- std::tuple< bool, Tape > processAndGetTape (const std::string &input) const

  Processes an input string through the Turing Machine and return accepting Tape.
- virtual ~TuringMachine ()

  Destructor.

3.16.1 Detailed Description

Class representing a Turing Machine.

3.16.2 Constructor & Destructor Documentation

3.16.2.1 TuringMachine::TuringMachine (const std::set< char > & *alphabetTuring,* const std::set< char > & *alphabetTape,* char *tapeBlank*)

Constructor.

Parameters

alphabetTuring	A set containing characters representing the alphabet of the Turing Machine
alphabetTape	A set containing characters representing the alphabet of the tape
tapeBlank	The blank symbol for the tape

3.16.3 Member Function Documentation

3.16.3.1 bool TuringMachine::addState (const std::string & *state,* bool *isStarting* = 0, bool *isFinal* = 0, const std::vector< char > & *storage* = `std::vector<char> ()`)

Adds a new state to the Turing Machine.

Parameters

state	The name of the state to be added
isStarting	A bool indicating whether the state is the start state
isEnding	A bool indicating whether the state is a final state

Returns

True if state was added

Note that start and final states may also be set to true afterwards. Comes in handy when adding states through loops (when adding states with different storages!)

3.16.3.2 bool TuringMachine::addTransition (const std::string & *from*, const std::string & *to*, char *read*, char *write*, Direction *dir*, const std::vector< char > & *fromStorage* = `std::vector<char> ()`, const std::vector< char > & *toStorage* = `std::vector<char> ()`)

Add a new transition to the Turing Machine (single track only)

Parameters

from	The name of the state where the transition starts
to	The name of the state where the transition leads to
read	The symbol read from the tape
write	The symbol to write to the tape
dir	The direction to move tape head in
fromStorage	Storage for the start state
toStorage	Storage for the start state

Returns

True if transition was added

3.16.3.3 bool TuringMachine::addTransition (const std::string & *from*, const std::string & *to*, const std::vector< char > & *read*, const std::vector< char > & *write*, Direction *dir*, const std::vector< char > & *fromStorage* = `std::vector<char> ()`, const std::vector< char > & *toStorage* = `std::vector<char> ()`)

Add a new transition to the Turing Machine (multitrack supported)

Parameters

from	The name of the state where the transition starts
to	The name of the state where the transition leads to
read	The vector of symbols read from the tape
write	The vector of symbol to write to the tape
dir	The direction to move tape head in
fromStorage	Storage for the start state
toStorage	Storage for the start state

Returns

A bool telling if the transition is added or not

3.16.3.4 bool TuringMachine::indicateAcceptingState (const std::string & *name*, const std::vector< char > & *storage* = `std::vector<char> ()`)

Indicates an accepting state after adding the states (so the state has to be in the TM first!). Useful for XML with state storage.

Parameters

name	Name of the accepting state
storage	Storage of the accepting state

Returns

True if start state pointer was added

3.16.3.5 bool TuringMachine::indicateStartState (const std::string & *name*, const std::vector< char > & *storage* = `std::vector<char> ()`)

Indicates start state pointer after adding the states (so the state has to be in the TM first!). Useful for XML with state storage.

Parameters

name	Name of the start state
storage	Storage of the start state

Returns

True if start state pointer was added

3.16.3.6 bool TuringMachine::process (const std::string & *input*) const

Processes an input string through the Turing Machine.

Parameters

input	The string to be processed by the Turing Machine

Returns

True if the input string is part of the language described by the TM

3.16.3.7 std::tuple< bool, Tape > TuringMachine::processAndGetTape (const std::string & *input*) const

Processes an input string through the Turing Machine and return accepting Tape.

Parameters

input	The string to be processed by the Turing Machine

Returns

tuple of bool if the string was accepted and the Tape

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

- Turing.h
- Turing.cpp

3.17 TuringState Class Reference

Class representing a state of a Turing Machine.

`#include <Turing.h>`

Public Member Functions

- TuringState (const std::string &name)

  Constructor.
- TuringState (const std::string &name, const std::vector< char > &storage)

  Constructor.
- bool isCalled (const std::string &name) const

  Checks if name of state is given name.
- bool hasThisStorage (const std::vector< char > &storage) const

  Checks if the state has given storage.
- virtual ~TuringState ()

  Destructor.

Friends

- std::ostream & operator<< (std::ostream &output, const TuringState &TS)

  Output overload.

3.17.1 Detailed Description

Class representing a state of a Turing Machine.

3.17.2 Constructor & Destructor Documentation

3.17.2.1 TuringState::TuringState (const std::string & *name*)

Constructor.

Parameters

name	The name of the TuringState

3.17.2.2 TuringState::TuringState (const std::string & *name,* const std::vector< char > & *storage*)

Constructor.

Parameters

name	The name of the TuringState
storage	The storage in the TuringState

3.17.3 Member Function Documentation

3.17.3.1 bool TuringState::hasThisStorage (const std::vector< char > & *storage*) const

Checks if the state has given storage.

Parameters

storage	The storage to check for

Returns

True if storage is given storage

3.17.3.2 bool TuringState::isCalled (const std::string & *name*) const

Checks if name of state is given name.

Parameters

name	Name to check

Returns

True if name is given name

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

- Turing.h
- Turing.cpp

3.18 TuringTransition Class Reference

Class representing a transition of a Turing Machine.

`#include <Turing.h>`

Public Member Functions

- TuringTransition (StatePtr from, StatePtr to, const std::vector< char > &read, const std::vector< char > &write, Direction dir)

    Constructor.
- bool match (StatePtr state, const std::vector< char > &symbols) const

    Checks if transition is for given state and tape symbol.
- std::tuple< StatePtr,
std::vector< char >, Direction > getTransition () const

    Gets next state, symbol(s) to write and direction.
- bool isThisTransition (StatePtr from, StatePtr to, const std::vector< char > &read, const std::vector< char > &write, Direction dir) const

    Checks if all of the given arguments are also the transition's.
- virtual ∼TuringTransition ()

Friends

- std::ostream & operator<< (std::ostream &output, const TuringTransition &TT)

    output overload

3.18.1 Detailed Description

Class representing a transition of a Turing Machine.

3.18.2 Constructor & Destructor Documentation

3.18.2.1 TuringTransition::TuringTransition (StatePtr *from*, StatePtr *to*, const std::vector< char > & *read*, const std::vector< char > & *write*, Direction *dir*)

Constructor.

Parameters

from	A pointer to the TuringState where this transition is coming from
to	A pointer to the TuringState where this transition is going to
read	The read symbol(s) on the tape (in a vector)
write	The symbol(s) to be written to the tape (in a vector)
dir	Direction in which to move head

3.18.2.2 TuringTransition::∼TuringTransition ( ) [virtual]

Destructor

3.18.3 Member Function Documentation

3.18.3.1 std::tuple< StatePtr, std::vector< char >, Direction > TuringTransition::getTransition ( ) const

Gets next state, symbol(s) to write and direction.

Returns

Tuple of next state, symbol(s) to write and direction to move in

3.18.3.2 bool TuringTransition::isThisTransition (StatePtr *from,* StatePtr *to,* const std::vector< char > & *read,* const std::vector< char > & *write,* Direction *dir*) const

Checks if all of the given arguments are also the transition's.

Parameters

from	From pointer to check
to	To pointer to check
read	Read symbol(s) to check
write	Write symbol(s) to check
dir	Direction to check

Returns

True if the same

3.18.3.3 bool TuringTransition::match (StatePtr *state,* const std::vector< char > & *symbols*) const

Checks if transition is for given state and tape symbol.

Parameters

state	State from which transition should start
symbols	Symbol(s) that should be under head on tape

Returns

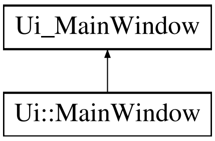
True if matching transition

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

- Turing.h
- Turing.cpp

3.19 Ui_MainWindow Class Reference

Inheritance diagram for Ui_MainWindow:



Public Member Functions

- void **setupUi** (QMainWindow ∗MainWindow)
- void **retranslateUi** (QMainWindow ∗MainWindow)

Public Attributes

- QWidget ∗ **centralWidget**
- QLabel ∗ **label**
- QPlainTextEdit ∗ **Input**
- QLabel ∗ **label_2**
- QComboBox ∗ **AnalyzeTypeSelect**
- QPushButton ∗ **VisualizeButton**
- QPushButton ∗ **AnalyzeButton**
- QMenuBar ∗ **menuBar**

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

- ui_mainwindow.h