**QCheck:**

**BoB question checker application**

**v. 001**

1. **Main Program Information** 
   1. Program identification
      1. Program Name
      2. Program Version
   2. Brief description of an application
      1. Task of application
      2. Main algorithm idea
2. **Program Functions**
3. **Program Layout**
   1. Used types
   2. Program structure: Modules, External libraries, Packages
   3. Compiling and linking
4. **Program Flow**
   1. Dialogs description
5. **Data flow**
6. **Data organization**
   1. Describe input data format: extended regex syntax , topic trees, abbreviation files.
   2. Output data format
7. **Testing**
8. **Main Program Information**
   1. **Program identification**
      1. **QCheck** (Question checker) GUI application.
      2. **Version 1.0**
   2. **Brief description of an application**
      1. **QCheck** is a tool with a GUI interface that aims to check matching of regular expression against a user question using one of language-dependant abbreviation file (English or German or Italian). At the end **QCheck** shows the user the “expanded” regular expression tree with all the abbreviations substituted.
      2. **QCheck** provides a user a GUI interface for abbreviation file specification, regular expression and the related question strings input. First, when the user specifies abbreviation file, application determines its language, preloads it and checks its validity with the help of BobHelper.java class from chatterbot project.

When regular expression and the related question both are specified by the user and button “check” is clicked, **QCheck** loads three “Bob ANTLR classes” from chatterbot project(Bob\_Parser, Bob\_Lexer and Bob\_TreeParser) and does regular expression validation (Bob\_Parser.java), builds and displays AST tree (regular expression expaned tree) and finally performs question matching against the tree (Bob\_TreeParser.java)

“Bob ANTLR classes” belongs to chatterbot project and are generated automaticaly from ANTLR parser grammar (ANother Tool for Language Recognition).

1. **Program functions**

**QCheck** allows the user to perform following actions:

* Abbreviation file loading dialog
* Checking regular expression input format between **macro-file format** (which supports only a basic Perl regular expression syntax) and **topic-tree format** (an extended regular expression, i.e. with Boolean connectivity operands (“**&&**”, “**!**”))

1. **Program Layout**
   1. Among the types that are used in **QCheck**, the most significant one are the “Bob ANTLR classes” (MyAST.java, BobHelper.java, Bob\_Parser.java, Bob\_Lexer.java and Bob\_TreeParser.java). They are provided by **chatterbot** project and implement the main logic.

**Classes:**

MyAST.java contains a modification abstract syntax tree, which is the output format for ANTLR parse. Modification adds to the basic set of parameters also line number information.

Bob\_Lexer.javaclass is automatically generated by **ANTLR**. It scans the extended regular expression string for the relevant tokens.

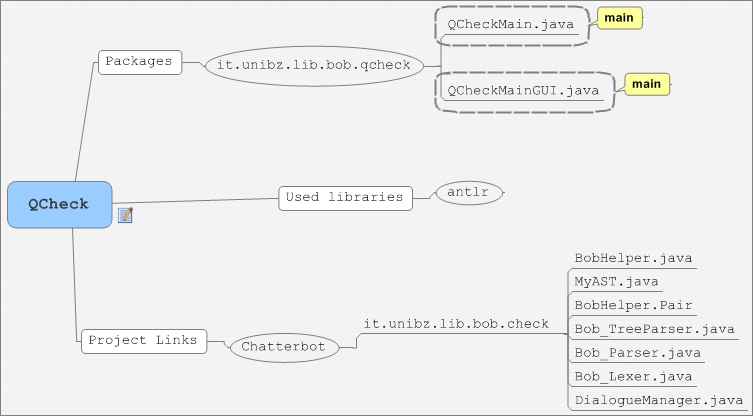
Bob\_Parser.java is automatically generated from Bob.g by ANTLR. It builds a parsing tree out of tokens, extracted by Lexer from regular expression string and thus validates Regular Expressions(bExpression() procedure that in case throws recognition exceptions).

Bob\_TreeParser.java is also generated from Bob.g. It evaluates whether some user question are matched by the tree that was built by the parser (bExpression(tree, stringToMatch) Boolean function). Algorithm works as following: it applies the required boolean logic and executs calls to regex.match() for the atomic (non-extended) regular expression patterns with the user question.

BobHelper.java defines the abbreviation file checking routine (by Boolean function checkAllRegexesInMacroMap).

ASTFrame ANTLR library class is used for regular expression tree (ASTTree) visualization (tree comes from Bob\_Parser getAST method).

* 1. **Program structure (see attached bob\_qcheck\_v002.jpeg)**

****

* + 1. **Packages and classes**

it.unibz.lib.bob.qcheck contains following classes:

* QCheckMainGUI.java – contains GUI description, action listeners for interaction with user and main checking logic with calls to external classes.
* QCheckMain.java – test main method. Can be easily deleted.
  + 1. **External JAR libraries:**

antlr.jar: extended Boolean regular expressions parser, regular expression tree (AST – Abstract Syntax Tree) classes, AST tree visualization frame.

* + 1. **Project dependences:**

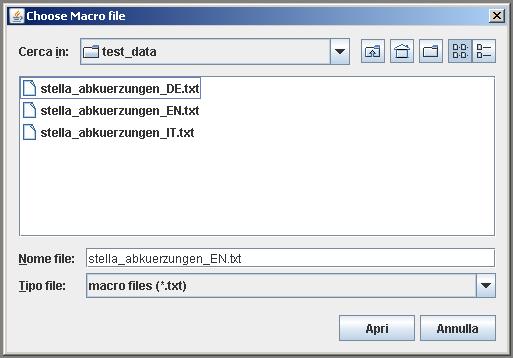
Chatterbot:Matchingregular expressions against question phrase, regular expression validation.

* 1. **Compiling and linking**

Linking: export as a single runnable jar file.

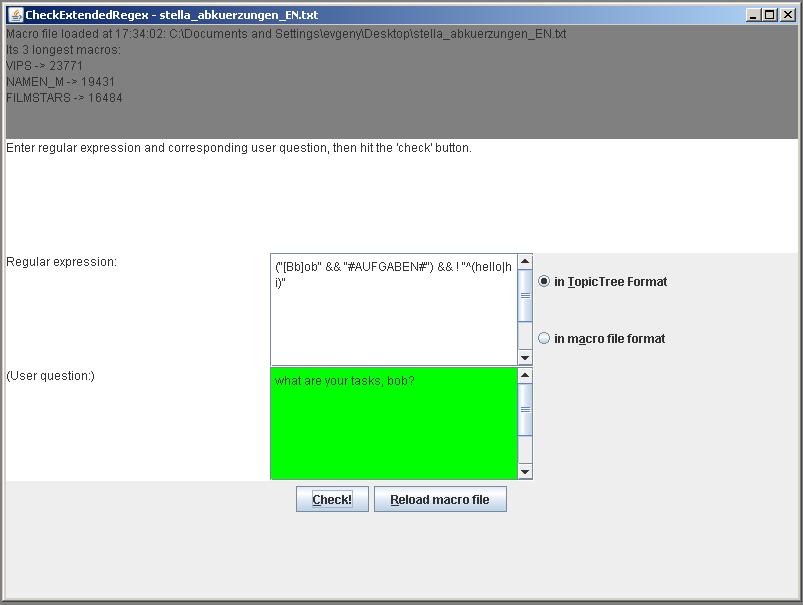
1. **Program Flow**
   1. **Dialogs description**

“Select abbrefiation file” dialog: user can select abbreviation file in one of 3 languages(German, English and Italian)

****

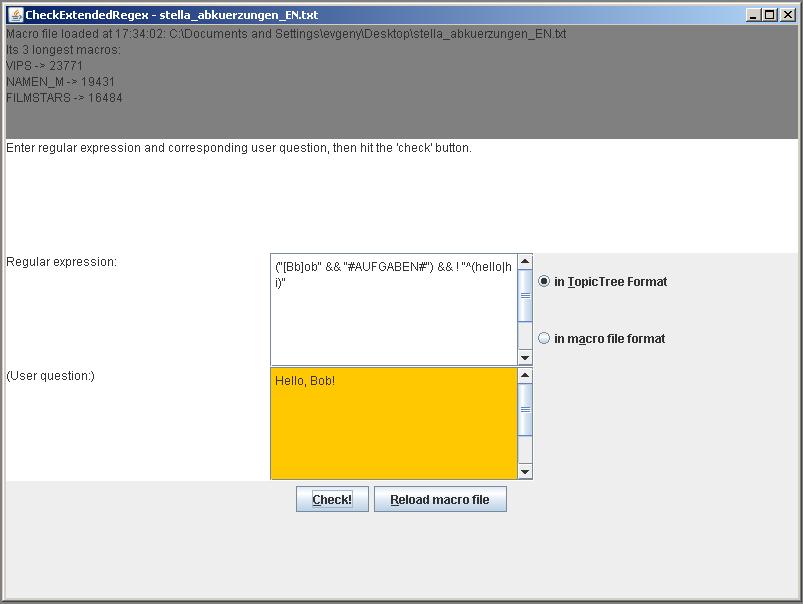
Select abbreviation file

“Main application window”: provides fields for regular expressions and user question. Contains area at the top for macro-file (abbreviations file) information and a frame under it for system messages (errors) output.

****

RE matches the question

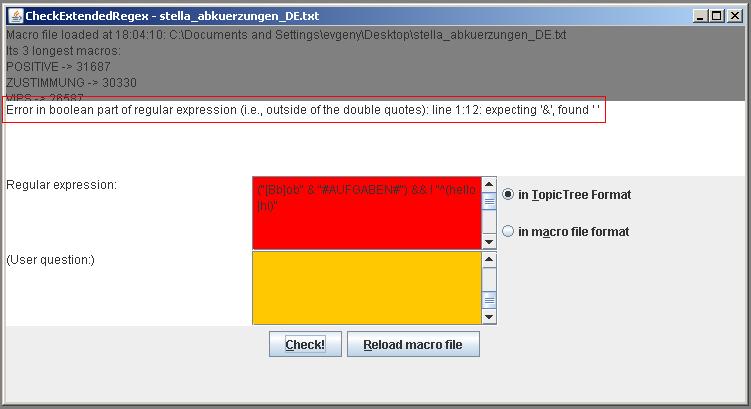
User question field can be 3 colors: white (no checking performed jet), green (matching succeeded) and orange (matching failed).



RE does not match the Question

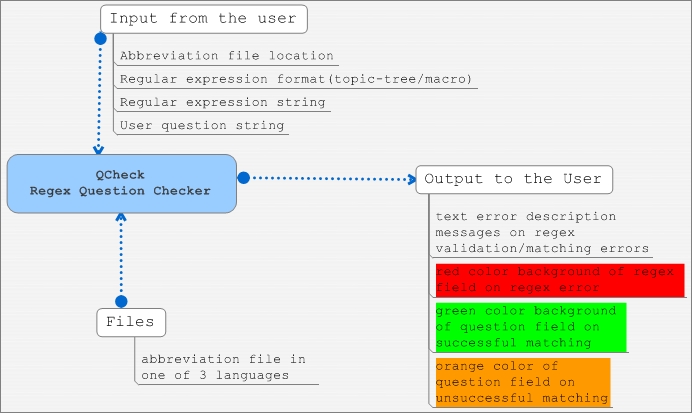
Regular expression field also might change the color depending on if regular expression validates or not (in this case it becomes red and in the message area (red rectangle) appears text with error description).

If otherwise, the RE you entered is correct and the system is able to parse it you will see an additional frame with an extended tree structure appearing (see the picture in the chapter 6.1).



Error in RE syntax

1. **Data Flow**

****

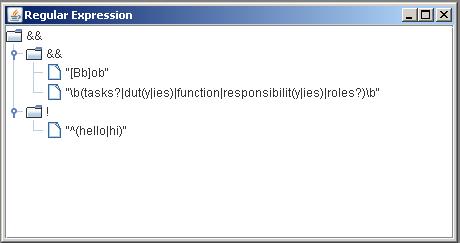
1. **Data organization**
   1. **Extended regular expressions structure**

Extended regular expression syntax is described in Bob.g file of chatterbot project (package it.unibz.lib.bob.check). It differs from the “ordinal” Perl regular expressions by introducing Boolean operators:

OR : "||"; AND: "&&"; NOT: "!"; And their parenthesis () grouping.

bob_qcheck_exregex.JPG

In this case will be matched all phrases that consist of 2 parts: first part should contain the words “Bob” or “bob” and the second one should match the “AUFGABEN” abbreviation (which looks like following “(tasks?|dut(y|ies)|function|responsibilit(y|ies)|roles?)"). And one more condition: the matched phrase should not start from words “hello” or “hi”. Order of parts is irrelevant, so both of “tasks bob?” and “bob tasks?” will be matched. The whole parsing tree structure of an extended regular expression could look like this:



Regular expression tree

* 1. **Abbreviation files structure**

It’s a text file with a list of regular expression’s abbreviations (one per row). Structure of abbreviation is following: ABBREVIATION\_NAME = “\b”+ +REGULAR\_EXPRESSION+”\b”, where “\b” a word boundary symbol used in RegEx. Example (for English version): bob_abbrev.JPG

In this case expressions that are matching this abbreviation will be:

“why”, “what fore”, “wherefore”, “for what reasons?”, “for which reasons?”

1. **Testing**

Make sure that you are using the standard English abbreviation file (actually it has only to contain the following line) bob_qcheck_abbrev.JPG

and topic-tree input format selected. Use this extended regular expression for test:

bob_qcheck_exregex.JPG

1. Positive test: What are your tasks, Bob?
2. Negative test: Hello bob, what are you your tasks?