



UNIVERSIDAD DE MÁLAGA

University of Malaga
School of Computer Science and Engineering

SyntaxTutor

Developer Manual

Version: 1.0.3
Date: 2025
Author: jose-rzm at GitHub



1 SyntaxTutor: An interactive Tool for Learning Syntax Analysis	1
1.1 Academic Context	1
1.2 Key Features	1
1.3 Interface Screenshots	2
1.3.1 Main Menu	2
1.3.2 LL(1) Learning Mode	3
1.3.3 SLR(1) Learning Mode	5
1.3.4 Assisted Mode: Guided Table Completion	8
1.4 Technologies Used	8
1.5 Downloads	8
1.6 Building from Source	9
1.6.1 Documentation	9
2 Changelog	11
2.1 [1.0.2] - 2025-07-16	11
2.1.1 Added	11
2.1.2 Fixed	11
2.2 [1.0.1] - 2025-06-17	11
2.2.1 Added	11
2.2.2 Fixed	11
2.2.3 Quality	11
2.3 [1.0.0] - 2025-06-15	12
2.3.1 Initial Release	12
3 Hierarchical Index	13
3.1 Class Hierarchy	13
4 Class Index	15
4.1 Class List	15
5 File Index	17
5.1 File List	17
6 Class Documentation	19
6.1 CenterAlignDelegate Class Reference	19
6.1.1 Member Function Documentation	19
6.1.1.1 initStyleOption() [1/2]	19
6.1.1.2 initStyleOption() [2/2]	20
6.2 CustomTextEdit Class Reference	20
6.2.1 Constructor & Destructor Documentation	20
6.2.1.1 CustomTextEdit()	20
6.2.2 Member Function Documentation	21
6.2.2.1 keyPressEvent()	21
6.2.2.2 sendRequested	21
6.3 GrammarFactory::FactoryItem Struct Reference	21

6.3.1 Detailed Description	22
6.3.2 Constructor & Destructor Documentation	22
6.3.2.1 FactoryItem()	22
6.3.3 Member Data Documentation	22
6.3.3.1 g_	22
6.3.3.2 st_	22
6.4 Grammar Struct Reference	23
6.4.1 Detailed Description	23
6.4.2 Constructor & Destructor Documentation	24
6.4.2.1 Grammar() [1/2]	24
6.4.2.2 Grammar() [2/2]	24
6.4.3 Member Function Documentation	24
6.4.3.1 AddProduction()	24
6.4.3.2 Debug()	24
6.4.3.3 FilterRulesByConsequent()	24
6.4.3.4 HasEmptyProduction()	24
6.4.3.5 SetAxiom()	25
6.4.3.6 Split()	25
6.4.4 Member Data Documentation	25
6.4.4.1 axiom_	25
6.4.4.2 g_	25
6.4.4.3 st_	25
6.5 GrammarFactory Struct Reference	26
6.5.1 Detailed Description	27
6.5.2 Member Function Documentation	27
6.5.2.1 AdjustTerminals()	27
6.5.2.2 CreateLv2Item()	28
6.5.2.3 GenerateNewNonTerminal()	28
6.5.2.4 GenLL1Grammar()	29
6.5.2.5 GenSLR1Grammar()	29
6.5.2.6 HasCycle()	30
6.5.2.7 HasDirectLeftRecursion()	30
6.5.2.8 HasIndirectLeftRecursion()	31
6.5.2.9 HasUnreachableSymbols()	31
6.5.2.10 Init()	32
6.5.2.11 IsInfinite()	32
6.5.2.12 LeftFactorize()	33
6.5.2.13 LongestCommonPrefix()	34
6.5.2.14 Lv1()	35
6.5.2.15 Lv2()	35
6.5.2.16 Lv3()	35
6.5.2.17 Lv4()	36
6.5.2.18 Lv5()	37

6.5.2.19 Lv6()	37
6.5.2.20 Lv7()	38
6.5.2.21 Merge()	38
6.5.2.22 NormalizeNonTerminals()	39
6.5.2.23 NullableSymbols()	40
6.5.2.24 PickOne()	40
6.5.2.25 RemoveLeftRecursion()	41
6.5.2.26 StartsWith()	42
6.5.3 Member Data Documentation	42
6.5.3.1 items	42
6.5.3.2 non_terminal_alphabet_	42
6.5.3.3 terminal_alphabet_	42
6.6 LL1Parser Class Reference	42
6.6.1 Constructor & Destructor Documentation	44
6.6.1.1 LL1Parser() [1/2]	44
6.6.1.2 LL1Parser() [2/2]	44
6.6.2 Member Function Documentation	44
6.6.2.1 ComputeFirstSets()	44
6.6.2.2 ComputeFollowSets()	45
6.6.2.3 CreateLL1Table()	45
6.6.2.4 First()	46
6.6.2.5 Follow()	47
6.6.2.6 PredictionSymbols()	48
6.6.3 Member Data Documentation	48
6.6.3.1 first_sets_	48
6.6.3.2 follow_sets_	49
6.6.3.3 gr_	49
6.6.3.4 ll1_t_	49
6.7 LLTableDialog Class Reference	49
6.7.1 Detailed Description	50
6.7.2 Constructor & Destructor Documentation	50
6.7.2.1 LLTableDialog()	50
6.7.3 Member Function Documentation	50
6.7.3.1 getTableData()	50
6.7.3.2 highlightIncorrectCells()	50
6.7.3.3 setInitialData()	51
6.7.3.4 submitted	51
6.8 LLTutorWindow Class Reference	51
6.8.1 Detailed Description	54
6.8.2 Constructor & Destructor Documentation	54
6.8.2.1 LLTutorWindow()	54
6.8.2.2 ~LLTutorWindow()	55
6.8.3 Member Function Documentation	55

6.8.3.1 addMessage()	55
6.8.3.2 addWidgetMessage()	55
6.8.3.3 animateLabelColor()	55
6.8.3.4 animateLabelPop()	56
6.8.3.5 buildTreeNode()	56
6.8.3.6 closeEvent()	56
6.8.3.7 computeSubtreeWidth()	56
6.8.3.8 drawTree()	56
6.8.3.9 eventFilter()	56
6.8.3.10 exportConversationToPdf()	56
6.8.3.11 feedback()	56
6.8.3.12 feedbackForA()	57
6.8.3.13 feedbackForA1()	57
6.8.3.14 feedbackForA2()	57
6.8.3.15 feedbackForAPrime()	57
6.8.3.16 feedbackForB()	57
6.8.3.17 feedbackForB1()	57
6.8.3.18 feedbackForB1TreeGraphics()	57
6.8.3.19 feedbackForB1TreeWidget()	57
6.8.3.20 feedbackForB2()	58
6.8.3.21 feedbackForBPrime()	58
6.8.3.22 feedbackForC()	58
6.8.3.23 feedbackForCPrime()	58
6.8.3.24 FormatGrammar()	58
6.8.3.25 generateQuestion()	58
6.8.3.26 handleTableSubmission()	59
6.8.3.27 markLastUserIncorrect()	59
6.8.3.28 sessionFinished	59
6.8.3.29 showTable()	59
6.8.3.30 showTableForCPrime()	60
6.8.3.31 showTreeGraphics()	60
6.8.3.32 solution()	60
6.8.3.33 solutionForA()	60
6.8.3.34 solutionForA1()	60
6.8.3.35 solutionForA2()	60
6.8.3.36 solutionForB()	60
6.8.3.37 solutionForB1()	60
6.8.3.38 solutionForB2()	60
6.8.3.39 TeachFirstTree()	61
6.8.3.40 TeachFollow()	61
6.8.3.41 TeachLL1Table()	61
6.8.3.42 TeachPredictionSymbols()	61
6.8.3.43 updateProgressPanel()	61

6.8.3.44 updateState()	61
6.8.3.45 verifyResponse()	62
6.8.3.46 verifyResponseForA()	62
6.8.3.47 verifyResponseForA1()	62
6.8.3.48 verifyResponseForA2()	62
6.8.3.49 verifyResponseForB()	62
6.8.3.50 verifyResponseForB1()	62
6.8.3.51 verifyResponseForB2()	62
6.8.3.52 verifyResponseForC()	62
6.8.3.53 wrongAnimation()	62
6.8.3.54 wrongUserResponseAnimation()	62
6.9 Lr0Item Struct Reference	63
6.9.1 Detailed Description	63
6.9.2 Constructor & Destructor Documentation	63
6.9.2.1 Lr0Item() [1/2]	63
6.9.2.2 Lr0Item() [2/2]	64
6.9.3 Member Function Documentation	64
6.9.3.1 AdvanceDot()	64
6.9.3.2 IsComplete()	65
6.9.3.3 NextToDoit()	65
6.9.3.4 operator==()	65
6.9.3.5 PrintItem()	66
6.9.3.6 ToString()	66
6.9.4 Member Data Documentation	66
6.9.4.1 antecedent_	66
6.9.4.2 consequent_	66
6.9.4.3 dot_	66
6.9.4.4 eol_	67
6.9.4.5 epsilon_	67
6.10 MainWindow Class Reference	67
6.10.1 Detailed Description	68
6.10.2 Constructor & Destructor Documentation	68
6.10.2.1 MainWindow()	68
6.10.2.2 ~MainWindow()	69
6.10.3 Member Function Documentation	69
6.10.3.1 setUserLevel()	69
6.10.3.2 thresholdFor()	69
6.10.3.3 userLevel()	69
6.10.3.4 userLevelChanged	69
6.10.3.5 userLevelUp	70
6.10.4 Property Documentation	70
6.10.4.1 userLevel	70
6.11 SLR1Parser::s_action Struct Reference	70

6.11.1 Member Data Documentation	71
6.11.1.1 action	71
6.11.1.2 item	71
6.12 SLR1Parser Class Reference	71
6.12.1 Detailed Description	73
6.12.2 Member Typedef Documentation	73
6.12.2.1 action_table	73
6.12.2.2 transition_table	73
6.12.3 Member Enumeration Documentation	74
6.12.3.1 Action	74
6.12.4 Constructor & Destructor Documentation	74
6.12.4.1 SLR1Parser() [1/2]	74
6.12.4.2 SLR1Parser() [2/2]	74
6.12.5 Member Function Documentation	74
6.12.5.1 AllItems()	74
6.12.5.2 Closure()	75
6.12.5.3 ClosureUtil()	75
6.12.5.4 ComputeFirstSets()	76
6.12.5.5 ComputeFollowSets()	76
6.12.5.6 Delta()	77
6.12.5.7 First()	78
6.12.5.8 Follow()	79
6.12.5.9 MakeInitialState()	80
6.12.5.10 MakeParser()	80
6.12.5.11 PrintItems()	81
6.12.5.12 SolveLRConflicts()	81
6.12.6 Member Data Documentation	82
6.12.6.1 actions_	82
6.12.6.2 first_sets_	82
6.12.6.3 follow_sets_	82
6.12.6.4 gr_	82
6.12.6.5 states_	82
6.12.6.6 transitions_	82
6.13 SLRTableDialog Class Reference	83
6.13.1 Detailed Description	83
6.13.2 Constructor & Destructor Documentation	83
6.13.2.1 SLRTableDialog()	83
6.13.3 Member Function Documentation	84
6.13.3.1 getTableData()	84
6.13.3.2 setInitialData()	84
6.14 SLRTutorWindow Class Reference	84
6.14.1 Detailed Description	87
6.14.2 Constructor & Destructor Documentation	87

6.14.2.1 SLRTutorWindow()	87
6.14.2.2 ~SLRTutorWindow()	88
6.14.3 Member Function Documentation	88
6.14.3.1 addMessage()	88
6.14.3.2 addUserState()	88
6.14.3.3 addUserTransition()	88
6.14.3.4 animateLabelColor()	88
6.14.3.5 animateLabelPop()	88
6.14.3.6 closeEvent()	89
6.14.3.7 exportConversationToPdf()	89
6.14.3.8 feedback()	89
6.14.3.9 feedbackForA()	89
6.14.3.10 feedbackForA1()	89
6.14.3.11 feedbackForA2()	89
6.14.3.12 feedbackForA3()	89
6.14.3.13 feedbackForA4()	89
6.14.3.14 feedbackForAPrime()	89
6.14.3.15 feedbackForB()	90
6.14.3.16 feedbackForB1()	90
6.14.3.17 feedbackForB2()	90
6.14.3.18 feedbackForBPrime()	90
6.14.3.19 feedbackForC()	90
6.14.3.20 feedbackForCA()	90
6.14.3.21 feedbackForCB()	90
6.14.3.22 feedbackForD()	90
6.14.3.23 feedbackForD1()	90
6.14.3.24 feedbackForD2()	90
6.14.3.25 feedbackForDPrime()	90
6.14.3.26 feedbackForE()	90
6.14.3.27 feedbackForE1()	90
6.14.3.28 feedbackForE2()	90
6.14.3.29 feedbackForF()	90
6.14.3.30 feedbackForFA()	90
6.14.3.31 feedbackForG()	91
6.14.3.32 fillSortedGrammar()	91
6.14.3.33 FormatGrammar()	91
6.14.3.34 generateQuestion()	91
6.14.3.35 launchSLRWizard()	91
6.14.3.36 markLastUserIncorrect()	91
6.14.3.37 sessionFinished	91
6.14.3.38 showTable()	91
6.14.3.39 solution()	92
6.14.3.40 solutionForA()	92

6.14.3.41 solutionForA1()	92
6.14.3.42 solutionForA2()	92
6.14.3.43 solutionForA3()	92
6.14.3.44 solutionForA4()	92
6.14.3.45 solutionForB()	92
6.14.3.46 solutionForC()	92
6.14.3.47 solutionForCA()	93
6.14.3.48 solutionForCB()	93
6.14.3.49 solutionForD()	93
6.14.3.50 solutionForD1()	93
6.14.3.51 solutionForD2()	93
6.14.3.52 solutionForE()	93
6.14.3.53 solutionForE1()	93
6.14.3.54 solutionForE2()	94
6.14.3.55 solutionForF()	94
6.14.3.56 solutionForFA()	94
6.14.3.57 solutionForG()	94
6.14.3.58 TeachClosure()	94
6.14.3.59 TeachClosureStep()	94
6.14.3.60 TeachDeltaFunction()	94
6.14.3.61 updateProgressPanel()	94
6.14.3.62 updateState()	94
6.14.3.63 verifyResponse()	95
6.14.3.64 verifyResponseForA()	95
6.14.3.65 verifyResponseForA1()	95
6.14.3.66 verifyResponseForA2()	95
6.14.3.67 verifyResponseForA3()	95
6.14.3.68 verifyResponseForA4()	95
6.14.3.69 verifyResponseForB()	95
6.14.3.70 verifyResponseForC()	95
6.14.3.71 verifyResponseForCA()	95
6.14.3.72 verifyResponseForCB()	96
6.14.3.73 verifyResponseForD()	96
6.14.3.74 verifyResponseForD1()	96
6.14.3.75 verifyResponseForD2()	96
6.14.3.76 verifyResponseForE()	96
6.14.3.77 verifyResponseForE1()	96
6.14.3.78 verifyResponseForE2()	96
6.14.3.79 verifyResponseForF()	96
6.14.3.80 verifyResponseForFA()	97
6.14.3.81 verifyResponseForG()	97
6.14.3.82 verifyResponseForH()	97
6.14.3.83 wrongAnimation()	97

6.14.3.84 wrongUserResponseAnimation()	97
6.15 SLRWizard Class Reference	97
6.15.1 Detailed Description	98
6.15.2 Constructor & Destructor Documentation	98
6.15.2.1 SLRWizard()	98
6.15.3 Member Function Documentation	99
6.15.3.1 stdVectorToQVector()	99
6.16 SLRWizardPage Class Reference	99
6.16.1 Detailed Description	100
6.16.2 Constructor & Destructor Documentation	100
6.16.2.1 SLRWizardPage()	100
6.17 state Struct Reference	100
6.17.1 Detailed Description	101
6.17.2 Member Function Documentation	101
6.17.2.1 operator==()	101
6.17.3 Member Data Documentation	101
6.17.3.1 id_	101
6.17.3.2 items_	101
6.18 SymbolTable Struct Reference	101
6.18.1 Detailed Description	102
6.18.2 Member Function Documentation	102
6.18.2.1 In()	102
6.18.2.2 IsTerminal()	102
6.18.2.3 IsTerminalWthoEol()	103
6.18.2.4 PutSymbol()	103
6.18.3 Member Data Documentation	104
6.18.3.1 EOL_	104
6.18.3.2 EPSILON_	104
6.18.3.3 non_terminals_	104
6.18.3.4 st_	104
6.18.3.5 terminals_	104
6.18.3.6 terminals_wtho_eol_	104
6.19 LLTutorWindow::TreeNode Struct Reference	104
6.19.1 Detailed Description	104
6.19.2 Member Data Documentation	104
6.19.2.1 children	104
6.19.2.2 label	105
6.20 TutorialManager Class Reference	105
6.20.1 Detailed Description	106
6.20.2 Constructor & Destructor Documentation	106
6.20.2.1 TutorialManager()	106
6.20.3 Member Function Documentation	106
6.20.3.1 addStep()	106

6.20.3.2 clearSteps()	107
6.20.3.3 eventFilter()	107
6.20.3.4 finishLL1()	107
6.20.3.5 finishSLR1()	107
6.20.3.6 hideOverlay()	107
6.20.3.7 ll1Finished	108
6.20.3.8 nextStep	108
6.20.3.9 setRootWindow()	109
6.20.3.10 slr1Finished	109
6.20.3.11 start()	110
6.20.3.12 stepStarted	110
6.20.3.13 tutorialFinished	110
6.21 TutorialStep Struct Reference	111
6.21.1 Detailed Description	111
6.21.2 Member Data Documentation	111
6.21.2.1 htmlText	111
6.21.2.2 target	111
6.22 UniqueQueue< T > Class Template Reference	111
6.22.1 Detailed Description	112
6.22.2 Member Function Documentation	112
6.22.2.1 clear()	112
6.22.2.2 empty()	112
6.22.2.3 front()	112
6.22.2.4 pop()	112
6.22.2.5 push()	112
7 File Documentation	115
7.1 backend/grammar.cpp File Reference	115
7.2 backend/grammar.hpp File Reference	115
7.2.1 Typedef Documentation	116
7.2.1.1 production	116
7.3 grammar.hpp	116
7.4 backend/grammar_factory.cpp File Reference	117
7.5 backend/grammar_factory.hpp File Reference	118
7.6 grammar_factory.hpp	118
7.7 backend/ll1_parser.cpp File Reference	120
7.8 backend/ll1_parser.hpp File Reference	120
7.9 ll1_parser.hpp	121
7.10 backend/lr0_item.cpp File Reference	122
7.11 backend/lr0_item.hpp File Reference	122
7.12 lr0_item.hpp	123
7.13 backend/slr1_parser.cpp File Reference	124
7.14 backend/slr1_parser.hpp File Reference	124

7.15 slr1_parser.hpp	125
7.16 backend/state.hpp File Reference	126
7.17 state.hpp	127
7.18 backend/symbol_table.cpp File Reference	128
7.19 backend/symbol_table.hpp File Reference	128
7.19.1 Enumeration Type Documentation	129
7.19.1.1 symbol_type	129
7.20 symbol_table.hpp	129
7.21 CHANGELOG.md File Reference	130
7.22 customtextedit.cpp File Reference	130
7.23 customtextedit.h File Reference	131
7.24 customtextedit.h	132
7.25 lltabledialog.cpp File Reference	132
7.26 lltabledialog.h File Reference	132
7.27 lltabledialog.h	133
7.28 lltutorwindow.cpp File Reference	134
7.29 lltutorwindow.h File Reference	134
7.29.1 Enumeration Type Documentation	135
7.29.1.1 State	135
7.30 lltutorwindow.h	136
7.31 main.cpp File Reference	139
7.31.1 Function Documentation	139
7.31.1.1 loadFonts()	139
7.31.1.2 main()	139
7.32 mainwindow.cpp File Reference	140
7.33 mainwindow.h File Reference	140
7.34 mainwindow.h	141
7.35 README.md File Reference	142
7.36 slrtabledialog.cpp File Reference	142
7.37 slrtabledialog.h File Reference	143
7.38 slrtabledialog.h	143
7.39 slrtutorwindow.cpp File Reference	144
7.40 slrtutorwindow.h File Reference	144
7.40.1 Enumeration Type Documentation	145
7.40.1.1 StateSlr	145
7.41 slrtutorwindow.h	146
7.42 slrwizard.h File Reference	150
7.43 slrwizard.h	150
7.44 slrwizardpage.h File Reference	152
7.45 slrwizardpage.h	153
7.46 tutorialmanager.cpp File Reference	154
7.47 tutorialmanager.h File Reference	154
7.48 tutorialmanager.h	155

7.49 UniqueQueue.h File Reference	156
7.50 UniqueQueue.h	157

Chapter 1

SyntaxTutor: An interactive Tool for Learning Syntax Analysis

SyntaxTutor is an educational application designed to help compiler students understand LL(1) and SLR(1) parsing algorithms. Through a visual and interactive interface, it guides users step-by-step through the computation of FIRST, FOLLOW, CLOSURE, GOTO, predictive parsing tables, and LR automata, offering real-time pedagogical feedback.

Rather than acting as a mere calculator, SyntaxTutor functions as a learning companion. It explains the reasoning behind each step, highlights common mistakes, and encourages students to engage with the theory behind the algorithms.

1.1 Academic Context

SyntaxTutor is part of a Final Degree Project (TFG) developed at the University of Málaga (UMA), in the Computer Engineering program. Its main goal is to offer an educational companion for students learning syntax analysis, going beyond traditional calculators by incorporating guided feedback, visualization, and gamified learning.

1.2 Key Features

- Educational Focus: built to teach, not just compute.
 - Visualization: derivation trees, intermediate steps, sets, and tables.
 - Exportable Results: useful for reports or coursework.
-

1.3 Interface Screenshots

1.3.1 Main Menu

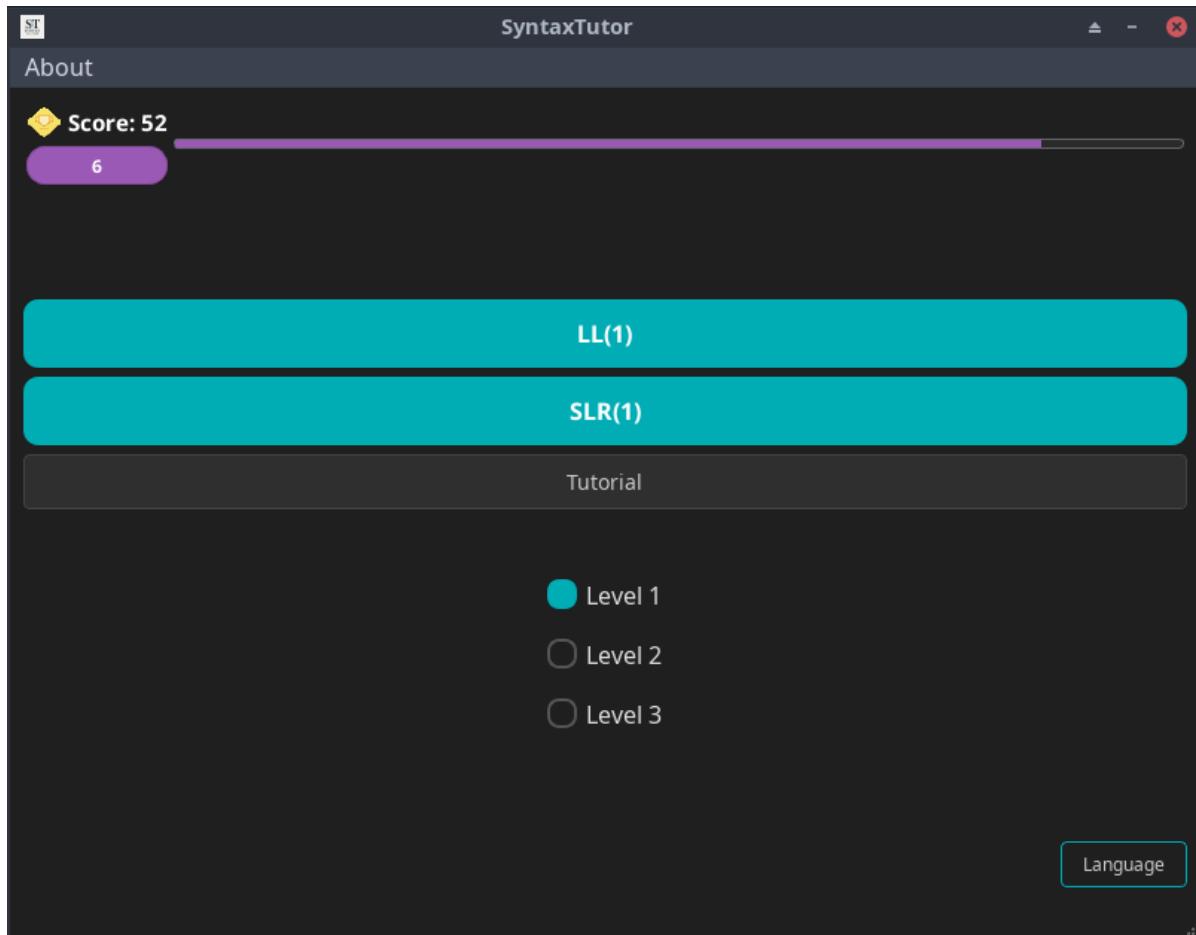


Figure 1.1 Main window

Home screen with gamification, levels, and language options.

1.3.2 LL(1) Learning Mode

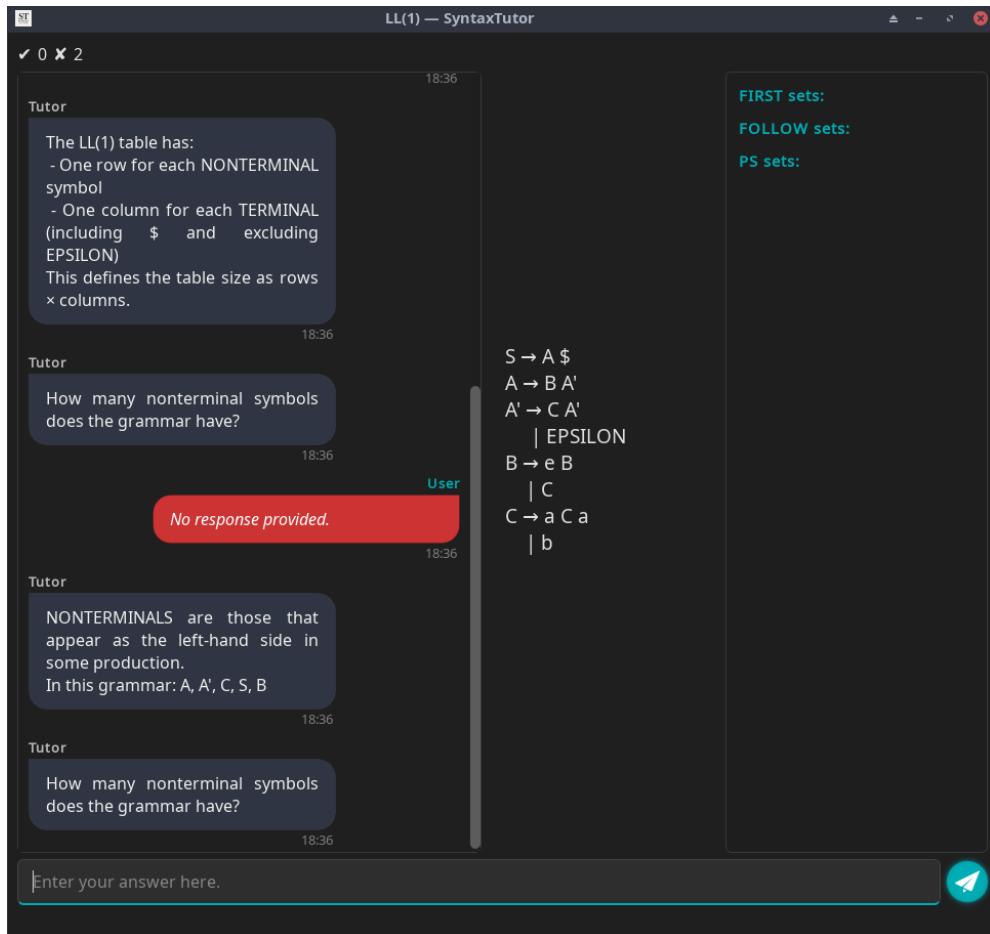


Figure 1.2 LL(1) dialog view

Interactive LL(1) tutor asks questions and provides feedback.

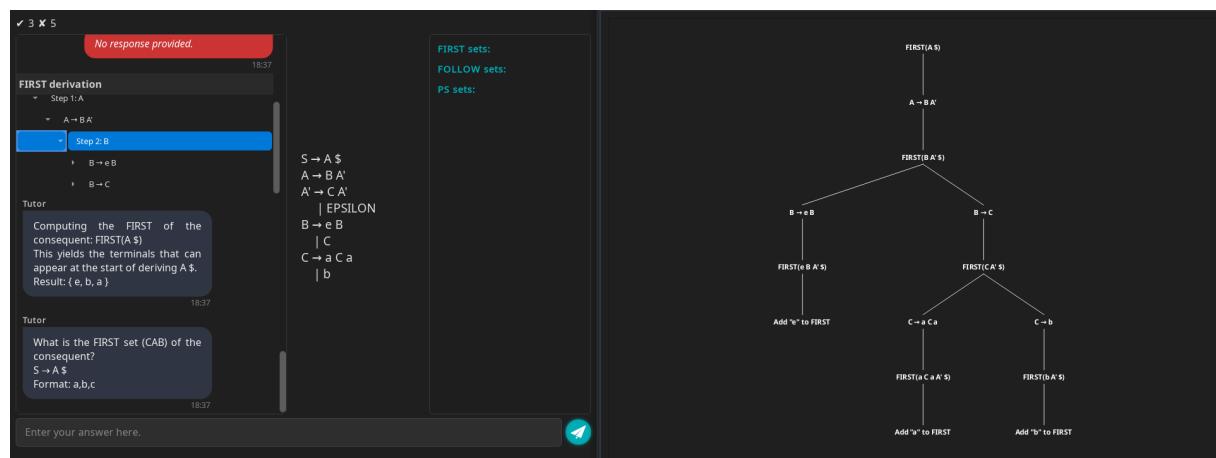


Figure 1.3 LL(1) derivation tree

Derivation tree view showing how FIRST sets are built step-by-step.

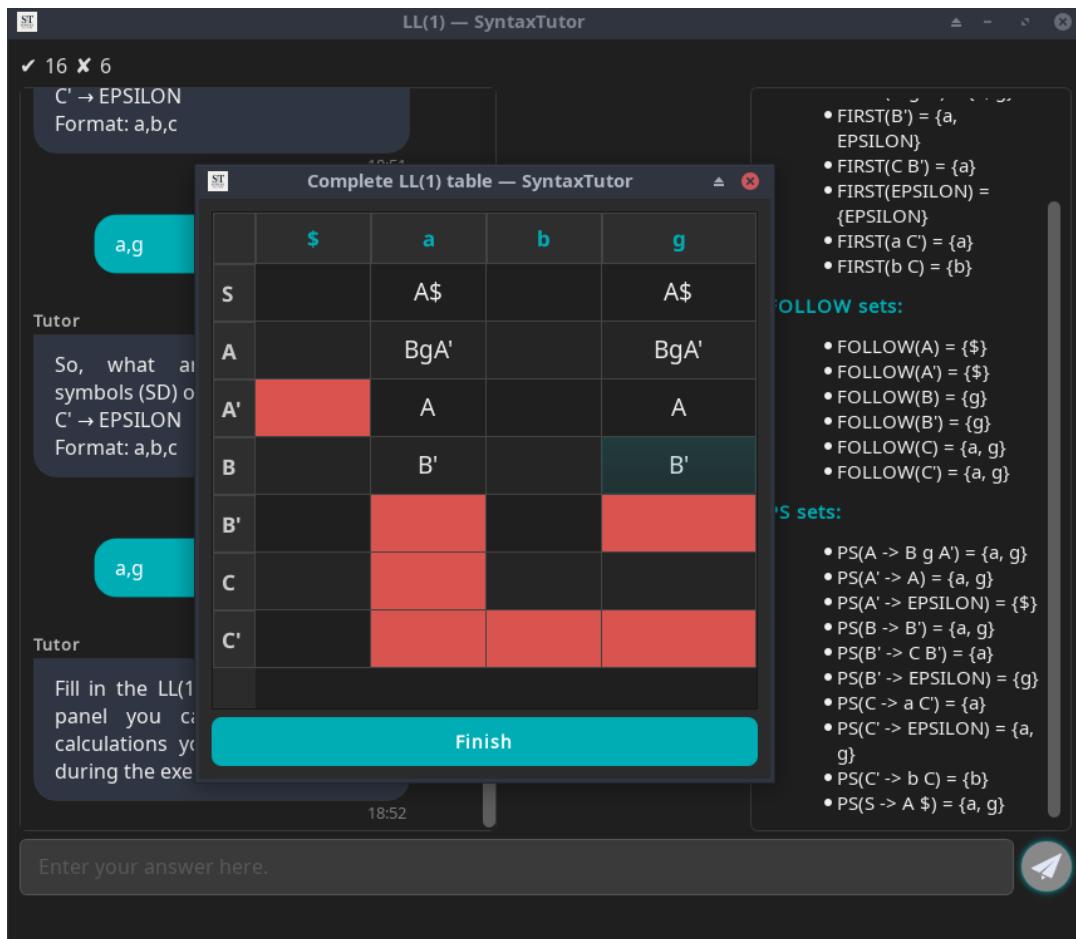


Figure 1.4 LL(1) table task

Completion of the LL(1) predictive table with visual guidance.

1.3.3 SLR(1) Learning Mode

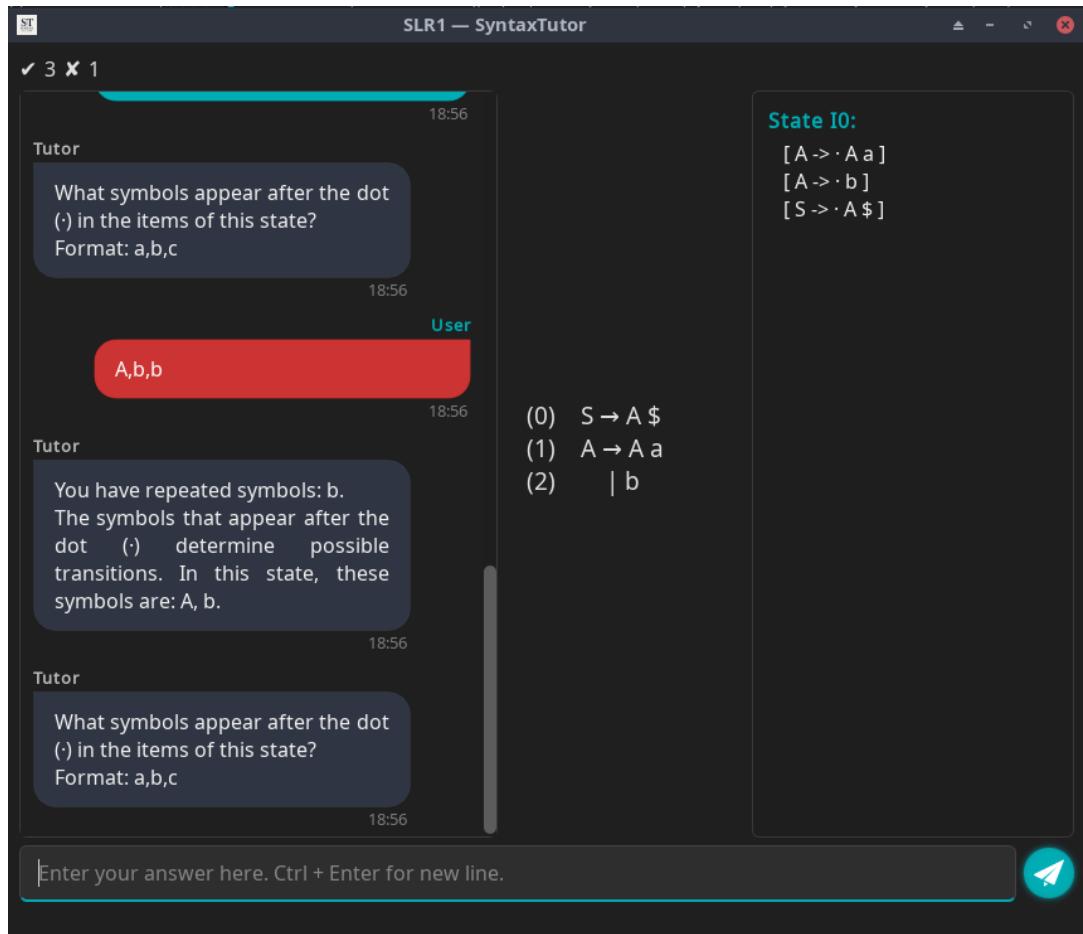


Figure 1.5 SLR(1) item view

User is asked to identify symbols after the dot in an LR(0) item.

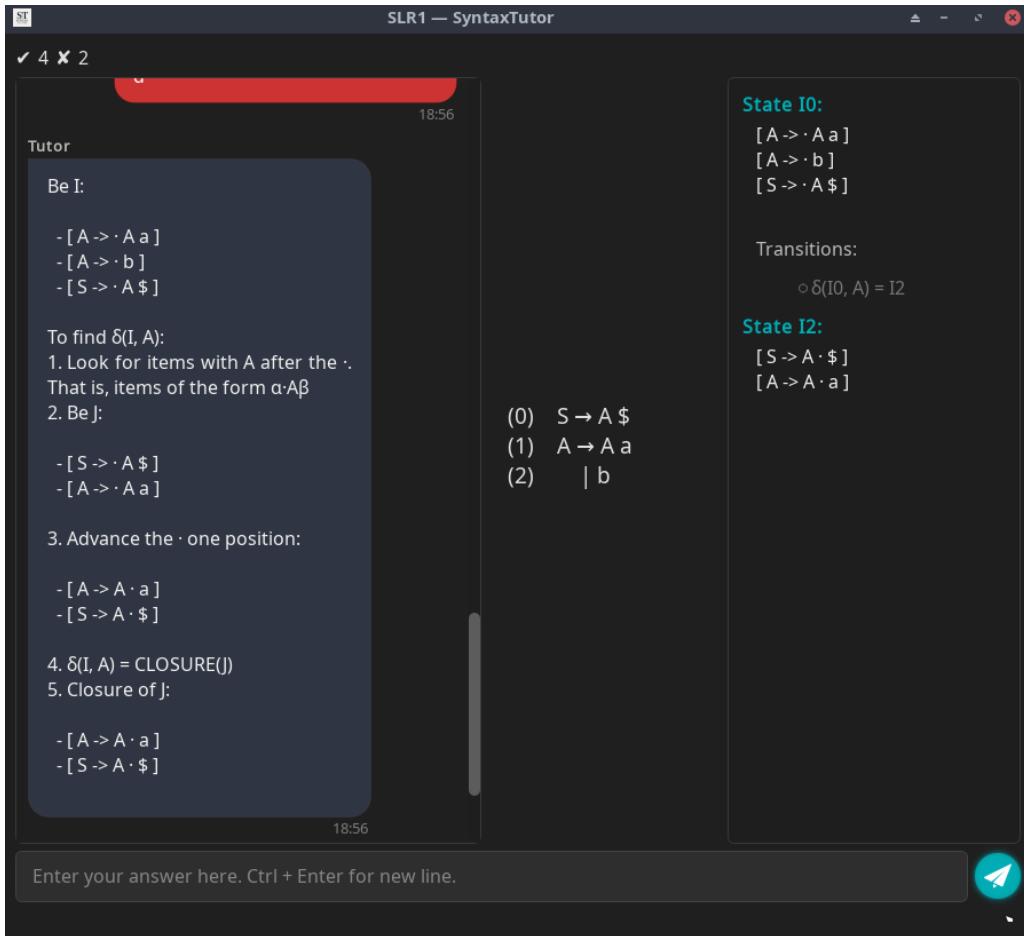


Figure 1.6 SLR(1) automaton construction

Step-by-step explanation of the GOTO/closure construction.

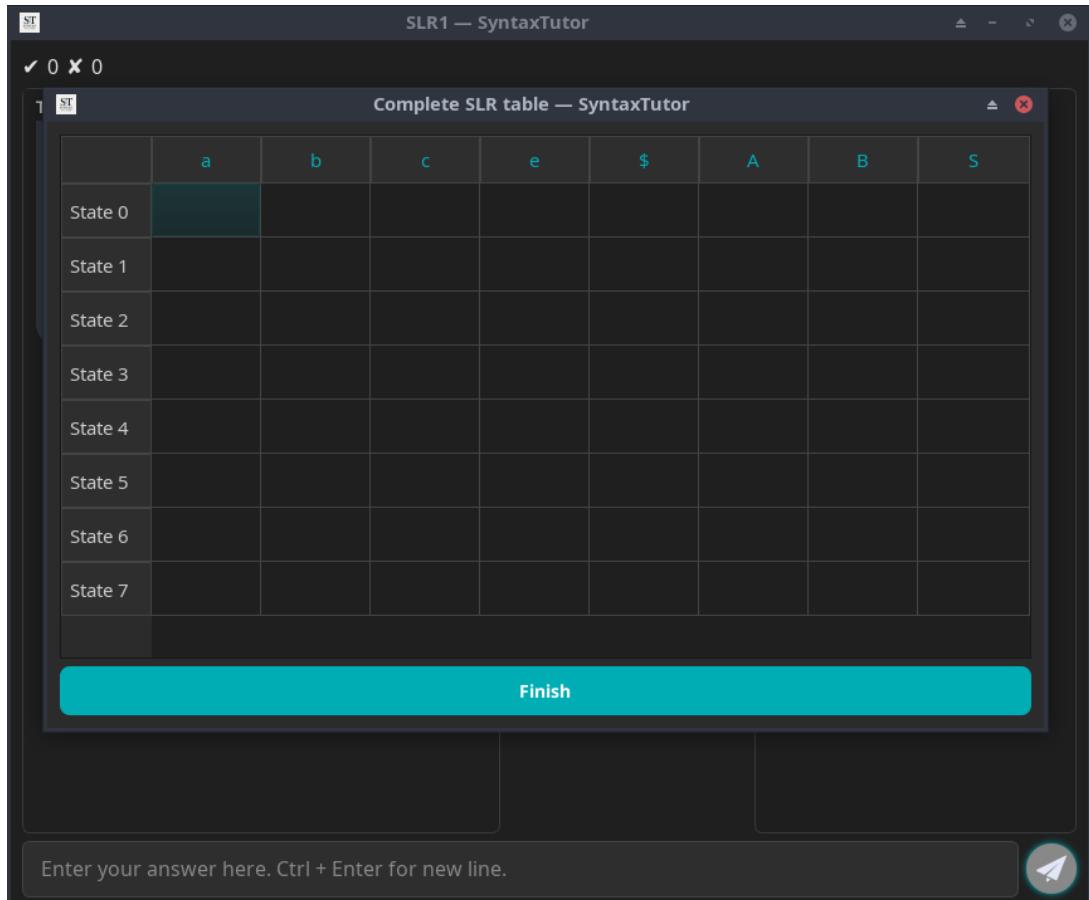


Figure 1.7 SLR(1) table fill-in

Interactive SLR(1) table to complete, with states and terminals/non-terminals.

1.3.4 Assisted Mode: Guided Table Completion

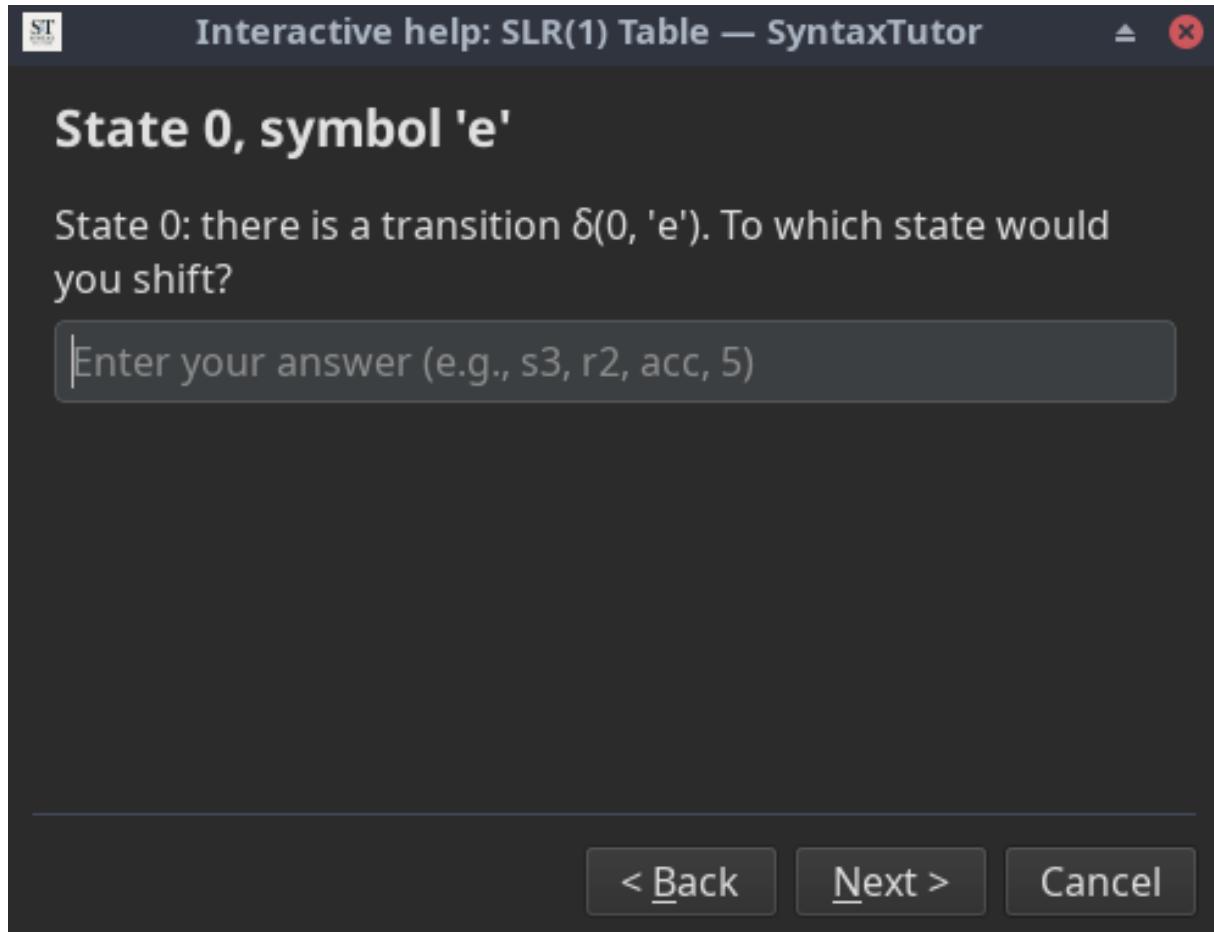


Figure 1.8 SLR(1) guided mode

SyntaxTutor walks the student through each cell in the parsing table with hints and context.

1.4 Technologies Used

- C++: efficient implementation of parsing algorithms
 - Qt6: modern, cross-platform graphical user interface.
 - Modular architecture: clean separation between logic and UI, designed for easy extensibility.
-

1.5 Downloads

Precompiled builds of SyntaxTutor are available in the Releases tab:

- Linux (X11): executable AppImage
- Windows: ZIP archive with the .exe
- macOS: .app bundles for both Apple Silicon (ARM) and Intel

Warning

The Windows and macOS versions are not digitally signed. Your operating system may display a warning when running the application. You can bypass it manually if you trust the source.

1.6 Building from Source

To build SyntaxTutor from source, you just need:

- Qt6 (including qmake6)
 - A C++20-compliant compiler
- ```
qmake6
make
```
- 

### 1.6.1 Documentation

Full documentation for the source code is available via Doxygen:

- Online HTML Documentation: <https://jose-rzm.github.io/SyntaxTutor/>
- PDF Reference Manual: [refman.pdf](#) (in the docs/latex/ folder)

The documentation includes:

- Detailed class and function reference
- Graphs of dependencies and inheritance
- Descriptions of parsing algorithms and internal logic

To regenerate it locally, install **Doxygen** and run:

```
doxygen
```

This will update the contents of the docs/ folder with both HTML and LaTeX output.



# Chapter 2

## Changelog

All notable changes to this project will be documented in this file.

### 2.1 [1.0.3] - 2025-11-04

Minor usability improvements in user input handling.

#### 2.1.1 Added

- Allow the user to write A -> . instead of A -> EPSILON .

#### 2.1.2 Fixed

- Accept input containing spaces in responses such as x, y

### 2.2 [1.0.2] - 2025-07-16

#### 2.2.1 Added

- User manual in Spanish (manual/SyntaxTutor-Manual-ES.pdf)
- User manual in English (manual/SyntaxTutor-Manual-EN.pdf)
- Developer manual (manual/SyntaxTutor-Developer-Manual.pdf)
- Script to customize titlepage (manual/patch\_refman\_title.sh)

#### 2.2.2 Fixed

- Fixed issue when exporting PDF in SLR mode.
- Fixed some feedback in SLR mode

### 2.3 [1.0.1] - 2025-06-17

#### 2.3.1 Added

- Added Doxyfile for automatic documentation generation with Doxygen.
- Completed missing translations for multilingual support (English/Spanish).

#### 2.3.2 Fixed

- Corrected a typo in the SLR(1) Quick Reference view.
- EPSILON is no longer shown when exporting LL(1) parse tables to PDF.
- Improved feedback message for the FA question in the SLR module.

### 2.3.3 Quality

- All changes successfully passed CI (GitHub Actions).
- Test suite: 158 tests passed (100% success rate).
- Maintained high test coverage across modules (most above 90%).

## 2.4 [1.0.0] - 2025-06-15

### 2.4.1 Initial Release

- First public version of SyntaxTutor.
- Includes LL(1) and SLR(1) modules with guided exercises.
- Features interactive tutoring, automatic grammar generation, feedback system, and performance tracking.

# Chapter 3

## Hierarchical Index

### 3.1 Class Hierarchy

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

|                                       |     |
|---------------------------------------|-----|
| GrammarFactory::FactoryItem . . . . . | 21  |
| Grammar . . . . .                     | 23  |
| GrammarFactory . . . . .              | 26  |
| LL1Parser . . . . .                   | 42  |
| Lr0Item . . . . .                     | 63  |
| QDialog                               |     |
| LLTableDialog . . . . .               | 49  |
| SLRTableDialog . . . . .              | 83  |
| QMainWindow                           |     |
| LLTutorWindow . . . . .               | 51  |
| MainWindow . . . . .                  | 67  |
| SLRTutorWindow . . . . .              | 84  |
| QObject                               |     |
| TutorialManager . . . . .             | 105 |
| QStyledItemDelegate                   |     |
| CenterAlignDelegate . . . . .         | 19  |
| CenterAlignDelegate . . . . .         | 19  |
| QTextEdit                             |     |
| CustomTextEdit . . . . .              | 20  |
| QWizard                               |     |
| SLRWizard . . . . .                   | 97  |
| QWizardPage                           |     |
| SLRWizardPage . . . . .               | 99  |
| SLR1Parser::s_action . . . . .        | 70  |
| SLR1Parser . . . . .                  | 71  |
| state . . . . .                       | 100 |
| SymbolTable . . . . .                 | 101 |
| LLTutorWindow::TreeNode . . . . .     | 104 |
| TutorialStep . . . . .                | 111 |
| UniqueQueue< T > . . . . .            | 111 |



# Chapter 4

## Class Index

### 4.1 Class List

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

|                                             |                                                                                             |     |
|---------------------------------------------|---------------------------------------------------------------------------------------------|-----|
| <a href="#">CenterAlignDelegate</a>         | .....                                                                                       | 19  |
| <a href="#">CustomTextEdit</a>              | .....                                                                                       | 20  |
| <a href="#">GrammarFactory::FactoryItem</a> | Represents an individual grammar item with its associated symbol table                      | 21  |
| <a href="#">Grammar</a>                     | Represents a context-free grammar, including its rules, symbol table, and starting symbol   | 23  |
| <a href="#">GrammarFactory</a>              | Responsible for creating and managing grammar items and performing checks on grammars       | 26  |
| <a href="#">LL1Parser</a>                   | .....                                                                                       | 42  |
| <a href="#">LLTableDialog</a>               | Dialog for filling and submitting an LL(1) parsing table                                    | 49  |
| <a href="#">LLTutorWindow</a>               | Main window for the LL(1) interactive tutoring mode in SyntaxTutor                          | 51  |
| <a href="#">Lr0Item</a>                     | Represents an LR(0) item used in LR automata construction                                   | 63  |
| <a href="#">MainWindow</a>                  | Main application window of SyntaxTutor, managing levels, exercises, and UI state            | 67  |
| <a href="#">SLR1Parser::s_action</a>        | .....                                                                                       | 70  |
| <a href="#">SLR1Parser</a>                  | Implements an SLR(1) parser for context-free grammars                                       | 71  |
| <a href="#">SLRTableDialog</a>              | Dialog window for completing and submitting an SLR(1) parsing table                         | 83  |
| <a href="#">SLRTutorWindow</a>              | Main window for the SLR(1) interactive tutoring mode in SyntaxTutor                         | 84  |
| <a href="#">SLRWizard</a>                   | Interactive assistant that guides the student step-by-step through the SLR(1) parsing table | 97  |
| <a href="#">SLRWizardPage</a>               | A single step in the SLR(1) guided assistant for table construction                         | 99  |
| <a href="#">state</a>                       | Represents a state in the LR(0) automaton                                                   | 100 |
| <a href="#">SymbolTable</a>                 | Stores and manages grammar symbols, including their classification and special markers      | 101 |
| <a href="#">LLTutorWindow::TreeNode</a>     | TreeNode structure used to build derivation trees                                           | 104 |
| <a href="#">TutorialManager</a>             | Manages interactive tutorials by highlighting UI elements and guiding the user              | 105 |

|                                                         |           |                     |
|---------------------------------------------------------|-----------|---------------------|
| <a href="#">TutorialStep</a>                            |           |                     |
| Represents a single step in the tutorial sequence       | . . . . . | <a href="#">111</a> |
| <a href="#">UniqueQueue&lt; T &gt;</a>                  |           |                     |
| A queue that ensures each element is inserted only once | . . . . . | <a href="#">111</a> |

# Chapter 5

## File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

|                             |     |
|-----------------------------|-----|
| customtextedit.cpp          | 130 |
| customtextedit.h            | 131 |
| lltabledialog.cpp           | 132 |
| lltabledialog.h             | 132 |
| lltutorwindow.cpp           | 134 |
| lltutorwindow.h             | 134 |
| main.cpp                    | 139 |
| mainwindow.cpp              | 140 |
| mainwindow.h                | 140 |
| slrtabledialog.cpp          | 142 |
| slrtabledialog.h            | 143 |
| slrtutorwindow.cpp          | 144 |
| slrtutorwindow.h            | 144 |
| slrwizard.h                 | 150 |
| slrwizardpage.h             | 152 |
| tutorialmanager.cpp         | 154 |
| tutorialmanager.h           | 154 |
| UniqueQueue.h               | 156 |
| backend/grammar.cpp         | 115 |
| backend/grammar.hpp         | 115 |
| backend/grammar_factory.cpp | 117 |
| backend/grammar_factory.hpp | 118 |
| backend/ll1_parser.cpp      | 120 |
| backend/ll1_parser.hpp      | 120 |
| backend/lr0_item.cpp        | 122 |
| backend/lr0_item.hpp        | 122 |
| backend/slr1_parser.cpp     | 124 |
| backend/slr1_parser.hpp     | 124 |
| backend/state.hpp           | 126 |
| backend/symbol_table.cpp    | 128 |
| backend/symbol_table.hpp    | 128 |

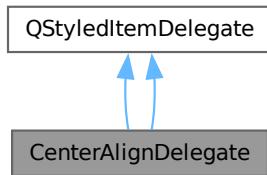


# Chapter 6

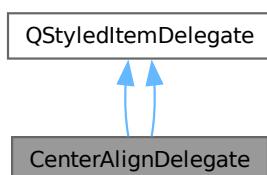
## Class Documentation

### 6.1 CenterAlignDelegate Class Reference

Inheritance diagram for CenterAlignDelegate:



Collaboration diagram for CenterAlignDelegate:



#### Public Member Functions

- void [initStyleOption](#) (QStyleOptionViewItem \*opt, const QModelIndex &idx) const override
- void [initStyleOption](#) (QStyleOptionViewItem \*opt, const QModelIndex &idx) const override

#### 6.1.1 Member Function Documentation

##### 6.1.1.1 [initStyleOption\(\)](#) [1/2]

```
void CenterAlignDelegate::initStyleOption (
 QStyleOptionViewItem * opt,
 const QModelIndex & idx) const [inline], [override]
```

### 6.1.1.2 initStyleOption() [2/2]

```
void CenterAlignDelegate::initStyleOption (
 QStyleOptionViewItem * opt,
 const QModelIndex & idx) const [inline], [override]
```

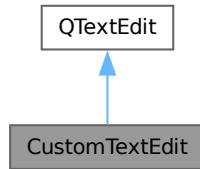
The documentation for this class was generated from the following files:

- [litabledialog.cpp](#)
- [slrtabledialog.cpp](#)

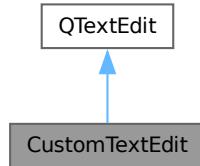
## 6.2 CustomTextEdit Class Reference

```
#include <customtextedit.h>
```

Inheritance diagram for CustomTextEdit:



Collaboration diagram for CustomTextEdit:



### Signals

- void [sendRequested \(\)](#)

### Public Member Functions

- [CustomTextEdit \(QWidget \\*parent=nullptr\)](#)

### Protected Member Functions

- void [keyPressEvent \(QKeyEvent \\*event\)](#) override

## 6.2.1 Constructor & Destructor Documentation

### 6.2.1.1 CustomTextEdit()

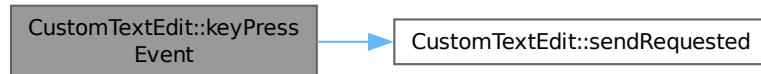
```
CustomTextEdit::CustomTextEdit (
 QWidget * parent = nullptr) [explicit]
```

## 6.2.2 Member Function Documentation

### 6.2.2.1 keyPressEvent()

```
void CustomTextEdit::keyPressEvent (
 QKeyEvent * event) [override], [protected]
```

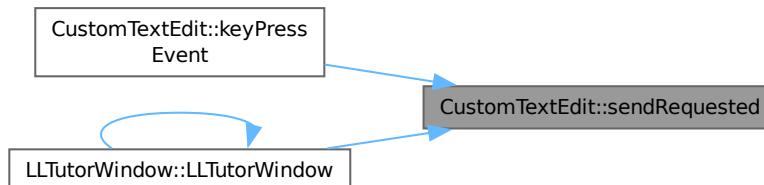
Here is the call graph for this function:



### 6.2.2.2 sendRequested

```
void CustomTextEdit::sendRequested () [signal]
```

Here is the caller graph for this function:



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

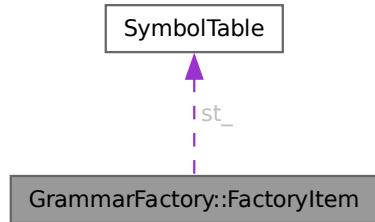
- [customtextedit.h](#)
- [customtextedit.cpp](#)

## 6.3 GrammarFactory::FactoryItem Struct Reference

Represents an individual grammar item with its associated symbol table.

```
#include <grammar_factory.hpp>
```

Collaboration diagram for GrammarFactory::FactoryItem:



## Public Member Functions

- **FactoryItem** (const std::unordered\_map< std::string, std::vector< production > > &grammar)  
Constructor that initializes a **FactoryItem** with the provided grammar.

## Public Attributes

- std::unordered\_map< std::string, std::vector< production > > **g\_**  
Stores the grammar rules where each key is a non-terminal symbol and each value is a vector of production rules.
- **SymbolTable st\_**  
Symbol table associated with this grammar item.

### 6.3.1 Detailed Description

Represents an individual grammar item with its associated symbol table.

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 FactoryItem()

GrammarFactory::FactoryItem::FactoryItem (

```

 const std::unordered_map< std::string, std::vector< production > > & grammar) [explicit]

```

Constructor that initializes a **FactoryItem** with the provided grammar.

#### Parameters

|         |                                                        |
|---------|--------------------------------------------------------|
| grammar | The grammar to initialize the <b>FactoryItem</b> with. |
|---------|--------------------------------------------------------|

### 6.3.3 Member Data Documentation

#### 6.3.3.1 g\_

std::unordered\_map<std::string, std::vector<**production**> > GrammarFactory::FactoryItem::**g\_**  
Stores the grammar rules where each key is a non-terminal symbol and each value is a vector of production rules.

#### 6.3.3.2 st\_

**SymbolTable** GrammarFactory::FactoryItem::**st\_**  
Symbol table associated with this grammar item.  
The documentation for this struct was generated from the following files:

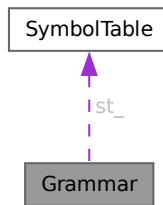
- backend/[grammar\\_factory.hpp](#)
- backend/[grammar\\_factory.cpp](#)

## 6.4 Grammar Struct Reference

Represents a context-free grammar, including its rules, symbol table, and starting symbol.

```
#include <grammar.hpp>
```

Collaboration diagram for Grammar:



### Public Member Functions

- [Grammar \(\)](#)  
Creates a new grammar instance.
- [Grammar \(const std::unordered\\_map< std::string, std::vector< production > > &grammar\)](#)  
Initializes the grammar with an initial set of rules.
- void [SetAxiom \(const std::string &axiom\)](#)  
Sets the axiom (entry point) of the grammar.  
Sets the axiom (entry point) of the grammar.
- bool [IsEmptyProduction \(const std::string &antecedent\) const](#)  
Checks if a given antecedent has an empty production.  
Checks if a given antecedent has an empty production.
- std::vector< std::pair< const std::string, production > > [FilterRulesByConsequent \(const std::string &arg\) const](#)  
Filters grammar rules that contain a specific token in their consequent.  
Filters grammar rules that contain a specific token in their consequent.
- void [Debug \(\) const](#)  
Prints the current grammar structure to standard output.  
Prints the current grammar structure to standard output.
- void [AddProduction \(const std::string &antecedent, const std::vector< std::string > &consequent\)](#)  
Adds a production rule to the grammar and updates the symbol table.  
Adds a production rule to the grammar and updates the symbol table.
- std::vector< std::string > [Split \(const std::string &s\)](#)  
Splits a string into grammar symbols using the current symbol table.  
Splits a string into grammar symbols using the current symbol table.

### Public Attributes

- std::unordered\_map< std::string, std::vector< production > > [g\\_](#)  
Stores the grammar rules with each antecedent mapped to a list of productions.  
Stores the grammar rules with each antecedent mapped to a list of productions.
- std::string [axiom\\_](#)  
The axiom or entry point of the grammar.  
The axiom or entry point of the grammar.
- [SymbolTable st\\_](#)  
Symbol table of the grammar.  
Symbol table of the grammar.

### 6.4.1 Detailed Description

Represents a context-free grammar, including its rules, symbol table, and starting symbol.

This structure encapsulates all components required to define and manipulate a grammar, including production rules, the associated symbol table, and metadata such as the start symbol. It supports construction, transformation, and analysis of grammars.

## 6.4.2 Constructor & Destructor Documentation

### 6.4.2.1 Grammar() [1/2]

Grammar::Grammar () [default]

### 6.4.2.2 Grammar() [2/2]

Grammar::Grammar (

```
 const std::unordered_map< std::string, std::vector< production > > & grammar) [explicit]
```

## 6.4.3 Member Function Documentation

### 6.4.3.1 AddProduction()

```
void Grammar::AddProduction (
 const std::string & antecedent,
 const std::vector< std::string > & consequent)
```

Adds a production rule to the grammar and updates the symbol table.

This function inserts a new production of the form  $A \rightarrow$  into the grammar, where antecedent is the non-terminal A and consequent is the sequence . It also updates the internal symbol table to reflect any new symbols introduced.

Parameters

|            |                                                    |
|------------|----------------------------------------------------|
| antecedent | The left-hand side non-terminal of the production. |
| consequent | The right-hand side sequence of grammar symbols.   |

### 6.4.3.2 Debug()

void Grammar::Debug () const

Prints the current grammar structure to standard output.

This function provides a debug view of the grammar by printing out all rules, the axiom, and other relevant details.

### 6.4.3.3 FilterRulesByConsequent()

```
std::vector< std::pair< const std::string, production > > Grammar::FilterRulesByConsequent (
 const std::string & arg) const
```

Filters grammar rules that contain a specific token in their consequent.

Parameters

|     |                                                              |
|-----|--------------------------------------------------------------|
| arg | The token to search for within the consequents of the rules. |
|-----|--------------------------------------------------------------|

Returns

std::vector of pairs where each pair contains an antecedent and its respective production that includes the specified token.

Searches for rules in which the specified token is part of the consequent and returns those rules.

### 6.4.3.4 HasEmptyProduction()

```
bool Grammar::HasEmptyProduction (
 const std::string & antecedent) const
```

Checks if a given antecedent has an empty production.

Parameters

|            |                                           |
|------------|-------------------------------------------|
| antecedent | The left-hand side (LHS) symbol to check. |
|------------|-------------------------------------------|

Returns

true if there exists an empty production for the antecedent, otherwise false.

An empty production is represented as <antecedent> -> ;, indicating that the antecedent can produce an empty string.

#### 6.4.3.5 SetAxiom()

```
void Grammar::SetAxiom (
 const std::string & axiom)
```

Sets the axiom (entry point) of the grammar.

Parameters

|       |                                                 |
|-------|-------------------------------------------------|
| axiom | The entry point or start symbol of the grammar. |
|-------|-------------------------------------------------|

Defines the starting point for the grammar, which is used in parsing algorithms and must be a non-terminal symbol present in the grammar.

#### 6.4.3.6 Split()

```
std::vector< std::string > Grammar::Split (
 const std::string & s)
```

Splits a string into grammar symbols using the current symbol table.

This function tokenizes the input string s into a sequence of grammar symbols based on the known entries in the symbol table. It uses a greedy approach, matching the longest valid symbol at each step.

Parameters

|   |                            |
|---|----------------------------|
| s | The input string to split. |
|---|----------------------------|

Returns

A vector of grammar symbols extracted from the string.

### 6.4.4 Member Data Documentation

#### 6.4.4.1 axiom\_

```
std::string Grammar::axiom_
```

The axiom or entry point of the grammar.

#### 6.4.4.2 g\_

```
std::unordered_map<std::string, std::vector<production> > Grammar::g_
```

Stores the grammar rules with each antecedent mapped to a list of productions.

#### 6.4.4.3 st\_

```
SymbolTable Grammar::st_
```

Symbol table of the grammar.

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

- backend/[grammar.hpp](#)
- backend/[grammar.cpp](#)

## 6.5 GrammarFactory Struct Reference

Responsible for creating and managing grammar items and performing checks on grammars.

```
#include <grammar_factory.hpp>
```

### Classes

- struct [FactoryItem](#)

Represents an individual grammar item with its associated symbol table.

### Public Member Functions

- void [Init \(\)](#)  
Initializes the [GrammarFactory](#) and populates the items vector with initial grammar items.
- [Grammar PickOne \(int level\)](#)  
Picks a random grammar based on the specified difficulty level (1, 2, or 3).
- [Grammar GenLL1Grammar \(int level\)](#)  
Generates a LL(1) random grammar based on the specified difficulty level.
- [Grammar GenSLR1Grammar \(int level\)](#)  
Generates a SLR(1) random grammar based on the specified difficulty level.
- [Grammar Lv1 \(\)](#)  
Generates a Level 1 grammar.
- [Grammar Lv2 \(\)](#)  
Generates a Level 2 grammar by combining Level 1 items.
- [Grammar Lv3 \(\)](#)  
Generates a Level 3 grammar by combining a Level 2 item and a Level 1 item.
- [Grammar Lv4 \(\)](#)  
Generates a Level 4 grammar by combining Level 3 and Level 1 items.
- [Grammar Lv5 \(\)](#)  
Generates a Level 5 grammar by combining Level 4 and Level 1 items.
- [Grammar Lv6 \(\)](#)  
Generates a Level 6 grammar by combining Level 5 and Level 1 items.
- [Grammar Lv7 \(\)](#)  
Generates a Level 7 grammar by combining Level 6 and Level 1 items.
- [FactoryItem CreateLv2Item \(\)](#)  
Creates a Level 2 grammar item for use in grammar generation.
- bool [HasUnreachableSymbols \(Grammar &grammar\) const](#)  
Checks if a grammar contains unreachable symbols (non-terminals that cannot be derived from the start symbol).
- bool [IsInfinite \(Grammar &grammar\) const](#)  
Checks if a grammar is infinite, meaning there are non-terminal symbols that can never derive a terminal string. This happens when a production leads to an infinite recursion or an endless derivation without reaching terminal symbols. For example, a production like:
- bool [HasDirectLeftRecursion \(const Grammar &grammar\) const](#)  
Checks if a grammar contains direct left recursion (a non-terminal can produce itself on the left side of a production in one step).
- bool [HasIndirectLeftRecursion \(const Grammar &grammar\) const](#)  
Checks if a grammar contains indirect left recursion.
- bool [HasCycle \(const std::unordered\\_map< std::string, std::unordered\\_set< std::string > > &graph\) const](#)  
Checks if directed graph has a cycle using topological sort.
- std::unordered\_set< std::string > [NullableSymbols \(const Grammar &grammar\) const](#)  
Find nullable symbols in a grammar.
- void [RemoveLeftRecursion \(Grammar &grammar\)](#)

Removes direct left recursion in a grammar. A grammar has direct left recursion when one of its productions is:

- void [LeftFactorize \(Grammar &grammar\)](#)  
Performs left factorization. A grammar can be left factorized if it has productions with the same prefix for one non-terminal. For example:
- std::vector< std::string > [LongestCommonPrefix \(const std::vector< production > &productions\)](#)  
Finds the longest common prefix among a set of productions.
- bool [StartsWith \(const production &prod, const std::vector< std::string > &prefix\)](#)  
Checks if a production starts with a given prefix.
- std::string [GenerateNewNonTerminal \(Grammar &grammar, const std::string &base\)](#)  
Generates a new non-terminal symbol that is unique in the grammar.
- void [NormalizeNonTerminals \(FactoryItem &item, const std::string &nt\) const](#)  
Replaces all non-terminal symbols in a grammar item with a single target non-terminal.
- void [AdjustTerminals \(FactoryItem &base, const FactoryItem &cmb, const std::string &target\\_nt\) const](#)  
Adjusts the terminal symbols between two grammar items.
- std::unordered\_map< std::string, std::vector< production > > [Merge \(const FactoryItem &base, const FactoryItem &cmb\) const](#)  
Merges the grammar rules of two grammar items into a single grammar.

## Public Attributes

- std::vector< [FactoryItem](#) > [items](#)  
A vector of [FactoryItem](#) objects representing different level 1 grammar items created by the Init method.
- std::vector< std::string > [terminal\\_alphabet](#)  
A vector of terminal symbols (alphabet) used in the grammar.
- std::vector< std::string > [non\\_terminal\\_alphabet](#)  
A vector of non-terminal symbols (alphabet) used in the grammar.

### 6.5.1 Detailed Description

Responsible for creating and managing grammar items and performing checks on grammars.

### 6.5.2 Member Function Documentation

#### 6.5.2.1 [AdjustTerminals\(\)](#)

```
void GrammarFactory::AdjustTerminals (
 FactoryItem & base,
 const FactoryItem & cmb,
 const std::string & target_nt) const
```

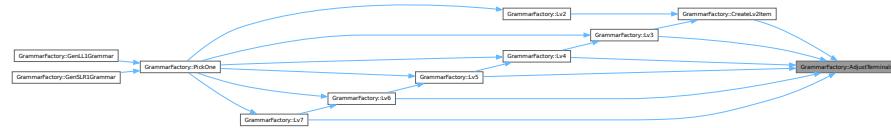
Adjusts the terminal symbols between two grammar items.

This function modifies the terminal symbols of a base grammar item so that they do not conflict with those of the item being combined. It also renames terminals to ensure consistency and inserts the target non-terminal where appropriate.

#### Parameters

|                        |                                                      |
|------------------------|------------------------------------------------------|
| <code>base</code>      | The base grammar item to adjust.                     |
| <code>cmb</code>       | The grammar item being combined with the base.       |
| <code>target_nt</code> | The target non-terminal symbol used for replacement. |

Here is the caller graph for this function:



### 6.5.2.2 CreateLv2Item()

[GrammarFactory::FactoryItem](#) `GrammarFactory::CreateLv2Item ()`

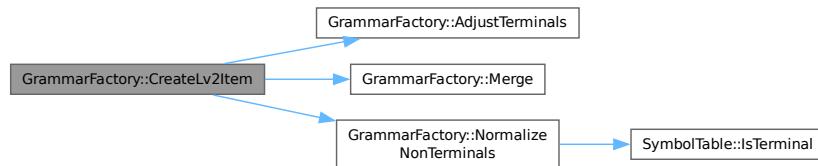
Creates a Level 2 grammar item for use in grammar generation.

This function generates a Level 2 grammar item, which can be used as a building block for creating more complex grammars.

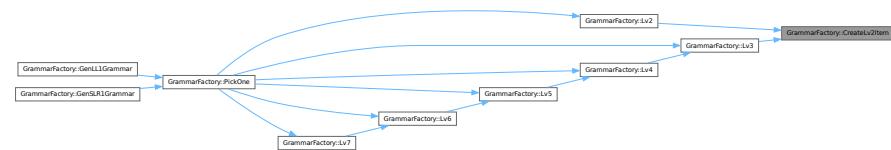
Returns

A [FactoryItem](#) representing a Level 2 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.2.3 GenerateNewNonTerminal()

```
std::string GrammarFactory::GenerateNewNonTerminal (
 Grammar & grammar,
 const std::string & base)
```

Generates a new non-terminal symbol that is unique in the grammar.

This function creates a new non-terminal symbol by appending a prime symbol ('') to the base name until the resulting symbol is not already present in the grammar's symbol table. It is used during left factorization to introduce new non-terminals for factored productions.

Parameters

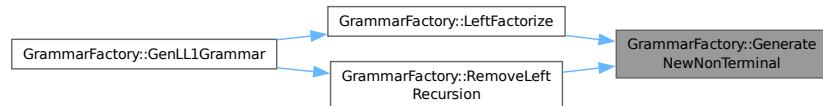
|         |                                                          |
|---------|----------------------------------------------------------|
| grammar | The grammar in which the new non-terminal will be added. |
|---------|----------------------------------------------------------|

|      |                                         |
|------|-----------------------------------------|
| base | The base name for the new non-terminal. |
|------|-----------------------------------------|

## Returns

A unique non-terminal symbol derived from the base name.

Here is the caller graph for this function:



#### 6.5.2.4 GenLL1Grammar()

**Grammar** GrammarFactory::GenLL1Grammar ( int level)

Generates a LL(1) random grammar based on the specified difficulty level.

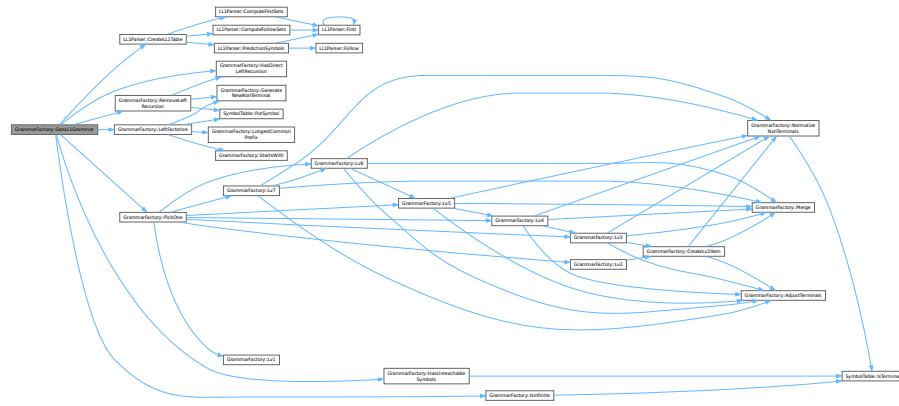
## Parameters

level The difficulty level (1, 2, or 3)

## Returns

A random LL(1) grammar.

Here is the call graph for this function:



### 6.5.2.5 GenSLR1Grammar()

**Grammar** GrammarFactory::GenSLR1Grammar ( int level)

Generates a SLR(1) random grammar based on the specified difficulty level.

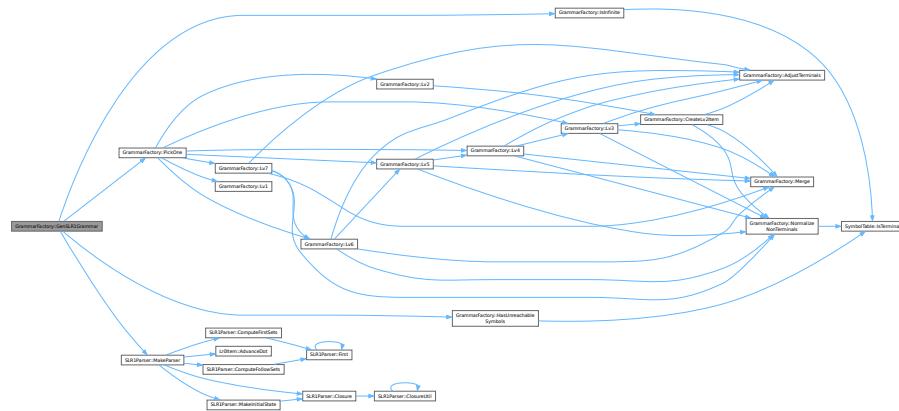
## Parameters

|       |                                   |
|-------|-----------------------------------|
| level | The difficulty level (1, 2, or 3) |
|-------|-----------------------------------|

## Returns

A random SLR(1) grammar.

Here is the call graph for this function:



### 6.5.2.6 HasCycle()

```
bool GrammarFactory::HasCycle (
 const std::unordered_map< std::string, std::unordered_set< std::string > > & graph) const
Checks if directed graph has a cycle using topological sort.
```

## Parameters

graph The directed graph.

## Returns

true if grammar has cycle.

Here is the caller graph for this function:



### 6.5.2.7 HasDirectLeftRecursion()

```
 bool GrammarFactory::HasDirectLeftRecursion (
```

Checks if a grammar contains direct left recursion (a non-terminal can produce itself on the left side of a production in one step).

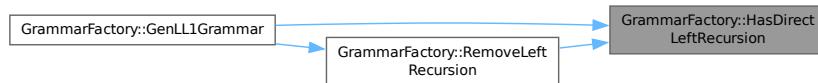
## Parameters

|         |                       |
|---------|-----------------------|
| grammar | The grammar to check. |
|---------|-----------------------|

## Returns

true if there is direct left recursion, false otherwise.

Here is the caller graph for this function:



## 6.5.2.8 HasIndirectLeftRecursion()

```
bool GrammarFactory::HasIndirectLeftRecursion (
 const Grammar & grammar) const
```

Checks if a grammar contains indirect left recursion.

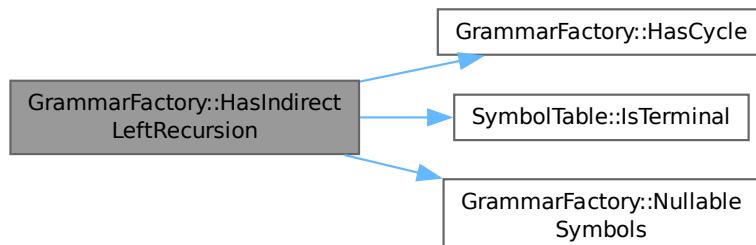
## Parameters

|         |                       |
|---------|-----------------------|
| grammar | The grammar to check. |
|---------|-----------------------|

## Returns

true if there is direct left recursion, false otherwise.

Here is the call graph for this function:



## 6.5.2.9 HasUnreachableSymbols()

```
bool GrammarFactory::HasUnreachableSymbols (
 Grammar & grammar) const
```

Checks if a grammar contains unreachable symbols (non-terminals that cannot be derived from the start symbol).

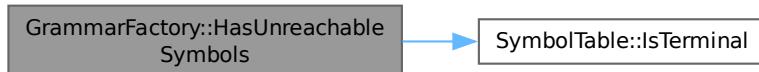
## Parameters

|         |                       |
|---------|-----------------------|
| grammar | The grammar to check. |
|---------|-----------------------|

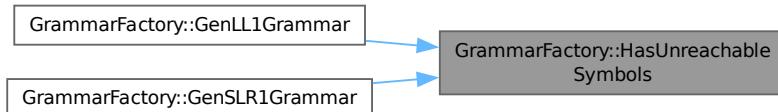
## Returns

true if there are unreachable symbols, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



## 6.5.2.10 Init()

`void GrammarFactory::Init ()`

Initializes the [GrammarFactory](#) and populates the items vector with initial grammar items.

## 6.5.2.11 IsInfinite()

`bool GrammarFactory::IsInfinite (`  
`Grammar & grammar) const`

Checks if a grammar is infinite, meaning there are non-terminal symbols that can never derive a terminal string. This happens when a production leads to an infinite recursion or an endless derivation without reaching terminal symbols. For example, a production like:

`S -> A  
A -> a A | B  
B -> c B`

could lead to an infinite derivation of non-terminals.

## Parameters

|         |                       |
|---------|-----------------------|
| grammar | The grammar to check. |
|---------|-----------------------|

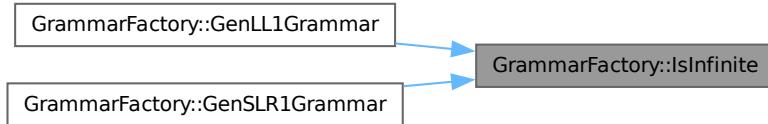
Returns

true if the grammar has infinite derivations, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.2.12 LeftFactorize()

```
void GrammarFactory::LeftFactorize (
 Grammar & grammar)
```

Performs left factorization. A grammar can be left factorized if it has productions with the same prefix for one non-terminal. For example:

$A \rightarrow a x \mid a y$

could be left factorized because it has "a" as the common prefix. The left factorization is done by adding a new non-terminal symbol that contains the uncommon part, and by unifying the common prefix in one production. So:

$A \rightarrow a x \mid a y$

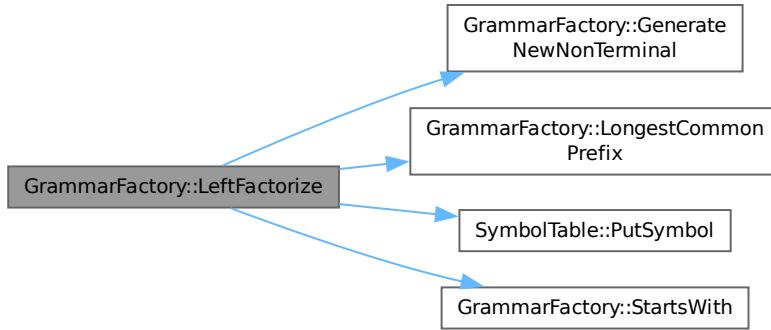
would become:

```
A -> a A'
A' -> x | y
```

Parameters

|         |                                    |
|---------|------------------------------------|
| grammar | The grammar to be left factorized. |
|---------|------------------------------------|

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.2.13 LongestCommonPrefix()

```
std::vector< std::string > GrammarFactory::LongestCommonPrefix (
 const std::vector< production > & productions)
```

Finds the longest common prefix among a set of productions.

This function computes the longest sequence of symbols that is common to the beginning of all productions in the given vector. It is used during left factorization to identify common prefixes that can be factored out.

Parameters

|             |                                     |
|-------------|-------------------------------------|
| productions | A vector of productions to analyze. |
|-------------|-------------------------------------|

Returns

A vector of strings representing the longest common prefix. If no common prefix exists, an empty vector is returned.

Here is the caller graph for this function:



### 6.5.2.14 Lv1()

**Grammar** `GrammarFactory::Lv1 ()`

Generates a Level 1 grammar.

Returns

A Level 1 grammar.

Here is the caller graph for this function:



### 6.5.2.15 Lv2()

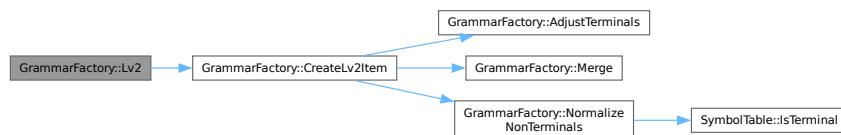
**Grammar** `GrammarFactory::Lv2 ()`

Generates a Level 2 grammar by combining Level 1 items.

Returns

A Level 2 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.2.16 Lv3()

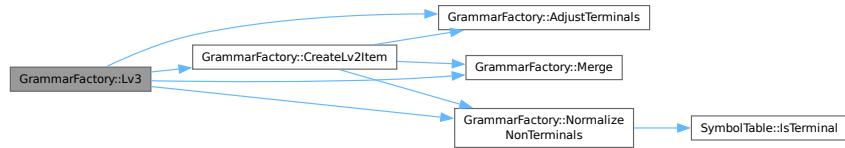
**Grammar** `GrammarFactory::Lv3 ()`

Generates a Level 3 grammar by combining a Level 2 item and a Level 1 item.

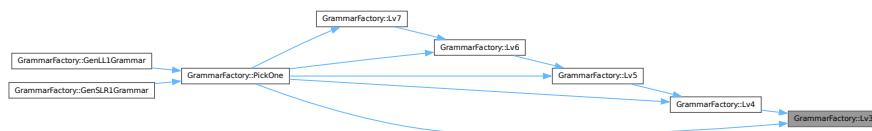
Returns

A Level 3 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.2.17 Lv4()

**Grammar** `GrammarFactory::Lv4 ()`

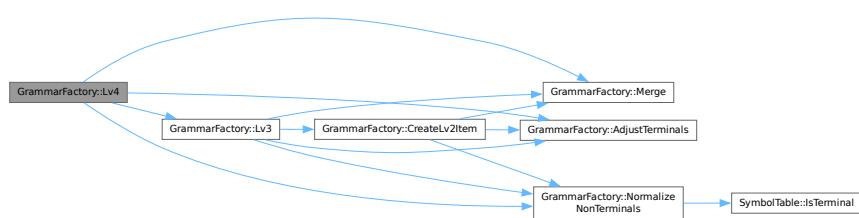
Generates a Level 4 grammar by combining Level 3 and Level 1 items.

This function creates a more complex grammar by combining elements from Level 3 and Level 1 grammars. It is used to generate grammars with increased complexity for testing or parsing purposes.

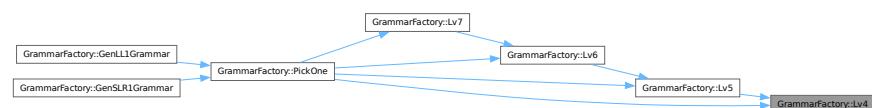
Returns

A Level 4 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.2.18 Lv5()

**Grammar** `GrammarFactory::Lv5 ()`

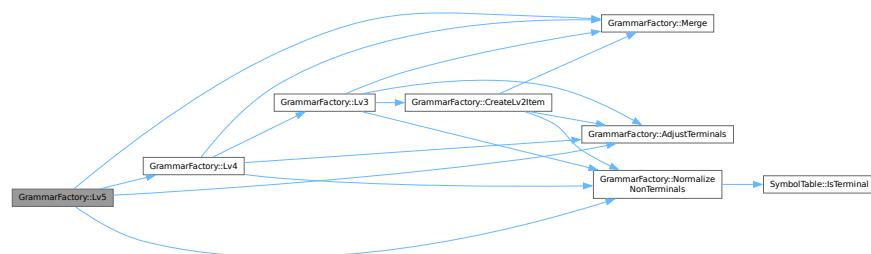
Generates a Level 5 grammar by combining Level 4 and Level 1 items.

This function creates a more advanced grammar by combining elements from Level 4 and Level 1 grammars. It is used to generate grammars with higher complexity for testing or parsing purposes.

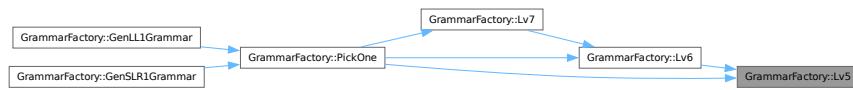
Returns

A Level 5 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.2.19 Lv6()

**Grammar** `GrammarFactory::Lv6 ()`

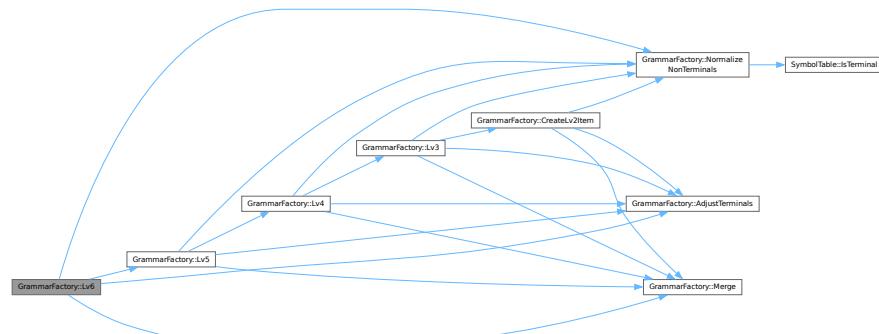
Generates a Level 6 grammar by combining Level 5 and Level 1 items.

This function creates a highly complex grammar by combining elements from Level 5 and Level 1 grammars. It is used to generate grammars with advanced structures for testing or parsing purposes.

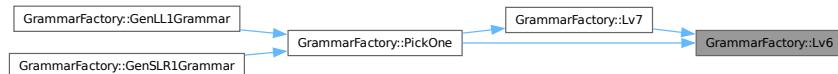
Returns

A Level 6 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.2.20 Lv7()

**Grammar** `GrammarFactory::Lv7 ()`

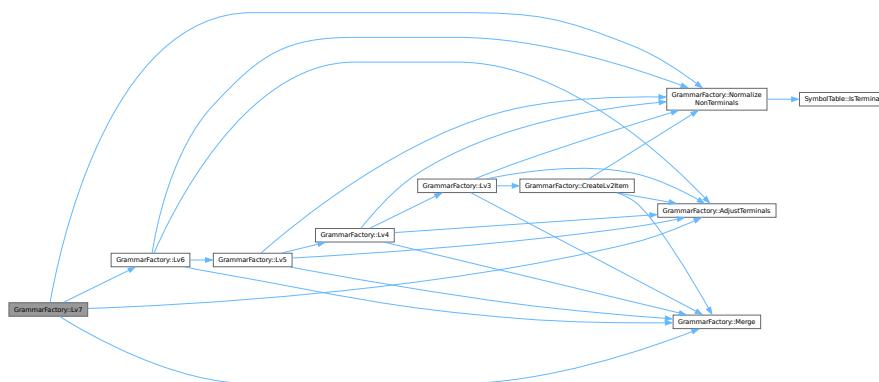
Generates a Level 7 grammar by combining Level 6 and Level 1 items.

This function creates a very complex grammar by combining elements from Level 6 and Level 1 grammars. It is used to generate grammars with highly advanced structures for testing or parsing purposes.

Returns

A Level 7 grammar.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.2.21 Merge()

```

std::unordered_map< std::string, std::vector< production > > GrammarFactory::Merge (
 const FactoryItem & base,
 const FactoryItem & cmb) const

```

Merges the grammar rules of two grammar items into a single grammar.

This function performs a raw combination of the production rules from both grammar items, resulting in a single grammar map that contains all productions.

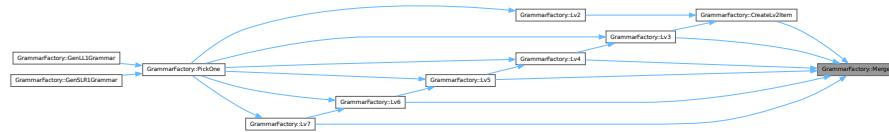
## Parameters

|      |                          |
|------|--------------------------|
| base | The first grammar item.  |
| cmb  | The second grammar item. |

## Returns

A merged grammar map containing all production rules from both inputs.

Here is the caller graph for this function:



## 6.5.2.22 NormalizeNonTerminals()

```
void GrammarFactory::NormalizeNonTerminals (
 FactoryItem & item,
 const std::string & nt) const
```

Replaces all non-terminal symbols in a grammar item with a single target non-terminal.

This function is used during grammar combination to normalize the non-terminal symbols in a given **FactoryItem**, so that they are consistent and compatible with another item.

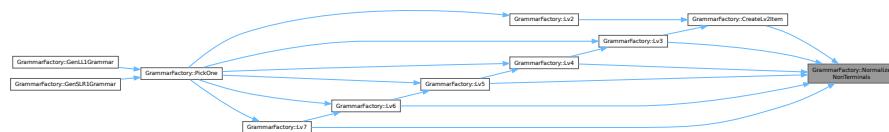
## Parameters

|      |                                                                  |
|------|------------------------------------------------------------------|
| item | The grammar item whose non-terminals will be renamed.            |
| nt   | The new non-terminal symbol that will replace all existing ones. |

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.2.23 NullableSymbols()

```
std::unordered_set< std::string > GrammarFactory::NullableSymbols (
 const Grammar & grammar) const
```

Find nullable symbols in a grammar.

Parameters

|         |                       |
|---------|-----------------------|
| grammar | The grammar to check. |
|---------|-----------------------|

Returns

set of nullable symbols.

Here is the caller graph for this function:



### 6.5.2.24 PickOne()

```
Grammar GrammarFactory::PickOne (
 int level)
```

Picks a random grammar based on the specified difficulty level (1, 2, or 3).

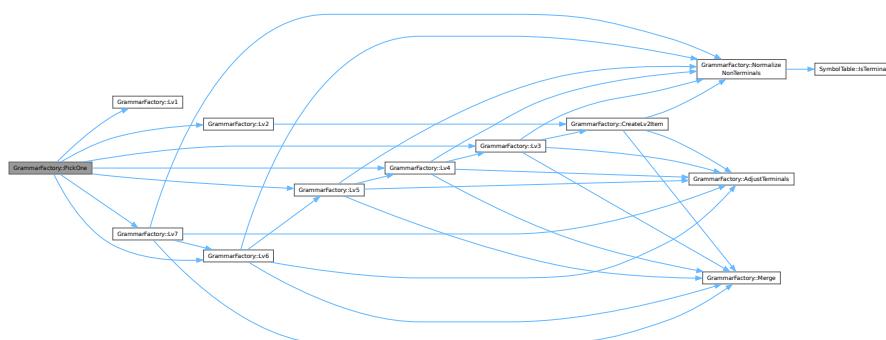
Parameters

|       |                                    |
|-------|------------------------------------|
| level | The difficulty level (1, 2, or 3). |
|-------|------------------------------------|

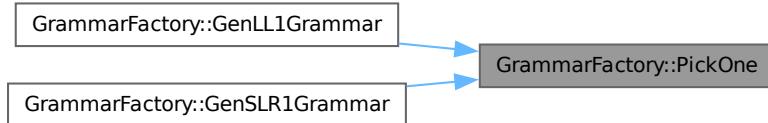
Returns

A randomly picked grammar.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.2.25 RemoveLeftRecursion()

```
void GrammarFactory::RemoveLeftRecursion (
 Grammar & grammar)
```

Removes direct left recursion in a grammar. A grammar has direct left recursion when one of its productions is.

$A \rightarrow A a$

where  $A$  is a non-terminal symbol and "a" the rest of the production. The procedure removes direct left recursion by adding a new non-terminal. So, if the productions with left recursion are:

$A \rightarrow A a | b$

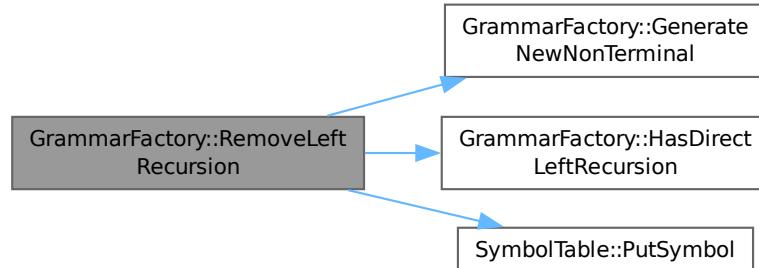
the result would be:

$A \rightarrow b A'$   
 $A' \rightarrow a A' | \epsilon$

Parameters

|         |                                       |
|---------|---------------------------------------|
| grammar | The grammar to remove left recursion. |
|---------|---------------------------------------|

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.5.2.26 StartsWith()

```
bool GrammarFactory::StartsWith (
 const production & prod,
 const std::vector< std::string > & prefix)
```

Checks if a production starts with a given prefix.

This function determines whether the symbols in a production match the provided prefix sequence at the beginning. It is used during left factorization to identify productions that share a common prefix.

Parameters

|        |                                                                             |
|--------|-----------------------------------------------------------------------------|
| prod   | The production to check.                                                    |
| prefix | The sequence of symbols to compare against the beginning of the production. |

Returns

true if the production starts with the prefix, false otherwise.

Here is the caller graph for this function:



## 6.5.3 Member Data Documentation

### 6.5.3.1 items

`std::vector<FactoryItem> GrammarFactory::items`

A vector of [FactoryItem](#) objects representing different level 1 grammar items created by the Init method.

### 6.5.3.2 non\_terminal\_alphabet\_

`std::vector<std::string> GrammarFactory::non_terminal_alphabet_`

Initial value:

```
{"A", "B", "C", "D",
 "E", "F", "G"}
```

A vector of non-terminal symbols (alphabet) used in the grammar.

### 6.5.3.3 terminal\_alphabet\_

`std::vector<std::string> GrammarFactory::terminal_alphabet_`

Initial value:

```
{"a", "b", "c", "d", "e", "f",
 "g", "h", "i", "j", "k", "l"}
```

A vector of terminal symbols (alphabet) used in the grammar.

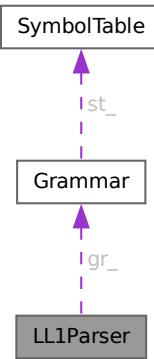
The documentation for this struct was generated from the following files:

- backend/[grammar\\_factory.hpp](#)
- backend/[grammar\\_factory.cpp](#)

## 6.6 LL1Parser Class Reference

```
#include <ll1_parser.hpp>
```

Collaboration diagram for LL1Parser:



### Public Member Functions

- `LL1Parser ()=default`
- `LL1Parser (Grammar gr)`
  - Constructs an `LL1Parser` with a grammar object and an input file.
- `bool CreateLL1Table ()`
  - Creates the LL(1) parsing table for the grammar.
- `void First (std::span< const std::string > rule, std::unordered_set< std::string > &result)`
  - Calculates the FIRST set for a given production rule in a grammar.
- `void ComputeFirstSets ()`
  - Computes the FIRST sets for all non-terminal symbols in the grammar.
- `void ComputeFollowSets ()`
  - Computes the FOLLOW sets for all non-terminal symbols in the grammar. The FOLLOW set of a non-terminal symbol A contains all terminal symbols that can appear immediately after A in any sentential form derived from the grammar's start symbol. Additionally, if A can be the last symbol in a derivation, the end-of-input marker (`\$`) is included in its FOLLOW set. This function computes the FOLLOW sets using the following rules:
    - `std::unordered_set< std::string > Follow (const std::string &arg)`
      - Computes the FOLLOW set for a given non-terminal symbol in the grammar.
    - `std::unordered_set< std::string > PredictionSymbols (const std::string &antecedent, const std::vector< std::string > &consequent)`
      - Computes the prediction symbols for a given production rule.

### Public Attributes

- `ll1_table ll1_t_`
  - The LL(1) parsing table, mapping non-terminals and terminals to productions.
- `Grammar gr_`
  - `Grammar` object associated with this parser.
- `std::unordered_map< std::string, std::unordered_set< std::string > > first_sets_`
  - FIRST sets for each non-terminal in the grammar.
- `std::unordered_map< std::string, std::unordered_set< std::string > > follow_sets_`
  - FOLLOW sets for each non-terminal in the grammar.

## 6.6.1 Constructor & Destructor Documentation

### 6.6.1.1 LL1Parser() [1/2]

`LL1Parser::LL1Parser () [default]`

### 6.6.1.2 LL1Parser() [2/2]

`LL1Parser::LL1Parser (`

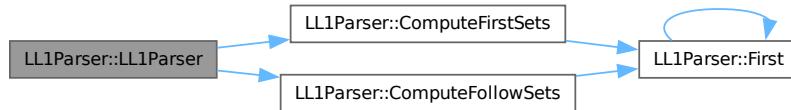
`Grammar gr) [explicit]`

Constructs an `LL1Parser` with a grammar object and an input file.

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <code>gr</code> | <code>Grammar</code> object to parse with |
|-----------------|-------------------------------------------|

Here is the call graph for this function:



## 6.6.2 Member Function Documentation

### 6.6.2.1 ComputeFirstSets()

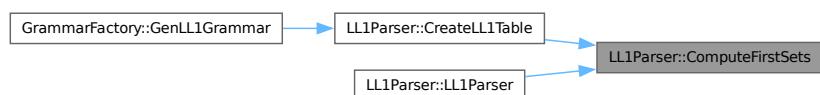
`void LL1Parser::ComputeFirstSets ()`

Computes the FIRST sets for all non-terminal symbols in the grammar.

This function calculates the FIRST set for each non-terminal symbol in the grammar by iteratively applying a least fixed-point algorithm. This approach ensures that the FIRST sets are fully populated by repeatedly expanding and updating the sets until no further changes occur (i.e., a fixed-point is reached). Here is the call graph for this function:



Here is the caller graph for this function:



### 6.6.2.2 ComputeFollowSets()

```
void LL1Parser::ComputeFollowSets ()
```

Computes the FOLLOW sets for all non-terminal symbols in the grammar. The FOLLOW set of a non-terminal symbol A contains all terminal symbols that can appear immediately after A in any sentential form derived from the grammar's start symbol. Additionally, if A can be the last symbol in a derivation, the end-of-input marker (`\$`) is included in its FOLLOW set. This function computes the FOLLOW sets using the following rules:

1. Initialize  $\text{FOLLOW}(S) = \{ \$ \}$ , where S is the start symbol.
2. For each production rule of the form  $A \rightarrow \alpha B \beta$ :
  - Add  $\text{FIRST}(\beta) \setminus \{\epsilon\}$  to  $\text{FOLLOW}(B)$ .
  - If  $\epsilon \in \text{FIRST}(\beta)$ , add  $\text{FOLLOW}(A)$  to  $\text{FOLLOW}(B)$ .
3. Repeat step 2 until no changes occur in any FOLLOW set. The computed FOLLOW sets are cached in the `follow_sets_` member variable for later use by the parser.

Note

This function assumes that the FIRST sets for all symbols have already been computed and are available in the `first_sets_` member variable.

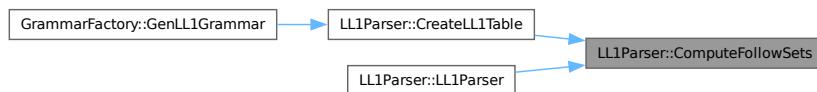
See also

[First](#)  
[follow\\_sets\\_](#)

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.6.2.3 CreateLL1Table()

```
bool LL1Parser::CreateLL1Table ()
```

Creates the LL(1) parsing table for the grammar.

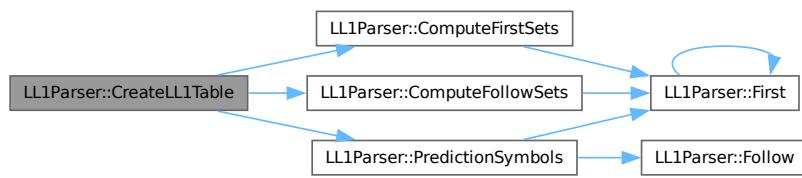
This function constructs the LL(1) parsing table by iterating over each production in the grammar and determining the appropriate cells for each non-terminal and director symbol (prediction symbol) combination. If the grammar is LL(1) compatible, each cell will contain at most one production, indicating no conflicts. If conflicts are found, the function will return false, signaling that the grammar is not LL(1).

- For each production rule  $A \rightarrow \alpha$ , the function calculates the prediction symbols using the `PredictionSymbols` function.
- It then fills the parsing table at the cell corresponding to the non-terminal A and each prediction symbol in the set.
- If a cell already contains a production, this indicates a conflict, meaning the grammar is not LL(1).

Returns

true if the table is created successfully, indicating the grammar is LL(1) compatible; false if any conflicts are detected, showing that the grammar does not meet LL(1) requirements.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.6.2.4 First()

```
void LL1Parser::First (
 std::span< const std::string > rule,
 std::unordered_set< std::string > & result)
```

Calculates the FIRST set for a given production rule in a grammar.

The FIRST set of a production rule contains all terminal symbols that can appear at the beginning of any string derived from that rule. If the rule can derive the empty string (epsilon), epsilon is included in the FIRST set.

This function computes the FIRST set by examining each symbol in the production rule:

- If a terminal symbol is encountered, it is added directly to the FIRST set, as it is the starting symbol of some derivation.
- If a non-terminal symbol is encountered, its FIRST set is recursively computed and added to the result, excluding epsilon unless it is followed by another symbol that could also lead to epsilon.
- If the entire rule could derive epsilon (i.e., each symbol in the rule can derive epsilon), then epsilon is added to the FIRST set.

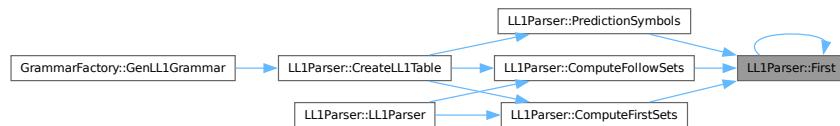
## Parameters

|        |                                                                                                                                                                                                                                        |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rule   | A span of strings representing the production rule for which to compute the FIRST set. Each string in the span is a symbol (either terminal or non-terminal).                                                                          |
| result | A reference to an unordered set of strings where the computed FIRST set will be stored. The set will contain all terminal symbols that can start derivations of the rule, and possibly epsilon if the rule can derive an empty string. |

Here is the call graph for this function:



Here is the caller graph for this function:



## 6.6.2.5 Follow()

```
std::unordered_set< std::string > LL1Parser::Follow (
 const std::string & arg)
```

Computes the FOLLOW set for a given non-terminal symbol in the grammar.

The FOLLOW set for a non-terminal symbol includes all symbols that can appear immediately to the right of that symbol in any derivation, as well as any end-of-input markers if the symbol can appear at the end of derivations. FOLLOW sets are used in LL(1) parsing table construction to determine possible continuations after a non-terminal.

## Parameters

|     |                                                          |
|-----|----------------------------------------------------------|
| arg | Non-terminal symbol for which to compute the FOLLOW set. |
|-----|----------------------------------------------------------|

## Returns

An unordered set of strings containing symbols that form the FOLLOW set for arg.

Here is the caller graph for this function:



### 6.6.2.6 PredictionSymbols()

```
std::unordered_set< std::string > LL1Parser::PredictionSymbols (
 const std::string & antecedent,
 const std::vector< std::string > & consequent)
```

Computes the prediction symbols for a given production rule.

- The prediction symbols for a rule determine the set of input symbols that can trigger this rule in the parsing table. This function calculates the prediction symbols based on the FIRST set of the consequent and, if epsilon (the empty symbol) is in the FIRST set, also includes the FOLLOW set of the antecedent.
- If the FIRST set of the consequent does not contain epsilon, the prediction symbols are simply the FIRST symbols of the consequent.

If the FIRST set of the consequent contains epsilon, the prediction symbols are computed as  $FIRST(\text{consequent}) \setminus \{\epsilon\} \cup FOLLOW(\text{antecedent})$ .

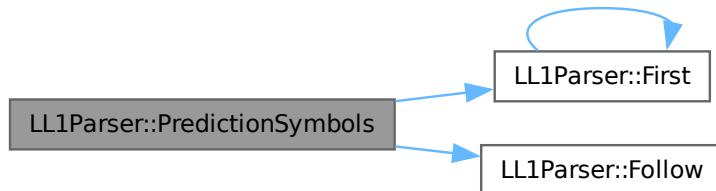
Parameters

|            |                                                                           |
|------------|---------------------------------------------------------------------------|
| antecedent | The left-hand side non-terminal symbol of the rule.                       |
| consequent | A vector of symbols on the right-hand side of the rule (production body). |

Returns

- An unordered set of strings containing the prediction symbols for the specified rule.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.6.3 Member Data Documentation

#### 6.6.3.1 first\_sets\_

```
std::unordered_map<std::string, std::unordered_set<std::string> > LL1Parser::first_sets_
FIRST sets for each non-terminal in the grammar.
```

### 6.6.3.2 follow\_sets\_

`std::unordered_map<std::string, std::unordered_set<std::string> > LL1Parser::follow_sets_`  
 FOLLOW sets for each non-terminal in the grammar.

### 6.6.3.3 gr\_

**Grammar** `LL1Parser::gr_`

Grammar object associated with this parser.

### 6.6.3.4 ll1\_t\_

`ll1_table LL1Parser::ll1_t_`

The LL(1) parsing table, mapping non-terminals and terminals to productions.  
 The documentation for this class was generated from the following files:

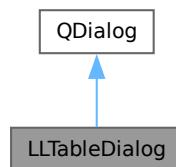
- backend/[ll1\\_parser.hpp](#)
- backend/[ll1\\_parser.cpp](#)

## 6.7 LLTableDialog Class Reference

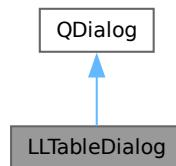
Dialog for filling and submitting an LL(1) parsing table.

#include <lltabledialog.h>

Inheritance diagram for LLTableDialog:



Collaboration diagram for LLTableDialog:



### Signals

- void [submitted](#) (const QVector< QVector< QString > > &data)  
     Signal emitted when the user submits the table.

## Public Member Functions

- **LLTableDialog** (const QStringList &rowHeaders, const QStringList &colHeaders, QWidget \*parent, QVector< QVector< QString > > \*initialData=nullptr)  
Constructs the LL(1) table dialog with given headers and optional initial data.
- QVector< QVector< QString > > **getTableData** () const  
Returns the contents of the table filled by the user.
- void **setInitialData** (const QVector< QVector< QString > > &data)  
Pre-fills the table with existing user data.
- void **highlightIncorrectCells** (const QList< QPair< int, int > > &coords)  
Highlights cells that are incorrect based on provided coordinates.

### 6.7.1 Detailed Description

Dialog for filling and submitting an LL(1) parsing table.

This class represents a dialog window that displays a table for users to complete the LL(1) parsing matrix. It provides functionality to initialize the table with data, retrieve the user's input, and highlight incorrect answers.

### 6.7.2 Constructor & Destructor Documentation

#### 6.7.2.1 LLTableDialog()

```
LLTableDialog::LLTableDialog (
 const QStringList & rowHeaders,
 const QStringList & colHeaders,
 QWidget * parent,
 QVector< QVector< QString > > * initialData = nullptr)
```

Constructs the LL(1) table dialog with given headers and optional initial data.

#### Parameters

|             |                                                |
|-------------|------------------------------------------------|
| rowHeaders  | Row labels (non-terminal symbols).             |
| colHeaders  | Column labels (terminal symbols).              |
| parent      | Parent widget.                                 |
| initialData | Optional initial table data to pre-fill cells. |

### 6.7.3 Member Function Documentation

#### 6.7.3.1 getTableData()

```
QVector< QVector< QString > > LLTableDialog::getTableData () const
```

Returns the contents of the table filled by the user.

#### Returns

A 2D vector representing the LL(1) table.

#### 6.7.3.2 highlightIncorrectCells()

```
void LLTableDialog::highlightIncorrectCells (
 const QList< QPair< int, int > > & coords)
```

Highlights cells that are incorrect based on provided coordinates.

#### Parameters

|        |                                                          |
|--------|----------------------------------------------------------|
| coords | A list of (row, column) pairs to highlight as incorrect. |
|--------|----------------------------------------------------------|

### 6.7.3.3 setInitialData()

```
void LLTableDialog::setInitialData (
 const QVector< QVector< QString > > & data)
```

Pre-fills the table with existing user data.

This is used to populate the table with a previous (possibly incorrect) answer when retrying a task or providing feedback.

Parameters

|             |                                                              |
|-------------|--------------------------------------------------------------|
| <b>data</b> | A 2D vector of strings representing the initial cell values. |
|-------------|--------------------------------------------------------------|

### 6.7.3.4 submitted

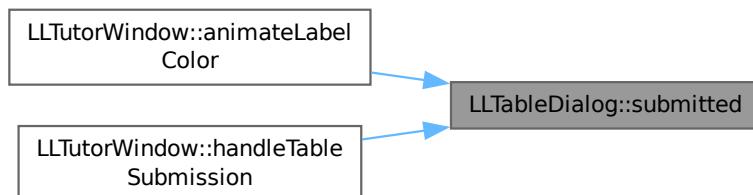
```
void LLTableDialog::submitted (
 const QVector< QVector< QString > > & data) [signal]
```

Signal emitted when the user submits the table.

Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <b>data</b> | The filled table data submitted by the user. |
|-------------|----------------------------------------------|

Here is the caller graph for this function:



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

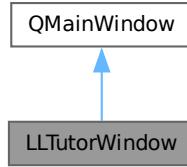
- [lltabledialog.h](#)
- [lltabledialog.cpp](#)

## 6.8 LLTutorWindow Class Reference

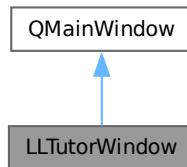
Main window for the LL(1) interactive tutoring mode in SyntaxTutor.

```
#include <lltutorwindow.h>
```

Inheritance diagram for LLTutorWindow:



Collaboration diagram for LLTutorWindow:



## Classes

- struct `TreeNode`  
`TreeNode` structure used to build derivation trees.

## Signals

- void `sessionFinished` (int cntRight, int cntWrong)

## Public Member Functions

- `LLTutorWindow` (const `Grammar` &grammar, `TutorialManager` \*tm=nullptr, `QWidget` \*parent=nullptr)  
Constructs the LL(1) tutor window with a given grammar.
- `~LLTutorWindow` ()
- `QString generateQuestion` ()  
Generates a question for the current state of the tutor.
- void `updateState` (bool isCorrect)  
Updates the tutor state after verifying user response.
- `QString FormatGrammar` (const `Grammar` &grammar)  
Formats a grammar for display in the chat interface.
- void `addMessage` (const `QString` &text, bool isUser)
- void `addWidgetMessage` (`QWidget` \*widget)  
< Add text message to chat
- void `exportConversationToPdf` (const `QString` &filePath)  
< Add widget (e.g., table, tree)
- void `showTable` ()  
< Export chat to PDF

- void `showTableForCPrime ()`  
    Display the full LL(1) table in C' ex.
- void `updateProgressPanel ()`
- void `animateLabelPop (QLabel *label)`
- void `animateLabelColor (QLabel *label, const QColor &flashColor)`
- void `wrongAnimation ()`  
    Visual shake/flash for incorrect answer.
- void `wrongUserResponseAnimation ()`  
    Animation specific to user chat input.
- void `markLastUserIncorrect ()`  
    Marks last message as incorrect.
- void `TeachFirstTree (const std::vector< std::string > &symbols, std::unordered_set< std::string > &first_set, int depth, std::unordered_set< std::string > &processing, QTreeWidgetItem *parent)`
- std::unique\_ptr< `TreeNode` > `buildTreeNode (const std::vector< std::string > &symbols, std::unordered_set< std::string > &first_set, int depth, std::vector< std::pair< std::string, std::vector< std::string > > > &active_derivations)`
- int `computeSubtreeWidth (const std::unique_ptr< TreeNode > &node, int hSpacing)`
- void `drawTree (const std::unique_ptr< TreeNode > &root, QGraphicsScene *scene, QPointF pos, int hSpacing, int vSpacing)`
- void `showTreeGraphics (std::unique_ptr< TreeNode > root)`
- bool `verifyResponse (const QString &userResponse)`
- bool `verifyResponseForA (const QString &userResponse)`
- bool `verifyResponseForA1 (const QString &userResponse)`
- bool `verifyResponseForA2 (const QString &userResponse)`
- bool `verifyResponseForB (const QString &userResponse)`
- bool `verifyResponseForB1 (const QString &userResponse)`
- bool `verifyResponseForB2 (const QString &userResponse)`
- bool `verifyResponseForC ()`
- QString `solution (const std::string &state)`
- QStringList `solutionForA ()`
- QString `solutionForA1 ()`
- QString `solutionForA2 ()`
- QSet< QString > `solutionForB ()`
- QSet< QString > `solutionForB1 ()`
- QSet< QString > `solutionForB2 ()`
- QString `feedback ()`
- QString `feedbackForA ()`
- QString `feedbackForA1 ()`
- QString `feedbackForA2 ()`
- QString `feedbackForAPrime ()`
- QString `feedbackForB ()`
- QString `feedbackForB1 ()`
- QString `feedbackForB2 ()`
- QString `feedbackForBPrime ()`
- QString `feedbackForC ()`
- QString `feedbackForCPrime ()`
- void `feedbackForB1TreeWidget ()`
- void `feedbackForB1TreeGraphics ()`
- QString `TeachFollow (const QString &nt)`
- QString `TeachPredictionSymbols (const QString &ant, const production &conseq)`
- QString `TeachLL1Table ()`
- void `handleTableSubmission (const QVector< QVector< QString > > &raw, const QStringList &colHeaders)`

## Protected Member Functions

- void `closeEvent` (QCloseEvent \*event) override
- bool `eventFilter` (QObject \*obj, QEvent \*event) override

### 6.8.1 Detailed Description

Main window for the LL(1) interactive tutoring mode in SyntaxTutor.

This class guides students through the construction and analysis of LL(1) parsing tables. It uses a finite-state sequence to present progressively more complex tasks, verifies user responses, provides corrective feedback, and supports visualizations like derivation trees.

The tutor is designed to teach the student how the LL(1) table is built, not just test it — including interactive tasks, animated feedback, and hints.

Key features include:

- Interactive question flow based on grammar analysis.
- Derivation tree generation (TeachFirst).
- Step-by-step verification of FIRST, FOLLOW, prediction symbols, and table entries.
- Exportable conversation log for grading or review.

### 6.8.2 Constructor & Destructor Documentation

#### 6.8.2.1 LLTutorWindow()

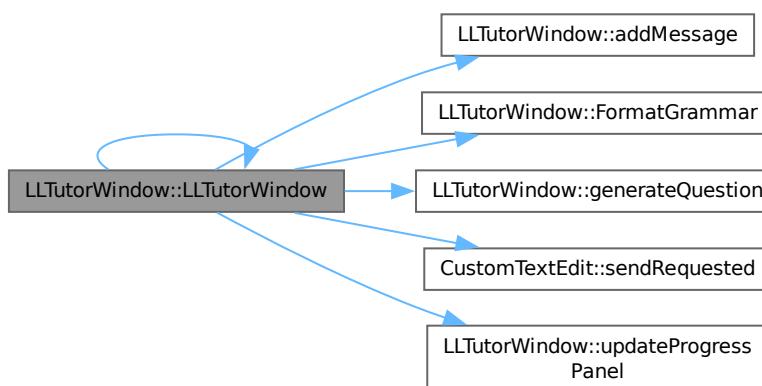
```
LLTutorWindow::LLTutorWindow (
 const Grammar & grammar,
 TutorialManager * tm = nullptr,
 QWidget * parent = nullptr) [explicit]
```

Constructs the LL(1) tutor window with a given grammar.

Parameters

|                      |                                                               |
|----------------------|---------------------------------------------------------------|
| <code>grammar</code> | The grammar to use during the session.                        |
| <code>tm</code>      | Optional pointer to the tutorial manager (for help overlays). |
| <code>parent</code>  | Parent widget.                                                |

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.8.2.2 ~LLTutorWindow()

```
LLTutorWindow::~LLTutorWindow ()
```

### 6.8.3 Member Function Documentation

#### 6.8.3.1 addMessage()

```
void LLTutorWindow::addMessage (
 const QString & text,
 bool isUser)
```

Here is the caller graph for this function:



#### 6.8.3.2 addWidgetMessage()

```
void LLTutorWindow::addWidgetMessage (
 QWidget * widget)
< Add text message to chat
```

#### 6.8.3.3 animateLabelColor()

```
void LLTutorWindow::animateLabelColor (
 QLabel * label,
 const QColor & flashColor)
```

Here is the call graph for this function:



#### 6.8.3.4 animateLabelPop()

```
void LLTutorWindow::animateLabelPop (
 QLabel * label)
```

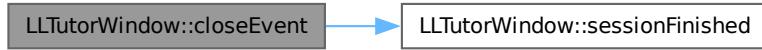
#### 6.8.3.5 buildTreeNode()

```
std::unique_ptr< LLTutorWindow::TreeNode > LLTutorWindow::buildTreeNode (
 const std::vector< std::string > & symbols,
 std::unordered_set< std::string > & first_set,
 int depth,
 std::vector< std::pair< std::string, std::vector< std::string > > > & active_derivations)
```

#### 6.8.3.6 closeEvent()

```
void LLTutorWindow::closeEvent (
 QCloseEvent * event) [inline], [override], [protected]
```

Here is the call graph for this function:



#### 6.8.3.7 computeSubtreeWidth()

```
int LLTutorWindow::computeSubtreeWidth (
 const std::unique_ptr< TreeNode > & node,
 int hSpacing)
```

#### 6.8.3.8 drawTree()

```
void LLTutorWindow::drawTree (
 const std::unique_ptr< TreeNode > & root,
 QGraphicsScene * scene,
 QPointF pos,
 int hSpacing,
 int vSpacing)
```

#### 6.8.3.9 eventFilter()

```
bool LLTutorWindow::eventFilter (
 QObject * obj,
 QEvent * event) [override], [protected]
```

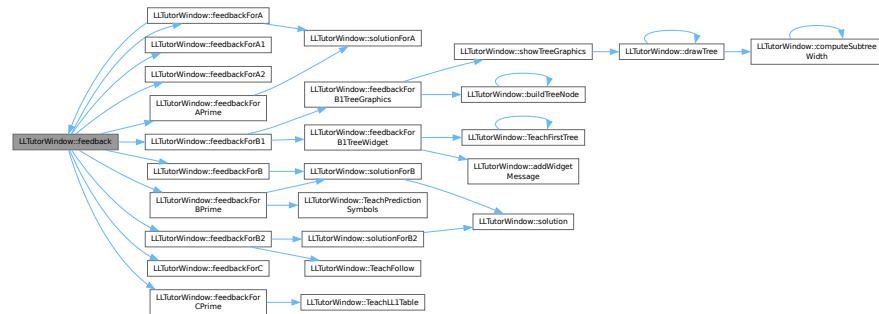
#### 6.8.3.10 exportConversationToPdf()

```
void LLTutorWindow::exportConversationToPdf (
 const QString & filePath)
< Add widget (e.g., table, tree)
```

#### 6.8.3.11 feedback()

```
QString LLTutorWindow::feedback ()
```

Here is the call graph for this function:



### 6.8.3.12 feedbackForA()

`QString LLTutorWindow::feedbackForA ()`

### 6.8.3.13 feedbackForA1()

`QString LLTutorWindow::feedbackForA1 ()`

### 6.8.3.14 feedbackForA2()

`QString LLTutorWindow::feedbackForA2 ()`

### 6.8.3.15 feedbackForAPrime()

`QString LLTutorWindow::feedbackForAPrime ()`

### 6.8.3.16 feedbackForB()

`QString LLTutorWindow::feedbackForB ()`

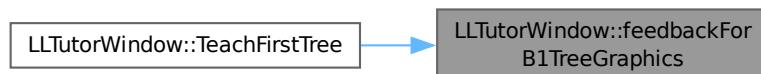
### 6.8.3.17 feedbackForB1()

`QString LLTutorWindow::feedbackForB1 ()`

### 6.8.3.18 feedbackForB1TreeGraphics()

`void LLTutorWindow::feedbackForB1TreeGraphics ()`

Here is the caller graph for this function:



### 6.8.3.19 feedbackForB1TreeWidget()

`void LLTutorWindow::feedbackForB1TreeWidget ()`

Here is the caller graph for this function:



#### 6.8.3.20 feedbackForB2()

`QString LLTutorWindow::feedbackForB2 ()`

#### 6.8.3.21 feedbackForBPrime()

`QString LLTutorWindow::feedbackForBPrime ()`

#### 6.8.3.22 feedbackForC()

`QString LLTutorWindow::feedbackForC ()`

#### 6.8.3.23 feedbackForCPrime()

`QString LLTutorWindow::feedbackForCPrime ()`

#### 6.8.3.24 FormatGrammar()

`QString LLTutorWindow::FormatGrammar (const Grammar & grammar)`

Formats a grammar for display in the chat interface.

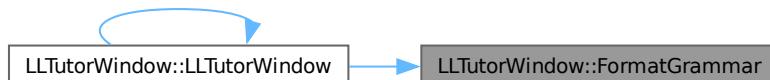
Parameters

|         |                        |
|---------|------------------------|
| grammar | The grammar to format. |
|---------|------------------------|

Returns

A `QString` representation.

Here is the caller graph for this function:



#### 6.8.3.25 generateQuestion()

`QString LLTutorWindow::generateQuestion ()`

Generates a question for the current state of the tutor.

Returns

A formatted question string.

Here is the caller graph for this function:



#### 6.8.3.26 handleTableSubmission()

```
void LLTutorWindow::handleTableSubmission (
 const QVector< QVector< QString > > & raw,
 const QStringList & colHeaders)
```

Here is the call graph for this function:



#### 6.8.3.27 markLastUserIncorrect()

```
void LLTutorWindow::markLastUserIncorrect ()
```

Marks last message as incorrect.

#### 6.8.3.28 sessionFinished

```
void LLTutorWindow::sessionFinished (
 int cntRight,
 int cntWrong) [signal]
```

Here is the caller graph for this function:



#### 6.8.3.29 showTable()

```
void LLTutorWindow::showTable ()
```

< Export chat to PDF

Display the full LL(1) table in C ex.

### 6.8.3.30 showTableForCPrime()

```
void LLTutorWindow::showTableForCPrime ()
Display the full LL(1) table in C' ex.
```

### 6.8.3.31 showTreeGraphics()

```
void LLTutorWindow::showTreeGraphics (
 std::unique_ptr< TreeNode > root)
```

### 6.8.3.32 solution()

```
QString LLTutorWindow::solution (
 const std::string & state)
```

### 6.8.3.33 solutionForA()

```
QStringList LLTutorWindow::solutionForA ()
```

### 6.8.3.34 solutionForA1()

```
QString LLTutorWindow::solutionForA1 ()
```

### 6.8.3.35 solutionForA2()

```
QString LLTutorWindow::solutionForA2 ()
```

### 6.8.3.36 solutionForB()

```
QSet< QString > LLTutorWindow::solutionForB ()
```

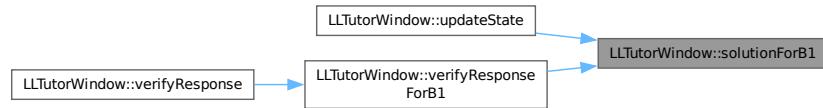
Here is the caller graph for this function:



### 6.8.3.37 solutionForB1()

```
QSet< QString > LLTutorWindow::solutionForB1 ()
```

Here is the caller graph for this function:



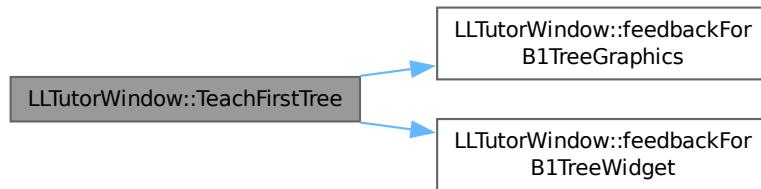
### 6.8.3.38 solutionForB2()

```
QSet< QString > LLTutorWindow::solutionForB2 ()
```

## 6.8.3.39 TeachFirstTree()

```
void LLTutorWindow::TeachFirstTree (
 const std::vector< std::string > & symbols,
 std::unordered_set< std::string > & first_set,
 int depth,
 std::unordered_set< std::string > & processing,
 QTreeWidgetItem * parent)
```

Here is the call graph for this function:



## 6.8.3.40 TeachFollow()

```
QString LLTutorWindow::TeachFollow (
 const QString & nt)
```

## 6.8.3.41 TeachLL1Table()

```
QString LLTutorWindow::TeachLL1Table ()
```

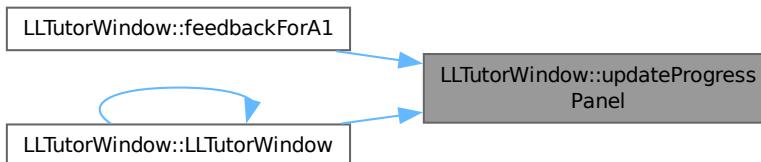
## 6.8.3.42 TeachPredictionSymbols()

```
QString LLTutorWindow::TeachPredictionSymbols (
 const QString & ant,
 const production & conseq)
```

## 6.8.3.43 updateProgressPanel()

```
void LLTutorWindow::updateProgressPanel ()
```

Here is the caller graph for this function:



## 6.8.3.44 updateState()

```
void LLTutorWindow::updateState (
 bool isCorrect)
```

Updates the tutor state after verifying user response.

Parameters

|           |                                      |
|-----------|--------------------------------------|
| isCorrect | Whether the user answered correctly. |
|-----------|--------------------------------------|

#### 6.8.3.45 verifyResponse()

```
bool LLTutorWindow::verifyResponse (
 const QString & userResponse)
```

#### 6.8.3.46 verifyResponseForA()

```
bool LLTutorWindow::verifyResponseForA (
 const QString & userResponse)
```

#### 6.8.3.47 verifyResponseForA1()

```
bool LLTutorWindow::verifyResponseForA1 (
 const QString & userResponse)
```

#### 6.8.3.48 verifyResponseForA2()

```
bool LLTutorWindow::verifyResponseForA2 (
 const QString & userResponse)
```

#### 6.8.3.49 verifyResponseForB()

```
bool LLTutorWindow::verifyResponseForB (
 const QString & userResponse)
```

#### 6.8.3.50 verifyResponseForB1()

```
bool LLTutorWindow::verifyResponseForB1 (
 const QString & userResponse)
```

#### 6.8.3.51 verifyResponseForB2()

```
bool LLTutorWindow::verifyResponseForB2 (
 const QString & userResponse)
```

#### 6.8.3.52 verifyResponseForC()

```
bool LLTutorWindow::verifyResponseForC ()
```

#### 6.8.3.53 wrongAnimation()

```
void LLTutorWindow::wrongAnimation ()
```

Visual shake/flash for incorrect answer.

#### 6.8.3.54 wrongUserResponseAnimation()

```
void LLTutorWindow::wrongUserResponseAnimation ()
```

Animation specific to user chat input.

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

- [lltutorwindow.h](#)
- [lltutorwindow.cpp](#)

## 6.9 Lr0Item Struct Reference

Represents an LR(0) item used in LR automata construction.

```
#include <lr0_item.hpp>
```

### Public Member Functions

- `Lr0Item (std::string antecedent, std::vector< std::string > consequent, std::string epsilon, std::string eol)`  
Constructs an LR(0) item with the dot at position 0.
- `Lr0Item (std::string antecedent, std::vector< std::string > consequent, unsigned int dot, std::string epsilon, std::string eol)`  
Constructs an LR(0) item with a custom dot position.
- `std::string NextToDot () const`  
Returns the symbol immediately after the dot, or empty if the dot is at the end.
- `void PrintItem () const`  
Prints the LR(0) item to the standard output in a human-readable format.
- `std::string ToString () const`  
Converts the item to a string representation, including the dot position.
- `void AdvanceDot ()`  
Advances the dot one position to the right.
- `bool IsComplete () const`  
Checks whether the dot has reached the end of the production.
- `bool operator== (const Lr0Item &other) const`  
Equality operator for comparing two LR(0) items.

### Public Attributes

- `std::string antecedent_`  
The non-terminal on the left-hand side of the production.
- `std::vector< std::string > consequent_`  
The sequence of symbols on the right-hand side of the production.
- `std::string epsilon_`  
The symbol representing the empty string ( $\epsilon$ ).
- `std::string eol_`  
The symbol representing end-of-line or end-of-input (\$).
- `unsigned int dot_ = 0`  
The position of the dot ( $\cdot$ ) in the production.

### 6.9.1 Detailed Description

Represents an LR(0) item used in LR automata construction.

An LR(0) item has a production of the form  $A \rightarrow \alpha \bullet \beta$ , where the dot indicates the current parsing position.

This structure tracks the antecedent (left-hand side), consequent (right-hand side), the dot position, and special symbols like EPSILON and end-of-line (\$).

### 6.9.2 Constructor & Destructor Documentation

#### 6.9.2.1 Lr0Item() [1/2]

```
Lr0Item::Lr0Item (
 std::string antecedent,
 std::vector< std::string > consequent,
 std::string epsilon,
 std::string eol)
```

Constructs an LR(0) item with the dot at position 0.

## Parameters

|            |                                        |
|------------|----------------------------------------|
| antecedent | The left-hand side non-terminal.       |
| consequent | The right-hand side of the production. |
| epsilon    | The EPSILON symbol.                    |
| eol        | The end-of-line symbol.                |

Here is the caller graph for this function:



## 6.9.2.2 Lr0Item() [2/2]

```
Lr0Item::Lr0Item (
 std::string antecedent,
 std::vector< std::string > consequent,
 unsigned int dot,
 std::string epsilon,
 std::string eol)
```

Constructs an LR(0) item with a custom dot position.

## Parameters

|            |                                        |
|------------|----------------------------------------|
| antecedent | The left-hand side non-terminal.       |
| consequent | The right-hand side of the production. |
| dot        | The position of the dot.               |
| epsilon    | The EPSILON symbol.                    |
| eol        | The end-of-line symbol.                |

## 6.9.3 Member Function Documentation

## 6.9.3.1 AdvanceDot()

```
void Lr0Item::AdvanceDot ()
```

Advances the dot one position to the right.

Here is the caller graph for this function:



### 6.9.3.2 IsComplete()

`bool Lr0Item::IsComplete () const`

Checks whether the dot has reached the end of the production.

Returns

true if the item is complete; false otherwise.

Here is the caller graph for this function:



### 6.9.3.3 NextToDot()

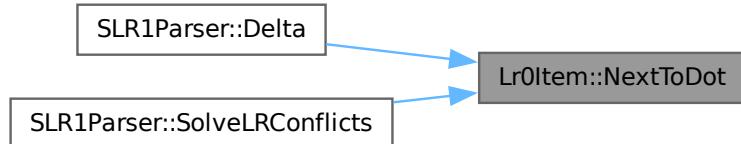
`std::string Lr0Item::NextToDot () const`

Returns the symbol immediately after the dot, or empty if the dot is at the end.

Returns

The symbol after the dot, or an empty string.

Here is the caller graph for this function:



### 6.9.3.4 operator==( )

`bool Lr0Item::operator== (`

`const Lr0Item & other) const`

Equality operator for comparing two LR(0) items.

Parameters

|       |                           |
|-------|---------------------------|
| other | The item to compare with. |
|-------|---------------------------|

Returns

true if both items are equal; false otherwise.

Here is the call graph for this function:



### 6.9.3.5 PrintItem()

`void Lr0Item::PrintItem () const`

Prints the LR(0) item to the standard output in a human-readable format.

### 6.9.3.6 ToString()

`std::string Lr0Item::ToString () const`

Converts the item to a string representation, including the dot position.

Returns

A string representation of the item.

Here is the caller graph for this function:



## 6.9.4 Member Data Documentation

### 6.9.4.1 antecedent\_

`std::string Lr0Item::antecedent_`

The non-terminal on the left-hand side of the production.

### 6.9.4.2 consequent\_

`std::vector<std::string> Lr0Item::consequent_`

The sequence of symbols on the right-hand side of the production.

### 6.9.4.3 dot\_

`unsigned int Lr0Item::dot_ = 0`

The position of the dot (·) in the production.

#### 6.9.4.4 eol\_

`std::string Lr0Item::eol_`

The symbol representing end-of-line or end-of-input (\$).

#### 6.9.4.5 epsilon\_

`std::string Lr0Item::epsilon_`

The symbol representing the empty string (  $\epsilon$  ).

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

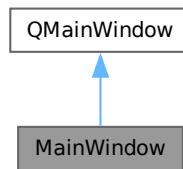
- [backend/lr0\\_item.hpp](#)
- [backend/lr0\\_item.cpp](#)

## 6.10 MainWindow Class Reference

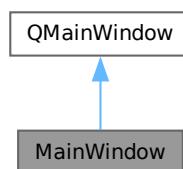
Main application window of SyntaxTutor, managing levels, exercises, and UI state.

#include <mainwindow.h>

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



### Signals

- `void userLevelChanged (unsigned lvl)`  
Emitted when the user's level changes.
- `void userLevelUp (unsigned newLevel)`  
Emitted when the user levels up.

### Public Member Functions

- `MainWindow (QWidget *parent=nullptr)`  
Constructs the main window.
- `~MainWindow ()`

Destructor.

- `unsigned thresholdFor (unsigned level)`  
Returns the required score threshold to unlock a level.
- `unsigned userLevel () const`  
Returns the current user level.
- `void setUserLevel (unsigned lvl)`  
Sets the user level, clamping it to the allowed maximum.

## Properties

- `unsigned userLevel`

### 6.10.1 Detailed Description

Main application window of SyntaxTutor, managing levels, exercises, and UI state.

This class serves as the central hub of the application. It handles level selection, navigation to LL(1) and SLR(1) exercises, tutorial management, settings persistence, and emits signals for user progress. It also includes UI logic for dynamic behavior like unlocking levels and changing language.

### 6.10.2 Constructor & Destructor Documentation

#### 6.10.2.1 MainWindow()

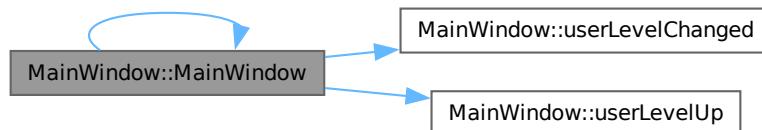
```
MainWindow::MainWindow (
 QWidget * parent = nullptr)
```

Constructs the main window.

#### Parameters

|        |                |
|--------|----------------|
| parent | Parent widget. |
|--------|----------------|

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.10.2.2 ~MainWindow()

`MainWindow::~MainWindow ()`  
Destructor.

### 6.10.3 Member Function Documentation

#### 6.10.3.1 setUserLevel()

`void MainWindow::setUserLevel (`  
`unsigned lvl) [inline]`  
Sets the user level, clamping it to the allowed maximum.

Parameters

|     |                      |
|-----|----------------------|
| lvl | New level to assign. |
|-----|----------------------|

Here is the call graph for this function:



#### 6.10.3.2 thresholdFor()

`unsigned MainWindow::thresholdFor (`  
`unsigned level) [inline]`  
Returns the required score threshold to unlock a level.

Parameters

|       |                   |
|-------|-------------------|
| level | The level number. |
|-------|-------------------|

Returns

The score needed to unlock the given level.

#### 6.10.3.3 userLevel()

`unsigned MainWindow::userLevel () const [inline]`  
Returns the current user level.

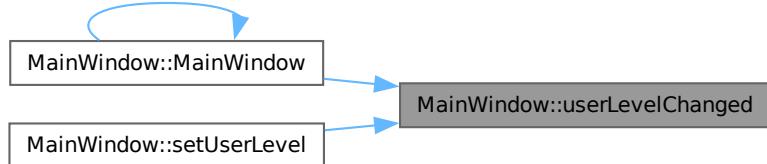
#### 6.10.3.4 userLevelChanged

`void MainWindow::userLevelChanged (`  
`unsigned lvl) [signal]`  
Emitted when the user's level changes.

Parameters

|     |                 |
|-----|-----------------|
| lvl | New user level. |
|-----|-----------------|

Here is the caller graph for this function:



#### 6.10.3.5 userLevelUp

```
void MainWindow::userLevelUp (
 unsigned newLevel) [signal]
```

Emitted when the user levels up.

Parameters

|          |                         |
|----------|-------------------------|
| newLevel | The new level achieved. |
|----------|-------------------------|

Here is the caller graph for this function:



#### 6.10.4 Property Documentation

##### 6.10.4.1 userLevel

```
unsigned MainWindow::userLevel [read], [write]
```

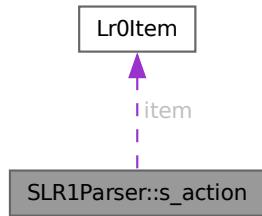
The documentation for this class was generated from the following files:

- [mainwindow.h](#)
- [mainwindow.cpp](#)

### 6.11 SLR1Parser::s\_action Struct Reference

```
#include <slr1_parser.hpp>
```

Collaboration diagram for SLR1Parser::s\_action:



## Public Attributes

- const [Lr0Item](#) \* item
- [Action](#) action

### 6.11.1 Member Data Documentation

#### 6.11.1.1 action

[Action](#) `SLR1Parser::s_action::action`

#### 6.11.1.2 item

`const Lr0Item* SLR1Parser::s_action::item`

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

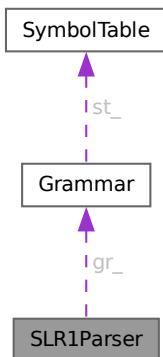
- `backend/slr1_parser.hpp`

## 6.12 SLR1Parser Class Reference

Implements an SLR(1) parser for context-free grammars.

`#include <slr1_parser.hpp>`

Collaboration diagram for SLR1Parser:



## Classes

- struct `s_action`

## Public Types

- enum class `Action` { `Shift` , `Reduce` , `Accept` , `Empty` }

Represents the possible actions in the SLR(1) parsing table.

- using `action_table`  
Represents the action table for the SLR(1) parser.
- using `transition_table`  
Represents the transition table for the SLR(1) parser.

## Public Member Functions

- `SLR1Parser ()=default`
- `SLR1Parser (Grammar gr)`
- `std::unordered_set< Lr0Item > AllItems () const`  
Retrieves all LR(0) items in the grammar. This function returns a set of all LR(0) items derived from the grammar's productions. Each LR(0) item represents a production with a marker indicating the current position in the production (e.g.,  $A \rightarrow \alpha \bullet \beta$ ).
- `void Closure (std::unordered_set< Lr0Item > &items)`  
Computes the closure of a set of LR(0) items.
- `void ClosureUtil (std::unordered_set< Lr0Item > &items, unsigned int size, std::unordered_set< std::string > &visited)`  
Helper function for computing the closure of LR(0) items.
- `std::unordered_set< Lr0Item > Delta (const std::unordered_set< Lr0Item > &items, const std::string &str)`  
Computes the GOTO transition ( $\delta$ ) for a given set of LR(0) items and a symbol. This function is equivalent to the  $\delta(I, X)$  function in LR parsing, where it computes the set of items reached from a state  $I$  via symbol  $X$ .
- `bool SolveLRCConflicts (const state &st)`  
Resolves LR conflicts in a given state.
- `void First (std::span< const std::string > rule, std::unordered_set< std::string > &result)`  
Calculates the FIRST set for a given production rule in a grammar.
- `void ComputeFirstSets ()`  
Computes the FIRST sets for all non-terminal symbols in the grammar.
- `void ComputeFollowSets ()`  
Computes the FOLLOW sets for all non-terminal symbols in the grammar. The FOLLOW set of a non-terminal symbol  $A$  contains all terminal symbols that can appear immediately after  $A$  in any sentential form derived from the grammar's start symbol. Additionally, if  $A$  can be the last symbol in a derivation, the end-of-input marker (`\$`) is included in its FOLLOW set. This function computes the FOLLOW sets using the following rules:
  - `std::unordered_set< std::string > Follow (const std::string &arg)`  
Computes the FOLLOW set for a given non-terminal symbol in the grammar.
- `void MakeInitialState ()`  
Creates the initial state of the parser's state machine.
- `bool MakeParser ()`  
Constructs the SLR(1) parsing tables (action and transition tables).
- `std::string PrintItems (const std::unordered_set< Lr0Item > &items) const`  
Returns a string representation of a set of LR(0) items.

## Public Attributes

- [Grammar gr\\_](#)  
The grammar being processed by the parser.
- std::unordered\_map< std::string, std::unordered\_set< std::string > > [first\\_sets\\_](#)  
Cached FIRST sets for all symbols in the grammar.
- std::unordered\_map< std::string, std::unordered\_set< std::string > > [follow\\_sets\\_](#)  
Cached FOLLOW sets for all non-terminal symbols in the grammar.
- [action\\_table actions\\_](#)  
The action table used by the parser to determine shift/reduce actions.
- [transition\\_table transitions\\_](#)  
The transition table used by the parser to determine state transitions.
- std::unordered\_set< [state](#) > [states\\_](#)  
The set of states in the parser's state machine.

### 6.12.1 Detailed Description

Implements an SLR(1) parser for context-free grammars.

This class builds an SLR(1) parsing table and LR(0) automaton from a given grammar. It provides methods for computing closure sets, GOTO transitions, constructing states, and performing syntax analysis using the generated table.

### 6.12.2 Member Typedef Documentation

#### 6.12.2.1 [action\\_table](#)

using [SLR1Parser::action\\_table](#)

Initial value:

```
std::map<unsigned int, std::map<std::string, SLR1Parser::s_action>
```

Represents the action table for the SLR(1) parser.

The action table is a map that associates each state and input symbol with a specific action (Shift, Reduce, Accept, or Empty). It is used to determine the parser's behavior during the parsing process. The table is structured as:

- Outer map: Keys are state IDs (unsigned int).
- Inner map: Keys are input symbols (std::string), and values are [s\\_action](#) structs representing the action to take.

#### 6.12.2.2 [transition\\_table](#)

using [SLR1Parser::transition\\_table](#)

Initial value:

```
std::map<unsigned int, std::map<std::string, unsigned int>
```

Represents the transition table for the SLR(1) parser.

The transition table is a map that associates each state and symbol with the next state to transition to. It is used to guide the parser's state transitions during the parsing process.

The table is structured as:

- Outer map: Keys are state IDs (unsigned int).
- Inner map: Keys are symbols (std::string), and values are the next state IDs (unsigned int).

### 6.12.3 Member Enumeration Documentation

#### 6.12.3.1 Action

enum class `SLR1Parser::Action` [strong]

Represents the possible actions in the SLR(1) parsing table.

This enumeration defines the types of actions that can be taken by the parser during the parsing process:

- Shift: Shift the input symbol onto the stack and transition to a new state.
- Reduce: Reduce a production rule and pop symbols from the stack.
- Accept: Accept the input as a valid string in the grammar.
- Empty: No action is defined for the current state and input symbol.

Enumerator

|        |  |
|--------|--|
| Shift  |  |
| Reduce |  |
| Accept |  |
| Empty  |  |

### 6.12.4 Constructor & Destructor Documentation

#### 6.12.4.1 SLR1Parser() [1/2]

`SLR1Parser::SLR1Parser ()` [default]

#### 6.12.4.2 SLR1Parser() [2/2]

`SLR1Parser::SLR1Parser (`

`Grammar gr)` [explicit]

Here is the call graph for this function:



### 6.12.5 Member Function Documentation

#### 6.12.5.1 AllItems()

`std::unordered_set<Lr0Item> SLR1Parser::AllItems () const`

Retrieves all LR(0) items in the grammar. This function returns a set of all LR(0) items derived from the grammar's productions. Each LR(0) item represents a production with a marker indicating the current position in the production (e.g.,  $A \rightarrow \alpha \bullet \beta$ ).

Returns

A set of all LR(0) items in the grammar.

### 6.12.5.2 Closure()

```
void SLR1Parser::Closure (
 std::unordered_set< Lr0Item > & items)
```

Computes the closure of a set of LR(0) items.

This function computes the closure of a given set of LR(0) items by adding all items that can be derived from the current items using the grammar's productions. The closure operation ensures that all possible derivations are considered when constructing the parser's states.

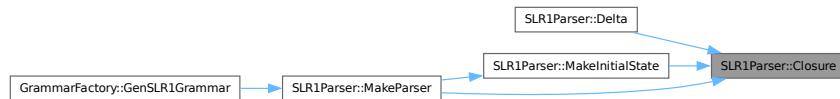
#### Parameters

|       |                                                          |
|-------|----------------------------------------------------------|
| items | The set of LR(0) items for which to compute the closure. |
|-------|----------------------------------------------------------|

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.12.5.3 ClosureUtil()

```
void SLR1Parser::ClosureUtil (
 std::unordered_set< Lr0Item > & items,
 unsigned int size,
 std::unordered_set< std::string > & visited)
```

Helper function for computing the closure of LR(0) items.

This function recursively computes the closure of a set of LR(0) items by adding items derived from non-terminal symbols. It avoids redundant work by tracking visited non-terminals and stopping when no new items are added.

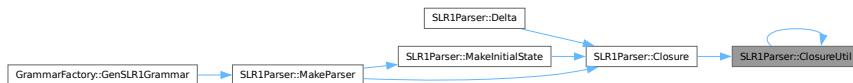
#### Parameters

|         |                                                                  |
|---------|------------------------------------------------------------------|
| items   | The set of LR(0) items being processed.                          |
| size    | The size of the items set at the start of the current iteration. |
| visited | A set of non-terminals that have already been processed.         |

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.12.5.4 ComputeFirstSets()

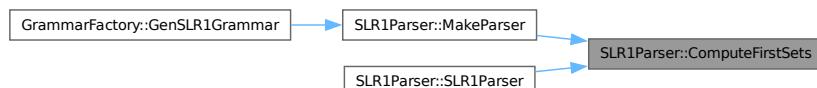
```
void SLR1Parser::ComputeFirstSets ()
```

Computes the FIRST sets for all non-terminal symbols in the grammar.

This function calculates the FIRST set for each non-terminal symbol in the grammar by iteratively applying a least fixed-point algorithm. This approach ensures that the FIRST sets are fully populated by repeatedly expanding and updating the sets until no further changes occur (i.e., a fixed-point is reached). Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.12.5.5 ComputeFollowSets()

```
void SLR1Parser::ComputeFollowSets ()
```

Computes the FOLLOW sets for all non-terminal symbols in the grammar. The FOLLOW set of a non-terminal symbol A contains all terminal symbols that can appear immediately after A in any sentential form derived from the grammar's start symbol. Additionally, if A can be the last symbol in a derivation, the end-of-input marker (`\$`) is included in its FOLLOW set. This function computes the FOLLOW sets using the following rules:

1. Initialize  $\text{FOLLOW}(S) = \{ \$ \}$ , where S is the start symbol.
2. For each production rule of the form  $A \rightarrow \alpha B \beta$ :
  - Add  $\text{FIRST}(\beta) \setminus \{\epsilon\}$  to  $\text{FOLLOW}(B)$ .
  - If  $\epsilon \in \text{FIRST}(\beta)$ , add  $\text{FOLLOW}(A)$  to  $\text{FOLLOW}(B)$ .
3. Repeat step 2 until no changes occur in any FOLLOW set. The computed FOLLOW sets are cached in the `follow_sets_` member variable for later use by the parser.

#### Note

This function assumes that the FIRST sets for all symbols have already been computed and are available in the `first_sets_` member variable.

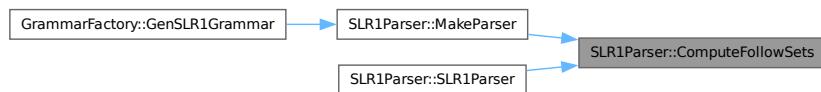
#### See also

[First](#)  
[follow\\_sets\\_](#)

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.12.5.6 Delta()

```
std::unordered_set< Lr0Item > SLR1Parser::Delta (
 const std::unordered_set< Lr0Item > & items,
 const std::string & str)
```

Computes the GOTO transition ( $\delta$ ) for a given set of LR(0) items and a symbol. This function is equivalent to the  $\delta(I, X)$  function in LR parsing, where it computes the set of items reached from a state  $I$  via symbol  $X$ .

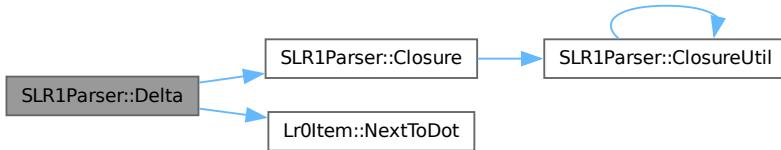
## Parameters

|       |                                             |
|-------|---------------------------------------------|
| items | The current set of LR(0) items (state).     |
| str   | The grammar symbol used for the transition. |

## Returns

The resulting item set after the GOTO transition.

Here is the call graph for this function:



## 6.12.5.7 First()

```
void SLR1Parser::First (
 std::span< const std::string > rule,
 std::unordered_set< std::string > & result)
```

Calculates the FIRST set for a given production rule in a grammar.

The FIRST set of a production rule contains all terminal symbols that can appear at the beginning of any string derived from that rule. If the rule can derive the empty string (epsilon), epsilon is included in the FIRST set.

This function computes the FIRST set by examining each symbol in the production rule:

- If a terminal symbol is encountered, it is added directly to the FIRST set, as it is the starting symbol of some derivation.
- If a non-terminal symbol is encountered, its FIRST set is recursively computed and added to the result, excluding epsilon unless it is followed by another symbol that could also lead to epsilon.
- If the entire rule could derive epsilon (i.e., each symbol in the rule can derive epsilon), then epsilon is added to the FIRST set.

## Parameters

|        |                                                                                                                                                                                                                                        |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rule   | A span of strings representing the production rule for which to compute the FIRST set. Each string in the span is a symbol (either terminal or non-terminal).                                                                          |
| result | A reference to an unordered set of strings where the computed FIRST set will be stored. The set will contain all terminal symbols that can start derivations of the rule, and possibly epsilon if the rule can derive an empty string. |

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.12.5.8 Follow()

```
std::unordered_set< std::string > SLR1Parser::Follow (
 const std::string & arg)
```

Computes the FOLLOW set for a given non-terminal symbol in the grammar.

The FOLLOW set for a non-terminal symbol includes all symbols that can appear immediately to the right of that symbol in any derivation, as well as any end-of-input markers if the symbol can appear at the end of derivations. FOLLOW sets are used in LL(1) parsing table construction to determine possible continuations after a non-terminal.

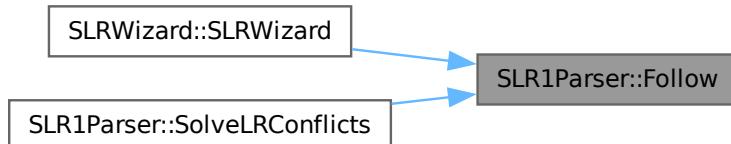
##### Parameters

|     |                                                          |
|-----|----------------------------------------------------------|
| arg | Non-terminal symbol for which to compute the FOLLOW set. |
|-----|----------------------------------------------------------|

##### Returns

An unordered set of strings containing symbols that form the FOLLOW set for arg.

Here is the caller graph for this function:



### 6.12.5.9 MakeInitialState()

```
void SLR1Parser::MakeInitialState ()
```

Creates the initial state of the parser's state machine.

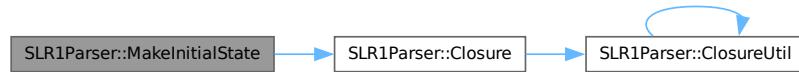
This function initializes the starting state of the parser by computing the closure of the initial set of LR(0) items derived from the grammar's start symbol. The initial state is added to the states\_ set, and its transitions are prepared for further processing in the parser construction.

See also

[states\\_](#)

[transitions\\_](#)

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.12.5.10 MakeParser()

```
bool SLR1Parser::MakeParser ()
```

Constructs the SLR(1) parsing tables (action and transition tables).

This function builds the SLR(1) parsing tables by computing the canonical collection of LR(0) items, generating the action and transition tables, and resolving conflicts (if any). It returns true if the grammar is SLR(1) and the tables are successfully constructed, or false if a conflict is detected that cannot be resolved.

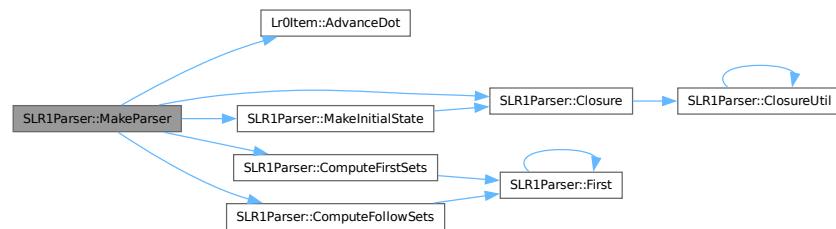
Returns

true if the parsing tables are successfully constructed, false if the grammar is not SLR(1) or a conflict is encountered.

See also

[actions\\_](#)  
[transitions\\_](#)  
[states\\_](#)

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.12.5.11 PrintItems()

```
std::string SLR1Parser::PrintItems (
 const std::unordered_set< Lr0Item > & items) const
```

Returns a string representation of a set of LR(0) items.  
 This function converts a set of LR(0) items into a human-readable string, including dot positions, to help visualize parser states.

Parameters

|       |                                  |
|-------|----------------------------------|
| items | The set of LR(0) items to print. |
|-------|----------------------------------|

Returns

A formatted string representation of the items.

#### 6.12.5.12 SolveLRConflicts()

```
bool SLR1Parser::SolveLRConflicts (
 const state & st)
```

Resolves LR conflicts in a given state.  
 This function attempts to resolve shift/reduce or reduce/reduce conflicts in a given state using SLR(1) parsing rules. It checks the FOLLOW sets of non-terminals to determine the correct action and updates the action table accordingly.

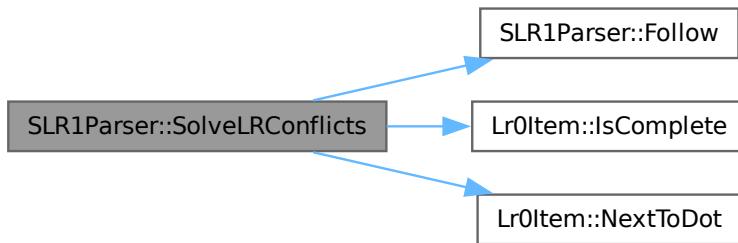
## Parameters

|    |                                          |
|----|------------------------------------------|
| st | The state in which to resolve conflicts. |
|----|------------------------------------------|

## Returns

true if all conflicts are resolved, false if an unresolvable conflict is detected.

Here is the call graph for this function:



## 6.12.6 Member Data Documentation

## 6.12.6.1 actions\_

**action\_table** `SLR1Parser::actions_`

The action table used by the parser to determine shift/reduce actions.

## 6.12.6.2 first\_sets\_

`std::unordered_map<std::string, std::unordered_set<std::string> > SLR1Parser::first_sets_`

Cached FIRST sets for all symbols in the grammar.

## 6.12.6.3 follow\_sets\_

`std::unordered_map<std::string, std::unordered_set<std::string> > SLR1Parser::follow_sets_`

Cached FOLLOW sets for all non-terminal symbols in the grammar.

## 6.12.6.4 gr\_

**Grammar** `SLR1Parser::gr_`

The grammar being processed by the parser.

## 6.12.6.5 states\_

`std::unordered_set<state> SLR1Parser::states_`

The set of states in the parser's state machine.

## 6.12.6.6 transitions\_

**transition\_table** `SLR1Parser::transitions_`

The transition table used by the parser to determine state transitions.

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

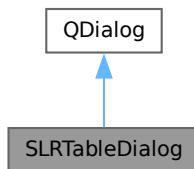
- backend/[slr1\\_parser.hpp](#)
- backend/[slr1\\_parser.cpp](#)

## 6.13 SLRTableDialog Class Reference

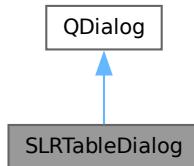
Dialog window for completing and submitting an SLR(1) parsing table.

```
#include <slrtabledialog.h>
```

Inheritance diagram for SLRTableDialog:



Collaboration diagram for SLRTableDialog:



### Public Member Functions

- **SLRTableDialog** (int rowCount, int colCount, const QStringList &colHeaders, QWidget \*parent=nullptr, QVector< QVector< QString > > \*initialData=nullptr)
   
Constructs the SLR(1) table dialog.
- QVector< QVector< QString > > **getTableData** () const
   
Retrieves the content of the table after user interaction.
- void **setInitialData** (const QVector< QVector< QString > > &data)
   
Fills the table with existing data.

### 6.13.1 Detailed Description

Dialog window for completing and submitting an SLR(1) parsing table.

This class displays a table-based UI for students to fill in the ACTION and GOTO parts of the SLR(1) parsing table. It supports initializing the table with data, retrieving user input, and integrating with correction logic in tutorial or challenge mode.

### 6.13.2 Constructor & Destructor Documentation

#### 6.13.2.1 SLRTableDialog()

```
SLRTableDialog::SLRTableDialog (
 int rowCount,
 int colCount,
 const QStringList & colHeaders,
```

```
QWidget * parent = nullptr,
QVector< QVector< QString > > * initialData = nullptr)
Constructs the SLR(1) table dialog.
```

Parameters

|             |                                                           |
|-------------|-----------------------------------------------------------|
| rowCount    | Number of rows (usually equal to number of LR(0) states). |
| colCount    | Number of columns (symbols = terminals + non-terminals).  |
| colHeaders  | Header labels for the columns.                            |
| parent      | Parent widget.                                            |
| initialData | Optional initial data to pre-fill the table.              |

### 6.13.3 Member Function Documentation

#### 6.13.3.1 getTableData()

```
QVector< QVector< QString > > SLRTableDialog::getTableData () const
Retrieves the content of the table after user interaction.
```

Returns

A 2D vector representing the current table values.

#### 6.13.3.2 setInitialData()

```
void SLRTableDialog::setInitialData (
 const QVector< QVector< QString > > & data)
```

Fills the table with existing data.

This method is used to show a previous user submission (e.g., during retries or feedback).

Parameters

|      |                                                 |
|------|-------------------------------------------------|
| data | 2D vector containing the table data to display. |
|------|-------------------------------------------------|

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

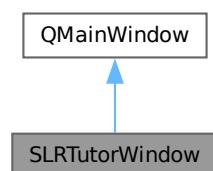
- [slrtabledialog.h](#)
- [slrtabledialog.cpp](#)

## 6.14 SLRTutorWindow Class Reference

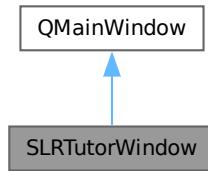
Main window for the SLR(1) interactive tutoring mode in SyntaxTutor.

```
#include <slrtutorwindow.h>
```

Inheritance diagram for SLRTutorWindow:



Collaboration diagram for SLRTutorWindow:



## Signals

- void `sessionFinished` (int cntRight, int cntWrong)

## Public Member Functions

- `SLRTutorWindow` (const `Grammar` &g, `TutorialManager` \*tm=nullptr, `QWidget` \*parent=nullptr)  
Constructs the SLR(1) tutor window with a given grammar.
- `~SLRTutorWindow` ()
- `QString generateQuestion` ()  
Generates a new question for the current tutor state.
- void `updateState` (bool isCorrect)  
Updates tutor state based on whether the last answer was correct.
- `QString FormatGrammar` (const `Grammar` &grammar)
- void `fillSortedGrammar` ()  
< Utility for displaying grammar
- void `addMessage` (const `QString` &text, bool isUser)  
< Prepares grammar in display-friendly format
- void `exportConversationToPdf` (const `QString` &filePath)  
< Add message to chat
- void `showTable` ()  
< Export full interaction
- void `launchSLRWizard` ()  
< Render SLR(1) table
- void `updateProgressPanel` ()
- void `addUserState` (unsigned id)  
< Refresh visual progress
- void `addUserTransition` (unsigned fromId, const std::string &symbol, unsigned toId)  
< Register a user-created state
- void `animateLabelPop` (`QLabel` \*label)
- void `animateLabelColor` (`QLabel` \*label, const `QColor` &flashColor)
- void `wrongAnimation` ()
- void `wrongUserResponseAnimation` ()
- void `markLastUserIncorrect` ()
- bool `verifyResponse` (const `QString` &userResponse)
- bool `verifyResponseForA` (const `QString` &userResponse)
- bool `verifyResponseForA1` (const `QString` &userResponse)
- bool `verifyResponseForA2` (const `QString` &userResponse)
- bool `verifyResponseForA3` (const `QString` &userResponse)
- bool `verifyResponseForA4` (const `QString` &userResponse)

- bool `verifyResponseForB` (const QString &userResponse)
- bool `verifyResponseForC` (const QString &userResponse)
- bool `verifyResponseForCA` (const QString &userResponse)
- bool `verifyResponseForCB` (const QString &userResponse)
- bool `verifyResponseForD` (const QString &userResponse)
- bool `verifyResponseForD1` (const QString &userResponse)
- bool `verifyResponseForD2` (const QString &userResponse)
- bool `verifyResponseForE` (const QString &userResponse)
- bool `verifyResponseForE1` (const QString &userResponse)
- bool `verifyResponseForE2` (const QString &userResponse)
- bool `verifyResponseForF` (const QString &userResponse)
- bool `verifyResponseForFA` (const QString &userResponse)
- bool `verifyResponseForG` (const QString &userResponse)
- bool `verifyResponseForH` ()
- QString `solution` (const std::string &`state`)
- std::unordered\_set< `Lr0Item` > `solutionForA` ()
- QString `solutionForA1` ()
- QString `solutionForA2` ()
- std::vector< std::pair< std::string, std::vector< std::string > > > `solutionForA3` ()
- std::unordered\_set< `Lr0Item` > `solutionForA4` ()
- unsigned `solutionForB` ()
- unsigned `solutionForC` ()
- QStringList `solutionForCA` ()
- std::unordered\_set< `Lr0Item` > `solutionForCB` ()
- QStringList `solutionForD` ()
- QString `solutionForD1` ()
- QString `solutionForD2` ()
- std::ptrdiff\_t `solutionForE` ()
- QSet< unsigned > `solutionForE1` ()
- QMap< unsigned, unsigned > `solutionForE2` ()
- QSet< unsigned > `solutionForF` ()
- QSet< QString > `solutionForFA` ()
- QSet< QString > `solutionForG` ()
- QString `feedback` ()
- QString `feedbackForA` ()
- QString `feedbackForA1` ()
- QString `feedbackForA2` ()
- QString `feedbackForA3` ()
- QString `feedbackForA4` ()
- QString `feedbackForAPrime` ()
- QString `feedbackForB` ()
- QString `feedbackForB1` ()
- QString `feedbackForB2` ()
- QString `feedbackForBPrime` ()
- QString `feedbackForC` ()
- QString `feedbackForCA` ()
- QString `feedbackForCB` ()
- QString `feedbackForD` ()
- QString `feedbackForD1` ()
- QString `feedbackForD2` ()
- QString `feedbackForDPrime` ()
- QString `feedbackForE` ()
- QString `feedbackForE1` ()
- QString `feedbackForE2` ()
- QString `feedbackForF` ()

- `QString feedbackForFA ()`
- `QString feedbackForG ()`
- `QString TeachDeltaFunction (const std::unordered_set< Lr0Item > &items, const QString &symbol)`
- `void TeachClosureStep (std::unordered_set< Lr0Item > &items, unsigned int size, std::unordered_set< std::string > &visited, int depth, QString &output)`
- `QString TeachClosure (const std::unordered_set< Lr0Item > &initialItems)`

#### Protected Member Functions

- `void closeEvent (QCloseEvent *event) override`

### 6.14.1 Detailed Description

Main window for the SLR(1) interactive tutoring mode in SyntaxTutor.

This class implements an interactive, step-by-step tutorial to teach students how to construct SLR(1) parsing tables, including closure, GOTO, automaton construction, FOLLOW sets, and the final table. It supports animated feedback, pedagogical guidance, error correction, and export of the tutoring session. The tutor follows a finite-state flow ([StateSlr](#)) to structure learning, with corrective explanations and automatic evaluation at each step.

### 6.14.2 Constructor & Destructor Documentation

#### 6.14.2.1 SLRTutorWindow()

```
SLRTutorWindow::SLRTutorWindow (
 const Grammar & g,
 TutorialManager * tm = nullptr,
 QWidget * parent = nullptr) [explicit]
```

Constructs the SLR(1) tutor window with a given grammar.

#### Parameters

|                     |                                                             |
|---------------------|-------------------------------------------------------------|
| <code>g</code>      | The grammar used for the session.                           |
| <code>tm</code>     | Optional pointer to the tutorial manager (for guided tour). |
| <code>parent</code> | Parent widget.                                              |

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.14.2.2 ~SLRTutorWindow()

`SLRTutorWindow::~SLRTutorWindow ()`

### 6.14.3 Member Function Documentation

#### 6.14.3.1 addMessage()

```
void SLRTutorWindow::addMessage (
 const QString & text,
 bool isUser)
< Prepares grammar in display-friendly format
```

#### 6.14.3.2 addUserState()

```
void SLRTutorWindow::addUserState (
 unsigned id)
< Refresh visual progress
```

Here is the caller graph for this function:



#### 6.14.3.3 addUserTransition()

```
void SLRTutorWindow::addUserTransition (
 unsigned fromId,
 const std::string & symbol,
 unsigned toId)
< Register a user-created state
```

#### 6.14.3.4 animateLabelColor()

```
void SLRTutorWindow::animateLabelColor (
 QLabel * label,
 const QColor & flashColor)
```

#### 6.14.3.5 animateLabelPop()

```
void SLRTutorWindow::animateLabelPop (
 QLabel * label)
```

#### 6.14.3.6 closeEvent()

```
void SLRTutorWindow::closeEvent (
 QCLOSEEvent * event) [inline], [override], [protected]
```

Here is the call graph for this function:



#### 6.14.3.7 exportConversationToPdf()

```
void SLRTutorWindow::exportConversationToPdf (
 const QString & filePath)
```

< Add message to chat

Here is the call graph for this function:



#### 6.14.3.8 feedback()

```
QString SLRTutorWindow::feedback ()
```

#### 6.14.3.9 feedbackForA()

```
QString SLRTutorWindow::feedbackForA ()
```

#### 6.14.3.10 feedbackForA1()

```
QString SLRTutorWindow::feedbackForA1 ()
```

#### 6.14.3.11 feedbackForA2()

```
QString SLRTutorWindow::feedbackForA2 ()
```

#### 6.14.3.12 feedbackForA3()

```
QString SLRTutorWindow::feedbackForA3 ()
```

#### 6.14.3.13 feedbackForA4()

```
QString SLRTutorWindow::feedbackForA4 ()
```

#### 6.14.3.14 feedbackForAPrime()

```
QString SLRTutorWindow::feedbackForAPrime ()
```

**6.14.3.15 feedbackForB()**

QString SLRTutorWindow::feedbackForB ()

**6.14.3.16 feedbackForB1()**

QString SLRTutorWindow::feedbackForB1 ()

**6.14.3.17 feedbackForB2()**

QString SLRTutorWindow::feedbackForB2 ()

**6.14.3.18 feedbackForBPrime()**

QString SLRTutorWindow::feedbackForBPrime ()

**6.14.3.19 feedbackForC()**

QString SLRTutorWindow::feedbackForC ()

**6.14.3.20 feedbackForCA()**

QString SLRTutorWindow::feedbackForCA ()

**6.14.3.21 feedbackForCB()**

QString SLRTutorWindow::feedbackForCB ()

**6.14.3.22 feedbackForD()**

QString SLRTutorWindow::feedbackForD ()

**6.14.3.23 feedbackForD1()**

QString SLRTutorWindow::feedbackForD1 ()

**6.14.3.24 feedbackForD2()**

QString SLRTutorWindow::feedbackForD2 ()

**6.14.3.25 feedbackForDPrime()**

QString SLRTutorWindow::feedbackForDPrime ()

**6.14.3.26 feedbackForE()**

QString SLRTutorWindow::feedbackForE ()

**6.14.3.27 feedbackForE1()**

QString SLRTutorWindow::feedbackForE1 ()

**6.14.3.28 feedbackForE2()**

QString SLRTutorWindow::feedbackForE2 ()

**6.14.3.29 feedbackForF()**

QString SLRTutorWindow::feedbackForF ()

**6.14.3.30 feedbackForFA()**

QString SLRTutorWindow::feedbackForFA ()

## 6.14.3.31 feedbackForG()

```
QString SLRTutorWindow::feedbackForG ()
```

## 6.14.3.32 fillSortedGrammar()

```
void SLRTutorWindow::fillSortedGrammar ()
< Utility for displaying grammar
```

## 6.14.3.33 FormatGrammar()

```
QString SLRTutorWindow::FormatGrammar (
 const Grammar & grammar)
```

## 6.14.3.34 generateQuestion()

```
QString SLRTutorWindow::generateQuestion ()
```

Generates a new question for the current tutor state.

Returns

The formatted question string.

## 6.14.3.35 launchSLRWizard()

```
void SLRTutorWindow::launchSLRWizard ()
< Render SLR(1) table
```

## 6.14.3.36 markLastUserIncorrect()

```
void SLRTutorWindow::markLastUserIncorrect ()
```

## 6.14.3.37 sessionFinished

```
void SLRTutorWindow::sessionFinished (
 int cntRight,
 int cntWrong) [signal]
```

Here is the caller graph for this function:



## 6.14.3.38 showTable()

```
void SLRTutorWindow::showTable ()
```

< Export full interaction

Here is the caller graph for this function:



#### 6.14.3.39 solution()

```
QString SLRTutorWindow::solution (
 const std::string & state)
```

#### 6.14.3.40 solutionForA()

```
std::unordered_set< Lr0Item > SLRTutorWindow::solutionForA ()
```

#### 6.14.3.41 solutionForA1()

```
QString SLRTutorWindow::solutionForA1 ()
```

#### 6.14.3.42 solutionForA2()

```
QString SLRTutorWindow::solutionForA2 ()
```

#### 6.14.3.43 solutionForA3()

```
std::vector< std::pair< std::string, std::vector< std::string > > > SLRTutorWindow::solutionForA3 ()
```

#### 6.14.3.44 solutionForA4()

```
std::unordered_set< Lr0Item > SLRTutorWindow::solutionForA4 ()
```

Here is the caller graph for this function:



#### 6.14.3.45 solutionForB()

```
unsigned SLRTutorWindow::solutionForB ()
```

Here is the caller graph for this function:



#### 6.14.3.46 solutionForC()

```
unsigned SLRTutorWindow::solutionForC ()
```

Here is the caller graph for this function:



## 6.14.3.47 solutionForCA()

```
QStringList SLRTutorWindow::solutionForCA ()
```

## 6.14.3.48 solutionForCB()

```
std::unordered_set< Lr0Item > SLRTutorWindow::solutionForCB ()
```

## 6.14.3.49 solutionForD()

QStringList SLRTutorWindow::solutionForD ()

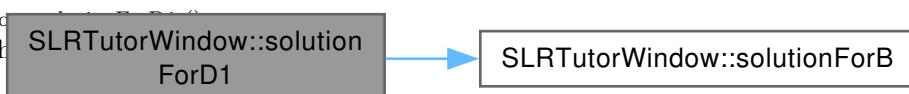
Here is the call graph:



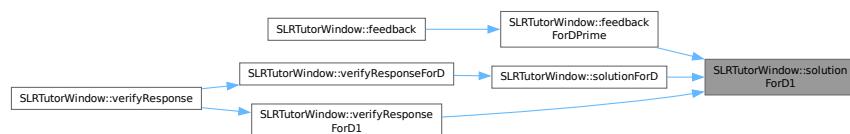
## 6.14.3.50 solutionForD1()

QString SLRTutorWindow::solutionForD1 ()

Here is the call graph:



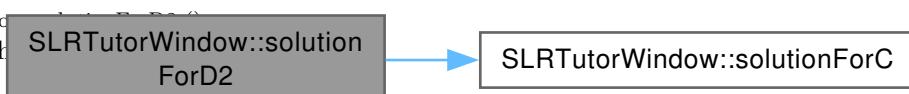
Here is the caller graph for this function:



## 6.14.3.51 solutionForD2()

QString SLRTutorWindow::solutionForD2 ()

Here is the call graph:



## 6.14.3.52 solutionForE()

```
std::ptrdiff_t SLRTutorWindow::solutionForE ()
```

## 6.14.3.53 solutionForE1()

```
QSet< unsigned > SLRTutorWindow::solutionForE1 ()
```

#### 6.14.3.54 solutionForE2()

```
QMap< unsigned, unsigned > SLRTutorWindow::solutionForE2 ()
```

#### 6.14.3.55 solutionForF()

```
QSet< unsigned > SLRTutorWindow::solutionForF ()
```

#### 6.14.3.56 solutionForFA()

```
QSet< QString > SLRTutorWindow::solutionForFA ()
```

Here is the call graph for this function:



#### 6.14.3.57 solutionForG()

```
QSet< QString > SLRTutorWindow::solutionForG ()
```

#### 6.14.3.58 TeachClosure()

```
QString SLRTutorWindow::TeachClosure (
 const std::unordered_set< Lr0Item > & initialItems)
```

#### 6.14.3.59 TeachClosureStep()

```
void SLRTutorWindow::TeachClosureStep (
 std::unordered_set< Lr0Item > & items,
 unsigned int size,
 std::unordered_set< std::string > & visited,
 int depth,
 QString & output)
```

#### 6.14.3.60 TeachDeltaFunction()

```
QString SLRTutorWindow::TeachDeltaFunction (
 const std::unordered_set< Lr0Item > & items,
 const QString & symbol)
```

#### 6.14.3.61 updateProgressPanel()

```
void SLRTutorWindow::updateProgressPanel ()
```

#### 6.14.3.62 updateState()

```
void SLRTutorWindow::updateState (
 bool isCorrect)
```

Updates tutor state based on whether the last answer was correct.

Parameters

|           |                                        |
|-----------|----------------------------------------|
| isCorrect | Whether the user's answer was correct. |
|-----------|----------------------------------------|

Here is the call graph for this function:



#### 6.14.3.63 verifyResponse()

```
bool SLRTutorWindow::verifyResponse (const QString & userResponse)
```

#### 6.14.3.64 verifyResponseForA()

```
bool SLRTutorWindow::verifyResponseForA (const QString & userResponse)
```

#### 6.14.3.65 verifyResponseForA1()

```
bool SLRTutorWindow::verifyResponseForA1 (const QString & userResponse)
```

#### 6.14.3.66 verifyResponseForA2()

```
bool SLRTutorWindow::verifyResponseForA2 (const QString & userResponse)
```

#### 6.14.3.67 verifyResponseForA3()

```
bool SLRTutorWindow::verifyResponseForA3 (const QString & userResponse)
```

#### 6.14.3.68 verifyResponseForA4()

```
bool SLRTutorWindow::verifyResponseForA4 (const QString & userResponse)
```

#### 6.14.3.69 verifyResponseForB()

```
bool SLRTutorWindow::verifyResponseForB (const QString & userResponse)
```

#### 6.14.3.70 verifyResponseForC()

```
bool SLRTutorWindow::verifyResponseForC (const QString & userResponse)
```

#### 6.14.3.71 verifyResponseForCA()

```
bool SLRTutorWindow::verifyResponseForCA (const QString & userResponse)
```

#### 6.14.3.72 verifyResponseForCB()

```
bool SLRTutorWindow::verifyResponseForCB (
 const QString & userResponse)
```

#### 6.14.3.73 verifyResponseForD()

```
bool SLRTutorWindow::verifyResponseForD (
 const QString & userResponse)
```

#### 6.14.3.74 verifyResponseForD1()

```
bool SLRTutorWindow::verifyResponseForD1 (
 const QString & userResponse)
```

#### 6.14.3.75 verifyResponseForD2()

```
bool SLRTutorWindow::verifyResponseForD2 (
 const QString & userResponse)
```

#### 6.14.3.76 verifyResponseForE()

```
bool SLRTutorWindow::verifyResponseForE (
 const QString & userResponse)
```

#### 6.14.3.77 verifyResponseForE1()

```
bool SLRTutorWindow::verifyResponseForE1 (
 const QString & userResponse)
```

Here is the call graph for this function:



#### 6.14.3.78 verifyResponseForE2()

```
bool SLRTutorWindow::verifyResponseForE2 (
 const QString & userResponse)
```

Here is the call graph for this function:



#### 6.14.3.79 verifyResponseForF()

```
bool SLRTutorWindow::verifyResponseForF (
 const QString & userResponse)
```

## 6.14.3.80 verifyResponseForFA()

```
bool SLRTutorWindow::verifyResponseForFA (
 const QString & userResponse)
```

## 6.14.3.81 verifyResponseForG()

```
bool SLRTutorWindow::verifyResponseForG (
 const QString & userResponse)
```

## 6.14.3.82 verifyResponseForH()

```
bool SLRTutorWindow::verifyResponseForH ()
```

## 6.14.3.83 wrongAnimation()

```
void SLRTutorWindow::wrongAnimation ()
```

## 6.14.3.84 wrongUserResponseAnimation()

```
void SLRTutorWindow::wrongUserResponseAnimation ()
```

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

- [slrtutorwindow.h](#)
- [slrtutorwindow.cpp](#)

## 6.15 SLRWizard Class Reference

Interactive assistant that guides the student step-by-step through the SLR(1) parsing table.

```
#include <slrwizard.h>
```

Inheritance diagram for SLRWizard:



Collaboration diagram for SLRWizard:



## Public Member Functions

- **SLRWizard** (`SLR1Parser &parser, const QVector< QVector< QString > > &rawTable, const QStringList &colHeaders, const QVector< QPair< QString, QVector< QString > > > &sortedGrammar, QWidget *parent=nullptr)`  
Constructs the SLR(1) wizard with all necessary parsing context.
- `QVector< QString > stdVectorToQVector` (`const std::vector< std::string > &vec`)  
Converts a `std::vector<std::string>` to `QVector<QString>` for UI compatibility.

### 6.15.1 Detailed Description

Interactive assistant that guides the student step-by-step through the SLR(1) parsing table. This wizard-based dialog presents the user with one cell of the SLR(1) parsing table at a time, asking them to deduce the correct ACTION or GOTO entry based on the LR(0) automaton and FOLLOW sets. It is designed as an educational aid to explain the reasoning behind each parsing decision. Each page includes:

- The current state and symbol (terminal or non-terminal).
- A guided explanation based on the grammar and LR(0) state.
- The expected entry (e.g., s3, r1, acc, or a state number).

### 6.15.2 Constructor & Destructor Documentation

#### 6.15.2.1 SLRWizard()

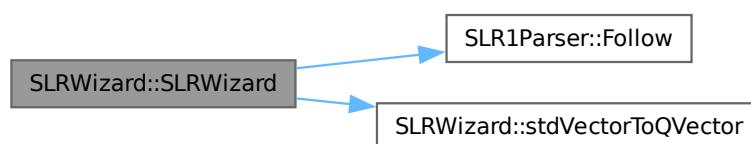
```
SLRWizard::SLRWizard (
 SLR1Parser & parser,
 const QVector< QVector< QString > > & rawTable,
 const QStringList & colHeaders,
 const QVector< QPair< QString, QVector< QString > > > & sortedGrammar,
 QWidget * parent = nullptr) [inline]
```

Constructs the SLR(1) wizard with all necessary parsing context.

#### Parameters

|                            |                                                                         |
|----------------------------|-------------------------------------------------------------------------|
| <code>parser</code>        | The SLR(1) parser instance containing the LR(0) states and transitions. |
| <code>rawTable</code>      | The target parsing table (student version or reference).                |
| <code>colHeaders</code>    | Header symbols (terminals and non-terminals).                           |
| <code>sortedGrammar</code> | Ordered list of grammar rules for reduce explanations.                  |
| <code>parent</code>        | Parent widget.                                                          |

Here is the call graph for this function:



### 6.15.3 Member Function Documentation

#### 6.15.3.1 stdVectorToQVector()

```
QVector< QString > SLRWizard::stdVectorToQVector (
```

const std::vector< std::string > & vec) [inline]

Converts a std::vector<std::string> to QVector<QString> for UI compatibility.

Parameters

|     |                              |
|-----|------------------------------|
| vec | The input vector of strings. |
|-----|------------------------------|

Returns

A QVector of QStringList.

Here is the caller graph for this function:



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

- [slrwizard.h](#)

## 6.16 SLRWizardPage Class Reference

A single step in the SLR(1) guided assistant for table construction.

```
#include <slrwizardpage.h>
```

Inheritance diagram for SLRWizardPage:



Collaboration diagram for SLRWizardPage:



## Public Member Functions

- **SLRWizardPage** (int `state`, const `QString` &`symbol`, const `QString` &`explanation`, const `QString` &`expected`, `QWidget` \*`parent`=`nullptr`)  
Constructs a page for a specific cell in the SLR(1) table.

### 6.16.1 Detailed Description

A single step in the SLR(1) guided assistant for table construction.

This wizard page presents a specific (state, symbol) cell in the SLR(1) parsing table, and prompts the student to enter the correct ACTION or GOTO value.

The page checks the user's input against the expected answer and provides immediate feedback, disabling the "Next" button until the correct response is entered.

### 6.16.2 Constructor & Destructor Documentation

#### 6.16.2.1 SLRWizardPage()

```
SLRWizardPage::SLRWizardPage (
 int state,
 const QString & symbol,
 const QString & explanation,
 const QString & expected,
 QWidget * parent = nullptr) [inline]
```

Constructs a page for a specific cell in the SLR(1) table.

#### Parameters

|                          |                                                                   |
|--------------------------|-------------------------------------------------------------------|
| <code>state</code>       | The state ID (row index in the table).                            |
| <code>symbol</code>      | The grammar symbol (column header).                               |
| <code>explanation</code> | A pedagogical explanation shown to the user.                      |
| <code>expected</code>    | The expected answer (e.g., "s2", "r1", "acc", or a state number). |
| <code>parent</code>      | The parent widget.                                                |

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

- [slrwizardpage.h](#)

### 6.17 state Struct Reference

Represents a state in the LR(0) automaton.

```
#include <state.hpp>
```

## Public Member Functions

- `bool operator==(const state &other) const`  
Equality operator for comparing states based on their items.

## Public Attributes

- `std::unordered_set<Lr0Item> items_`  
The set of LR(0) items that make up this state.
- `unsigned int id_`  
Unique identifier of the state.

### 6.17.1 Detailed Description

Represents a state in the LR(0) automaton.

Each state consists of a unique identifier and a set of LR(0) items that define its core. States are used to build the SLR(1) parsing table.

### 6.17.2 Member Function Documentation

#### 6.17.2.1 operator==()

```
bool state::operator==(
 const state & other) const [inline]
```

Equality operator for comparing states based on their items.

#### Parameters

|       |                            |
|-------|----------------------------|
| other | The state to compare with. |
|-------|----------------------------|

#### Returns

true if both states have the same item set; false otherwise.

### 6.17.3 Member Data Documentation

#### 6.17.3.1 id\_

```
unsigned int state::id_
```

Unique identifier of the state.

#### 6.17.3.2 items\_

```
std::unordered_set<Lr0Item> state::items_
```

The set of LR(0) items that make up this state.  
The documentation for this struct was generated from the following file:

- backend/[state.hpp](#)

## 6.18 SymbolTable Struct Reference

Stores and manages grammar symbols, including their classification and special markers.

```
#include <symbol_table.hpp>
```

## Public Member Functions

- void `PutSymbol` (const std::string &identifier, bool isTerminal)  
Adds a non-terminal symbol to the symbol table.
- bool `In` (const std::string &s) const  
Checks if a symbol exists in the symbol table.
- bool `IsTerminal` (const std::string &s) const  
Checks if a symbol is a terminal.
- bool `IsTerminalWthoEol` (const std::string &s) const  
Checks if a symbol is a terminal excluding EOL.

## Public Attributes

- std::string `EOL_` {"\$"}  
End-of-line symbol used in parsing, initialized as "\$".
- std::string `EPSILON_` {"EPSILON"}  
Epsilon symbol, representing empty transitions, initialized as "EPSILON".
- std::unordered\_map< std::string, `symbol_type` > `st_`  
Main symbol table, mapping identifiers to a pair of symbol type and its regex.
- std::unordered\_set< std::string > `terminals_` {`EOL_`}  
Set of all terminal symbols (including EOL).
- std::unordered\_set< std::string > `terminals_wtho_eol_` {}  
Set of terminal symbols excluding the EOL symbol (\$).
- std::unordered\_set< std::string > `non_terminals_`  
Set of all non-terminal symbols.

### 6.18.1 Detailed Description

Stores and manages grammar symbols, including their classification and special markers. This structure holds information about all terminals and non-terminals used in a grammar, as well as special symbols such as EPSILON and the end-of-line marker (\$). It supports symbol classification, membership checks, and filtered views such as terminals excluding \$.

### 6.18.2 Member Function Documentation

#### 6.18.2.1 In()

```
bool SymbolTable::In (
 const std::string & s) const
Checks if a symbol exists in the symbol table.
```

##### Parameters

|   |                              |
|---|------------------------------|
| s | Symbol identifier to search. |
|---|------------------------------|

##### Returns

true if the symbol is present, otherwise false.

#### 6.18.2.2 IsTerminal()

```
bool SymbolTable::IsTerminal (
 const std::string & s) const
Checks if a symbol is a terminal.
```

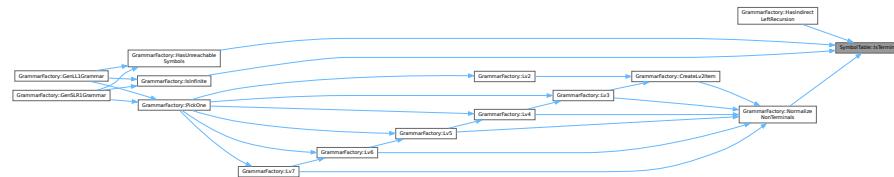
## Parameters

|   |                             |
|---|-----------------------------|
| s | Symbol identifier to check. |
|---|-----------------------------|

## Returns

true if the symbol is terminal, otherwise false.

Here is the caller graph for this function:



## 6.18.2.3 IsTerminalWthoEol()

```
bool SymbolTable::IsTerminalWthoEol (
 const std::string & s) const
```

Checks if a symbol is a terminal excluding EOL.

## Parameters

|   |                             |
|---|-----------------------------|
| s | Symbol identifier to check. |
|---|-----------------------------|

## Returns

true if the symbol is terminal, otherwise false.

## 6.18.2.4 PutSymbol()

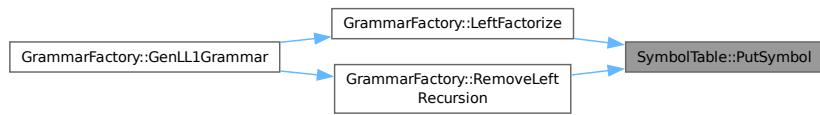
```
void SymbolTable::PutSymbol (
 const std::string & identifier,
 bool isTerminal)
```

Adds a non-terminal symbol to the symbol table.

## Parameters

|            |                                             |
|------------|---------------------------------------------|
| identifier | Name of the symbol.                         |
| isTerminal | True if the identifier is a terminal symbol |

Here is the caller graph for this function:



### 6.18.3 Member Data Documentation

#### 6.18.3.1 EOL\_

`std::string SymbolTable::EOL_ {"$"}  
End-of-line symbol used in parsing, initialized as "$".`

#### 6.18.3.2 EPSILON\_

`std::string SymbolTable::EPSILON_ {"EPSILON"}  
Epsilon symbol, representing empty transitions, initialized as "EPSILON".`

#### 6.18.3.3 non\_terminals\_

`std::unordered_set<std::string> SymbolTable::non_terminals_  
Set of all non-terminal symbols.`

#### 6.18.3.4 st\_

`std::unordered_map<std::string, symbol_type> SymbolTable::st_  
Initial value:`

```
{ {EOL_, symbol_type::TERMINAL}, {EPSILON_, symbol_type::TERMINAL} }
```

Main symbol table, mapping identifiers to a pair of symbol type and its regex.

#### 6.18.3.5 terminals\_

`std::unordered_set<std::string> SymbolTable::terminals_ {EOL_}  
Set of all terminal symbols (including EOL).`

#### 6.18.3.6 terminals\_wtho\_eol\_

`std::unordered_set<std::string> SymbolTable::terminals_wtho_eol_ {}  
Set of terminal symbols excluding the EOL symbol ($).`

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

- backend/[symbol\\_table.hpp](#)
- backend/[symbol\\_table.cpp](#)

## 6.19 LLTutorWindow::TreeNode Struct Reference

[TreeNode](#) structure used to build derivation trees.

```
#include <lltutorwindow.h>
```

Public Attributes

- `QString label`
- `std::vector< std::unique_ptr< TreeNode > > children`

### 6.19.1 Detailed Description

[TreeNode](#) structure used to build derivation trees.

### 6.19.2 Member Data Documentation

#### 6.19.2.1 children

```
std::vector<std::unique_ptr< TreeNode > > LLTutorWindow::TreeNode::children
```

### 6.19.2.2 label

`QString LLTutorWindow::TreeNode::label`

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

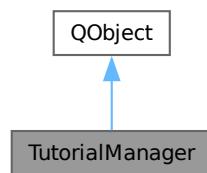
- [lltutorwindow.h](#)

## 6.20 TutorialManager Class Reference

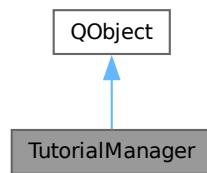
Manages interactive tutorials by highlighting UI elements and guiding the user.

#include <tutorialmanager.h>

Inheritance diagram for TutorialManager:



Collaboration diagram for TutorialManager:



### Public Slots

- void [nextStep \(\)](#)  
Advances to the next tutorial step.

### Signals

- void [stepStarted \(int index\)](#)  
Emitted when a new tutorial step starts.
- void [tutorialFinished \(\)](#)  
Emitted when the full tutorial is finished.
- void [ll1Finished \(\)](#)  
Emitted when the LL(1) tutorial ends.
- void [slr1Finished \(\)](#)  
Emitted when the SLR(1) tutorial ends.

## Public Member Functions

- **TutorialManager** (QWidget \*rootWindow)  
Constructs a **TutorialManager** for a given window.
- void **addStep** (QWidget \*target, const QString &htmlText)  
Adds a new step to the tutorial sequence.
- void **start** ()  
Starts the tutorial from the beginning.
- void **setRootWindow** (QWidget \*newRoot)  
Sets the root window (used for repositioning the overlay).
- void **clearSteps** ()  
Clears all steps in the tutorial.
- void **hideOverlay** ()  
Hides the tutorial overlay immediately.
- void **finishLL1** ()  
Ends the LL(1) tutorial sequence and emits its corresponding signal.
- void **finishSLR1** ()  
Ends the SLR(1) tutorial sequence and emits its corresponding signal.

## Protected Member Functions

- bool **eventFilter** (QObject \*obj, QEvent \*ev) override  
Intercepts UI events to handle overlay behavior.

### 6.20.1 Detailed Description

Manages interactive tutorials by highlighting UI elements and guiding the user. This class implements a step-by-step overlay system that visually highlights widgets and shows textual instructions to guide the user through the interface. It supports multiple tutorials (e.g., for LL(1) and SLR(1) modes), with custom steps and signals for tutorial completion.

### 6.20.2 Constructor & Destructor Documentation

#### 6.20.2.1 TutorialManager()

**TutorialManager::TutorialManager** (  
    QWidget \* rootWindow)

Constructs a **TutorialManager** for a given window.

#### Parameters

|            |                                                            |
|------------|------------------------------------------------------------|
| rootWindow | The main application window used for relative positioning. |
|------------|------------------------------------------------------------|

### 6.20.3 Member Function Documentation

#### 6.20.3.1 addStep()

```
void TutorialManager::addStep (
 QWidget * target,
 const QString & htmlText)
```

Adds a new step to the tutorial sequence.

#### Parameters

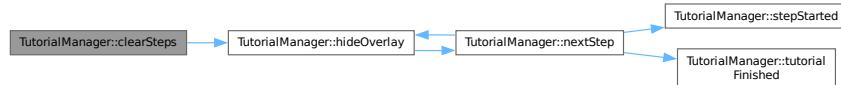
|          |                                              |
|----------|----------------------------------------------|
| target   | The widget to highlight during the step.     |
| htmlText | The instructional HTML message for the step. |

### 6.20.3.2 clearSteps()

```
void TutorialManager::clearSteps ()
```

Clears all steps in the tutorial.

Here is the call graph for this function:



### 6.20.3.3 eventFilter()

```
bool TutorialManager::eventFilter (
```

    QObject \* obj,

    QEvent \* ev) [override], [protected]

Intercepts UI events to handle overlay behavior.

### 6.20.3.4 finishLL1()

```
void TutorialManager::finishLL1 ()
```

Ends the LL(1) tutorial sequence and emits its corresponding signal.

Here is the call graph for this function:



### 6.20.3.5 finishSLR1()

```
void TutorialManager::finishSLR1 ()
```

Ends the SLR(1) tutorial sequence and emits its corresponding signal.

Here is the call graph for this function:



### 6.20.3.6 hideOverlay()

```
void TutorialManager::hideOverlay ()
```

Hides the tutorial overlay immediately.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.20.3.7 ll1Finished

`void TutorialManager::ll1Finished () [signal]`

Emitted when the LL(1) tutorial ends.

Here is the caller graph for this function:

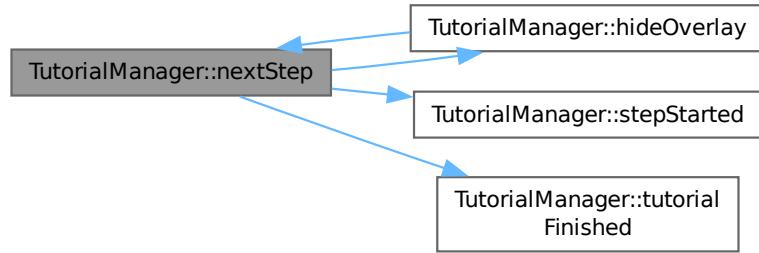


#### 6.20.3.8 nextStep

`void TutorialManager::nextStep () [slot]`

Advances to the next tutorial step.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.20.3.9 setRootWindow()

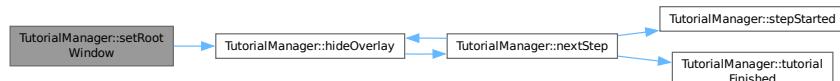
```
void TutorialManager::setRootWindow (
 QWidget * newRoot)
```

Sets the root window (used for repositioning the overlay).

Parameters

|         |                                   |
|---------|-----------------------------------|
| newRoot | The new main window to reference. |
|---------|-----------------------------------|

Here is the call graph for this function:



### 6.20.3.10 slr1Finished

```
void TutorialManager::slr1Finished () [signal]
```

Emitted when the SLR(1) tutorial ends.

Here is the caller graph for this function:

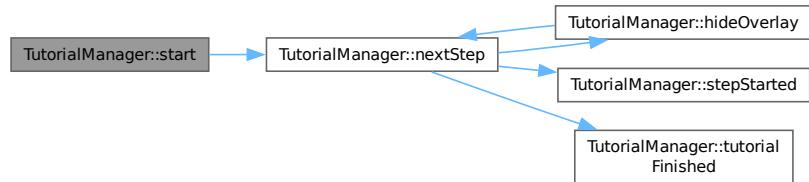


#### 6.20.3.11 start()

`void TutorialManager::start ()`

Starts the tutorial from the beginning.

Here is the call graph for this function:



#### 6.20.3.12 stepStarted

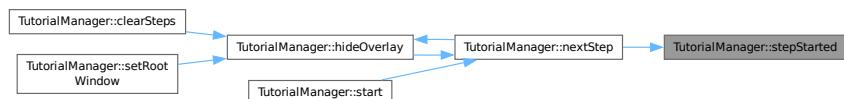
`void TutorialManager::stepStarted ( int index ) [signal]`

Emitted when a new tutorial step starts.

Parameters

|       |                            |
|-------|----------------------------|
| index | Index of the current step. |
|-------|----------------------------|

Here is the caller graph for this function:

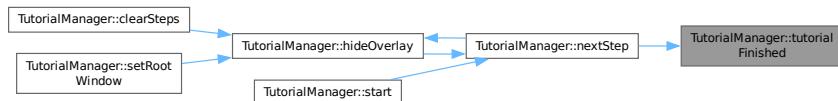


#### 6.20.3.13 tutorialFinished

`void TutorialManager::tutorialFinished () [signal]`

Emitted when the full tutorial is finished.

Here is the caller graph for this function:



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

- [tutorialmanager.h](#)
- [tutorialmanager.cpp](#)

## 6.21 TutorialStep Struct Reference

Represents a single step in the tutorial sequence.

```
#include <tutorialmanager.h>
```

Public Attributes

- `QWidget * target`  
Widget to highlight during the tutorial step.
- `QString htmlText`  
HTML text to show as instruction or explanation.

### 6.21.1 Detailed Description

Represents a single step in the tutorial sequence.

Each step highlights a target widget and displays an associated HTML-formatted message.

### 6.21.2 Member Data Documentation

#### 6.21.2.1 `htmlText`

`QString TutorialStep::htmlText`

HTML text to show as instruction or explanation.

#### 6.21.2.2 `target`

`QWidget* TutorialStep::target`

Widget to highlight during the tutorial step.

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

- [tutorialmanager.h](#)

## 6.22 UniqueQueue< T > Class Template Reference

A queue that ensures each element is inserted only once.

```
#include <UniqueQueue.h>
```

Public Member Functions

- `void push (const T &value)`  
Pushes an element to the queue if it hasn't been inserted before.
- `void pop ()`  
Removes the front element from the queue.

- const T & `front () const`  
Accesses the front element of the queue.
- bool `empty () const`  
Checks whether the queue is empty.
- void `clear ()`  
Clears the queue and the set of seen elements.

### 6.22.1 Detailed Description

```
template<typename T>
class UniqueQueue< T >
```

A queue that ensures each element is inserted only once.

This data structure behaves like a standard FIFO queue but prevents duplicate insertions. Internally, it uses a std::queue for ordering and a std::unordered\_set to track seen elements.

Template Parameters

|   |                                                                            |
|---|----------------------------------------------------------------------------|
| T | The type of elements stored in the queue. Must be hashable and comparable. |
|---|----------------------------------------------------------------------------|

### 6.22.2 Member Function Documentation

#### 6.22.2.1 clear()

```
template<typename T>
void UniqueQueue< T >::clear () [inline]
```

Clears the queue and the set of seen elements.

#### 6.22.2.2 empty()

```
template<typename T>
bool UniqueQueue< T >::empty () const [inline]
```

Checks whether the queue is empty.

Returns

true if the queue is empty; false otherwise.

#### 6.22.2.3 front()

```
template<typename T>
const T & UniqueQueue< T >::front () const [inline]
```

Accesses the front element of the queue.

Returns

A reference to the front element.

#### 6.22.2.4 pop()

```
template<typename T>
void UniqueQueue< T >::pop () [inline]
```

Removes the front element from the queue.

#### 6.22.2.5 push()

```
template<typename T>
void UniqueQueue< T >::push (
 const T & value) [inline]
```

Pushes an element to the queue if it hasn't been inserted before.

## Parameters

|       |                        |
|-------|------------------------|
| value | The element to insert. |
|-------|------------------------|

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

- [UniqueQueue.h](#)



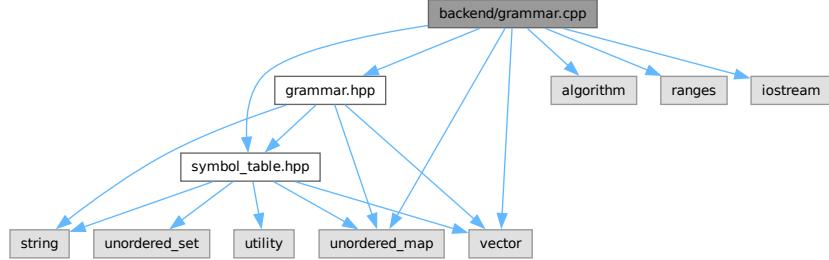
# Chapter 7

## File Documentation

### 7.1 backend/grammar.cpp File Reference

```
#include "grammar.hpp"
#include "symbol_table.hpp"
#include <algorithm>
#include <iostream>
#include <ranges>
#include <unordered_map>
#include <vector>
```

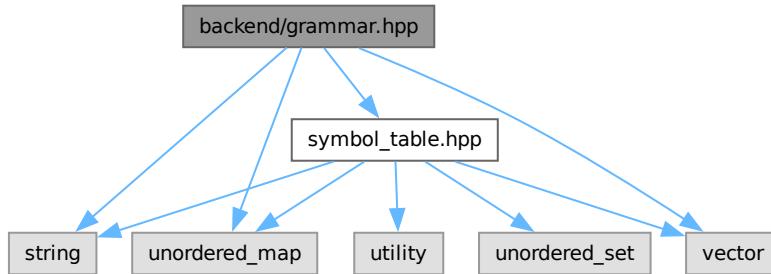
Include dependency graph for grammar.cpp:



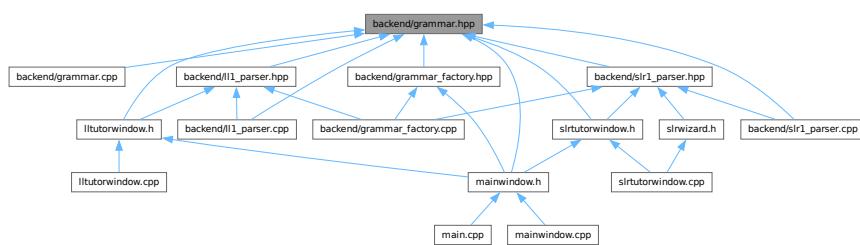
### 7.2 backend/grammar.hpp File Reference

```
#include "symbol_table.hpp"
#include <string>
#include <unordered_map>
#include <vector>
```

Include dependency graph for grammar.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct **Grammar**

Represents a context-free grammar, including its rules, symbol table, and starting symbol.

## TypeDefs

- using **production** = std::vector<std::string>
- Represents the right-hand side of a grammar rule.

### 7.2.1 Typedef Documentation

#### 7.2.1.1 production

using **production** = std::vector<std::string>

Represents the right-hand side of a grammar rule.

A production is a sequence of grammar symbols (terminals or non-terminals) that can be derived from a non-terminal symbol in the grammar.

For example, in the rule A → a B c, the production would be: {"a", "B", "c"}

## 7.3 grammar.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it

```

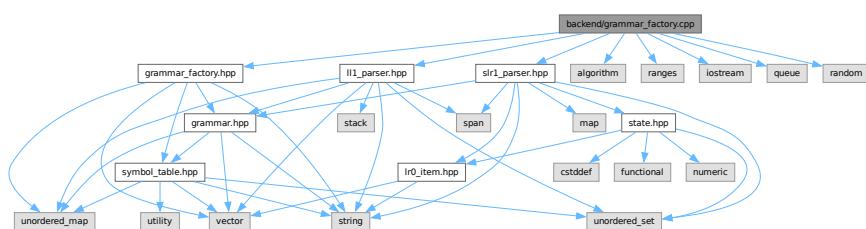
```

00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018 #pragma once
00019 #include "symbol_table.hpp"
00020 #include <string>
00021 #include <unordered_map>
00022 #include <vector>
00023
00024 using production = std::vector<std::string>;
00025
00026 struct Grammar {
00027
00028 Grammar();
00029 explicit Grammar(
00030 const std::unordered_map<std::string, std::vector<production>&>&
00031 grammar);
00032
00033 void SetAxiom(const std::string& axiom);
00034
00035 bool HasEmptyProduction(const std::string& antecedent) const;
00036
00037 std::vector<std::pair<const std::string, production>>
00038 FilterRulesByConsequent(const std::string& arg) const;
00039
00040 void Debug() const; // NOSONAR
00041
00042 void AddProduction(const std::string& antecedent,
00043 const std::vector<std::string>& consequent);
00044
00045 std::vector<std::string> Split(const std::string& s);
00046
00047 std::unordered_map<std::string, std::vector<production>> g_;
00048
00049 std::string axiom_;
00050
00051 SymbolTable st_;
00052
00053 };
00054
00055 }
```

## 7.4 backend/grammar\_factory.cpp File Reference

```

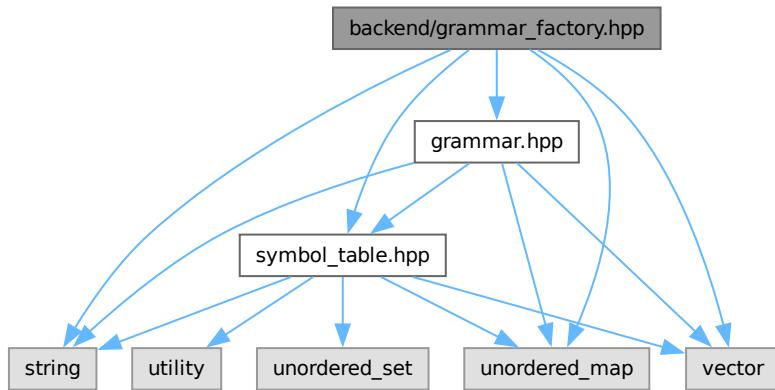
#include "grammar_factory.hpp"
#include "ll1_parser.hpp"
#include "srl1_parser.hpp"
#include <algorithm>
#include <iostream>
#include <queue>
#include <random>
#include <ranges>
Include dependency graph for grammar_factory.cpp:
```



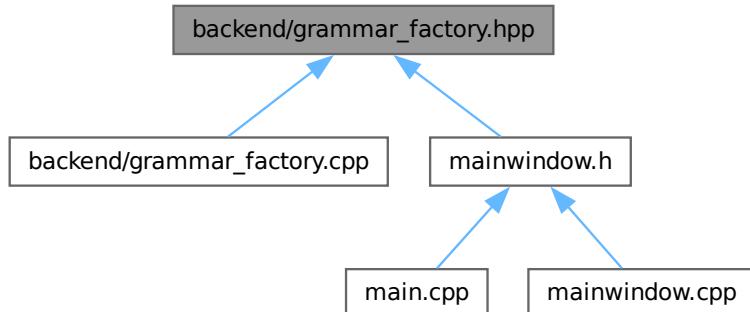
## 7.5 backend/grammar\_factory.hpp File Reference

```
#include "grammar.hpp"
#include "symbol_table.hpp"
#include <string>
#include <unordered_map>
#include <vector>
```

Include dependency graph for grammar\_factory.hpp:



This graph shows which files directly or indirectly include this file:



### Classes

- struct [GrammarFactory](#)

Responsible for creating and managing grammar items and performing checks on grammars.

- struct [GrammarFactory::FactoryItem](#)

Represents an individual grammar item with its associated symbol table.

## 7.6 grammar\_factory.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #pragma once
00020 #include "grammar.hpp"
00021 #include "symbol_table.hpp"
00022 #include <string>
00023 #include <unordered_map>
00024 #include <vector>
00025
00026 struct GrammarFactory {
00027
00028 struct FactoryItem {
00029 std::unordered_map<std::string, std::vector<production>> g_;
00030
00031 SymbolTable st_;
00032
00033 explicit FactoryItem(
00034 const std::unordered_map<std::string, std::vector<production>>&
00035 grammar);
00036 };
00037
00038 void Init();
00039
00040 Grammar PickOne(int level);
00041
00042 Grammar GenLL1Grammar(int level);
00043 Grammar GenSLR1Grammar(int level);
00044
00045 Grammar Lv1();
00046
00047 Grammar Lv2();
00048
00049 Grammar Lv3();
00050
00051 Grammar Lv4();
00052
00053 Grammar Lv5();
00054
00055 Grammar Lv6();
00056
00057 Grammar Lv7();
00058
00059 FactoryItem CreateLv2Item();
00060
00061 bool HasUnreachableSymbols(Grammar& grammar) const;
00062
00063 bool IsInfinite(Grammar& grammar) const;
00064
00065 bool HasDirectLeftRecursion(const Grammar& grammar) const;
00066
00067 bool HasIndirectLeftRecursion(const Grammar& grammar) const;
00068
00069 bool HasCycle(
00070 const std::unordered_map<std::string, std::unordered_set<std::string>>&
00071 graph) const;
00072
00073 std::unordered_set<std::string>
00074 NullableSymbols(const Grammar& grammar) const;
00075
00076
00077 void RemoveLeftRecursion(Grammar& grammar);
00078
00079 void LeftFactorize(Grammar& grammar);
00080
00081 std::vector<std::string>
00082 LongestCommonPrefix(const std::vector<production>& productions);
00083
00084 bool StartsWith(production& prod,
00085 const std::vector<std::string>& prefix);
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
010010
010011
010012
010013
010014
010015
010016
010017
010018
010019
010020
010021
010022
010023
010024
010025
010026
010027
010028
010029
010030
010031
010032
010033
010034
010035
010036
010037
010038
010039
010040
010041
010042
010043
010044
010045
010046
010047
010048
010049
010050
010051
010052
010053
010054
010055
010056
010057
010058
010059
010060
010061
010062
010063
010064
010065
010066
010067
010068
010069
010070
010071
010072
010073
010074
010075
010076
010077
010078
010079
010080
010081
010082
010083
010084
010085
010086
010087
010088
010089
010090
010091
010092
010093
010094
010095
010096
010097
010098
010099
0100100
0100101
0100102
0100103
0100104
0100105
0100106
0100107
0100108
0100109
0100110
0100111
0100112
0100113
0100114
0100115
0100116
0100117
0100118
0100119
0100120
0100121
0100122
0100123
0100124
0100125
0100126
0100127
0100128
0100129
0100130
0100131
0100132
0100133
0100134
0100135
0100136
0100137
0100138
0100139
0100140
0100141
0100142
0100143
0100144
0100145
0100146
0100147
0100148
0100149
0100150
0100151
0100152
0100153
0100154
0100155
0100156
0100157
0100158
0100159
0100160
0100161
0100162
0100163
0100164
0100165
0100166
0100167
0100168
0100169
0100170
0100171
0100172
0100173
0100174
0100175
0100176
0100177
0100178
0100179
0100180
0100181
0100182
0100183
0100184
0100185
0100186
0100187
0100188
0100189
0100190
0100191
0100192
0100193
0100194
0100195
0100196
0100197
0100198
0100199
0100200
0100201
0100202
0100203
0100204
0100205
0100206
0100207
0100208
0100209
0100210
0100211
0100212
0100213
0100214
0100215
0100216
0100217
0100218
0100219
0100220
0100221
0100222
0100223
0100224
0100225
0100226
0100227
0100228
0100229
0100230
0100231
0100232
0100233
0100234
0100235
0100236
0100237
0100238
0100239
0100240
0100241
0100242
0100243
0100244
0100245
0100246
0100247
0100248
0100249
0100250
0100251
0100252
0100253
0100254
0100255
0100256
0100257
0100258
0100259
0100260
0100261
0100262
0100263
0100264
0100265
0100266
0100267
0100268
0100269
0100270
0100271
0100272
0100273
0100274
0100275
0100276
0100277
0100278
0100279
0100280
0100281
0100282
0100283
0100284
0100285
0100286
0100287
0100288
0100289
0100290
0100291
0100292
0100293
0100294
0100295
0100296
0100297
0100298
0100299
0100300
0100301
0100302
0100303
0100304
0100305
0100306
0100307
0100308
0100309
0100310
0100311
0100312
0100313
0100314
0100315
0100316
0100317
0100318
0100319
0100320
0100321
0100322
0100323
0100324
0100325
0100326
0100327
0100328
0100329
0100330
0100331
0100332
0100333
0100334
0100335
0100336
0100337
0100338
0100339
0100340
0100341
0100342
0100343
0100344
0100345
0100346
0100347
0100348
0100349
0100350
0100351
0100352
0100353
0100354
0100355
0100356
0100357
0100358
0100359
0100360
0100361
0100362
0100363
0100364
0100365
0100366
0100367
0100368
0100369
0100370
0100371
0100372
0100373
0100374
0100375
0100376
0100377
0100378
0100379
0100380
0100381
0100382
0100383
0100384
0100385
0100386
0100387
0100388
0100389
0100390
0100391
0100392
0100393
0100394
0100395
0100396
0100397
0100398
0100399
0100400
0100401
0100402
0100403
0100404
0100405
0100406
0100407
0100408
0100409
0100410
0100411
0100412
0100413
0100414
0100415
0100416
0100417
0100418
0100419
0100420
0100421
0100422
0100423
0100424
0100425
0100426
0100427
0100428
0100429
0100430
0100431
0100432
0100433
0100434
0100435
0100436
0100437
0100438
0100439
0100440
0100441
0100442
0100443
0100444
0100445
0100446
0100447
0100448
0100449
0100450
0100451
0100452
0100453
0100454
0100455
0100456
0100457
0100458
0100459
0100460
0100461
0100462
0100463
0100464
0100465
0100466
0100467
0100468
0100469
0100470
0100471
0100472
0100473
0100474
0100475
0100476
0100477
0100478
0100479
0100480
0100481
0100482
0100483
0100484
0100485
0100486
0100487
0100488
0100489
0100490
0100491
0100492
0100493
0100494
0100495
0100496
0100497
0100498
0100499
0100500
0100501
0100502
0100503
0100504
0100505
0100506
0100507
0100508
0100509
0100510
0100511
0100512
0100513
0100514
0100515
0100516
0100517
0100518
0100519
0100520
0100521
0100522
0100523
0100524
0100525
0100526
0100527
0100528
0100529
0100530
0100531
0100532
0100533
0100534
0100535
0100536
0100537
0100538
0100539
0100540
0100541
0100542
0100543
0100544
0100545
0100546
0100547
0100548
0100549
0100550
0100551
0100552
0100553
0100554
0100555
0100556
0100557
0100558
0100559
0100560
0100561
0100562
0100563
0100564
0100565
0100566
0100567
0100568
0100569
0100570
0100571
0100572
0100573
0100574
0100575
0100576
0100577
0100578
0100579
0100580
0100581
0100582
0100583
0100584

```

```

00311 std::string GenerateNewNonTerminal(Grammar& grammar,
00312 const std::string& base);
00313
00326 void NormalizeNonTerminals(FactoryItem& item, const std::string& nt) const;
00327
00340 void AdjustTerminals(FactoryItem& base, const FactoryItem& cmb,
00341 const std::string& target_nt) const;
00342
00356 std::unordered_map<std::string, std::vector<production>>
00357 Merge(const FactoryItem& base, const FactoryItem& cmb) const;
00358
00363 std::vector<FactoryItem> items;
00364
00368 std::vector<std::string> terminal_alphabet_{"a", "b", "c", "d", "e", "f",
00369 "g", "h", "i", "j", "k", "l"};
00370
00374 std::vector<std::string> non_terminal_alphabet_{"A", "B", "C", "D",
00375 "E", "F", "G"};
00376 };

```

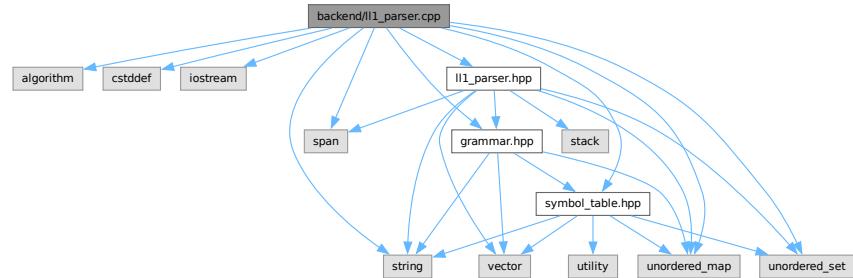
## 7.7 backend/ll1\_parser.cpp File Reference

```

#include <algorithm>
#include <cstddef>
#include <iostream>
#include
#include <string>
#include <unordered_map>
#include <unordered_set>
#include "grammar.hpp"
#include "ll1_parser.hpp"
#include "symbol_table.hpp"

```

Include dependency graph for ll1\_parser.cpp:



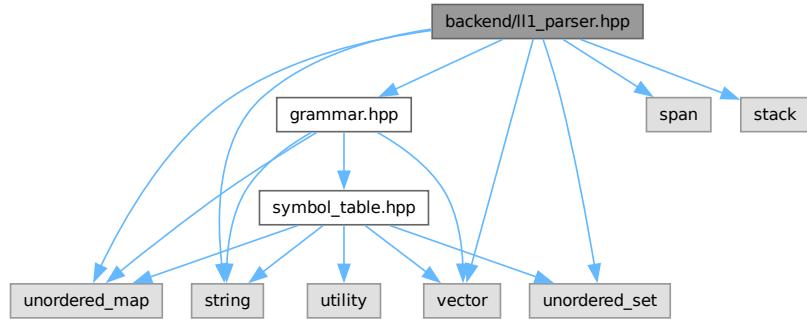
## 7.8 backend/ll1\_parser.hpp File Reference

```

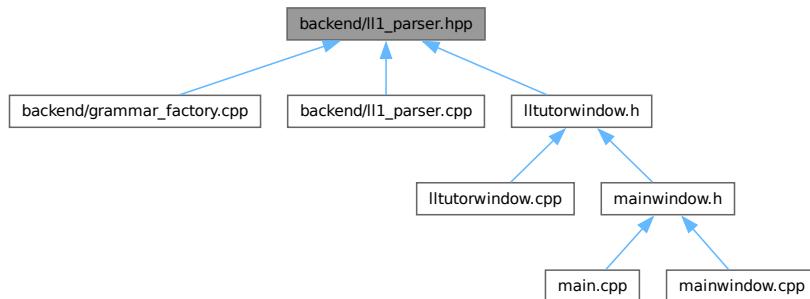
#include "grammar.hpp"
#include
#include <stack>
#include <string>
#include <unordered_map>
#include <unordered_set>
#include <vector>

```

Include dependency graph for ll1\_parser.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [LL1Parser](#)

## 7.9 ll1\_parser.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #pragma once
00020 #include "grammar.hpp"
00021 #include
00022 #include <stack>
00023 #include <string>
00024 #include <unordered_map>

```

```

00025 #include <unordered_set>
00026 #include <vector>
00027
00028 class LL1Parser {
00029
00045 using ll1_table = std::unordered_map<
00046 std::string, std::unordered_map<std::string, std::vector<production>>;
00047
00048 public:
00049 LL1Parser() = default;
00055 explicit LL1Parser(Grammar gr);
00056
00078 bool CreateLL1Table();
00079
00106 void First(std::span<const std::string> rule,
00107 std::unordered_set<std::string>& result);
00108
00119 void ComputeFirstSets();
00120
00146 void ComputeFollowSets();
00147
00162 std::unordered_set<std::string> Follow(const std::string& arg);
00163
00185 std::unordered_set<std::string>
00186 PredictionSymbols(const std::string& antecedent,
00187 const std::vector<std::string>& consequent);
00188
00191 ll1_table ll1_t_;
00192
00194 Grammar gr_;
00195
00197 std::unordered_map<std::string, std::unordered_set<std::string>>
00198 first_sets_;
00199
00201 std::unordered_map<std::string, std::unordered_set<std::string>>
00202 follow_sets_;
00203 };

```

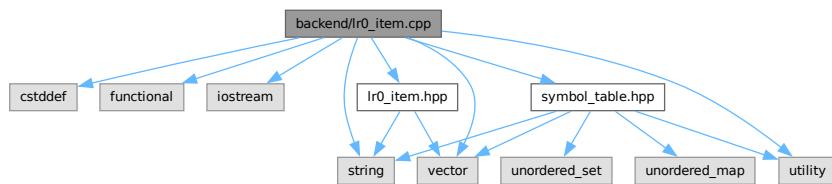
## 7.10 backend/lr0\_item.cpp File Reference

```

#include <cstddef>
#include <functional>
#include <iostream>
#include <string>
#include <utility>
#include <vector>
#include "lr0_item.hpp"
#include "symbol_table.hpp"

```

Include dependency graph for lr0\_item.cpp:



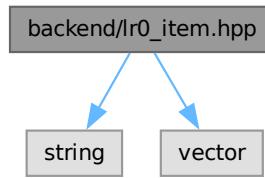
## 7.11 backend/lr0\_item.hpp File Reference

```

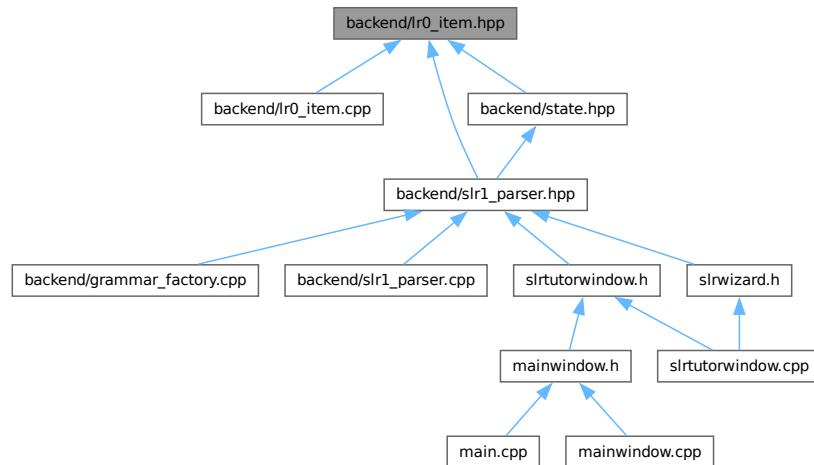
#include <string>
#include <vector>

```

Include dependency graph for lr0\_item.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct Lr0Item

Represents an LR(0) item used in LR automata construction.

## 7.12 lr0\_item.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #pragma once

```

```

00020 #include <string>
00021 #include <vector>
00022
00034
00035 struct Lr0Item {
00039 std::string antecedent_;
00040
00044 std::vector<std::string> consequent_;
00045
00049 std::string epsilon_;
00050
00054 std::string eol_;
00055
00059 unsigned int dot_ = 0;
00060
00068 Lr0Item(std::string antecedent, std::vector<std::string> consequent,
00069 std::string epsilon, std::string eol);
00070
00079 Lr0Item(std::string antecedent, std::vector<std::string> consequent,
00080 unsigned int dot, std::string epsilon, std::string eol);
00081
00087 std::string NextToDot() const;
00088
00093 void PrintItem() const;
00094
00100 std::string ToString() const;
00101
00105 void AdvanceDot();
00106
00111 bool IsComplete() const;
00112
00118 bool operator==(const Lr0Item& other) const;
00119 };
00120
00121 namespace std {
00122 template <> struct hash<Lr0Item> {
00123 size_t operator()(const Lr0Item& item) const;
00124 };
00125 } // namespace std

```

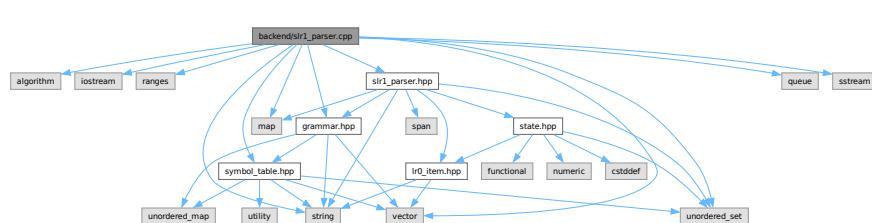
## 7.13 backend/slr1\_parser.cpp File Reference

```

#include <algorithm>
#include <iostream>
#include <map>
#include <queue>
#include <ranges>
#include <sstream>
#include <string>
#include <unordered_set>
#include <vector>
#include "grammar.hpp"
#include "slr1_parser.hpp"
#include "symbol_table.hpp"

```

Include dependency graph for slr1\_parser.cpp:



## 7.14 backend/slr1\_parser.hpp File Reference

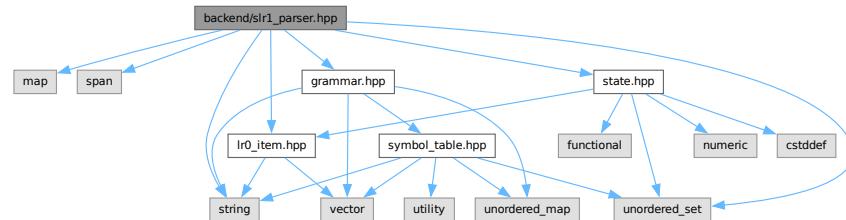
```

#include <map>
#include

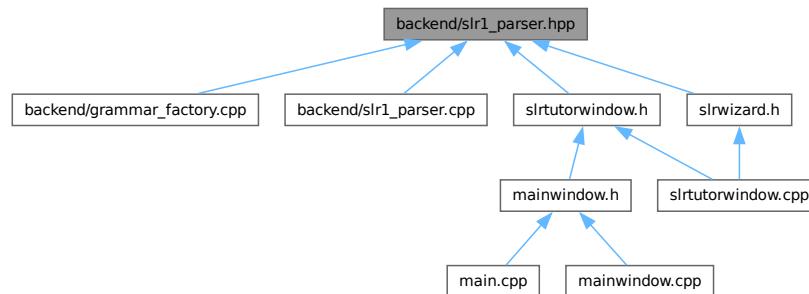
```

```
#include <string>
#include <unordered_set>
#include "grammar.hpp"
#include "lr0_item.hpp"
#include "state.hpp"
```

Include dependency graph for slr1\_parser.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- class [SLR1Parser](#)  
Implements an SLR(1) parser for context-free grammars.
- struct [SLR1Parser::s\\_action](#)

## 7.15 slr1\_parser.hpp

[Go to the documentation of this file.](#)

```
00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018 #pragma once
00019 #include <map>
```

```

00021 #include
00022 #include <string>
00023 #include <unordered_set>
00024
00025 #include "grammar.hpp"
00026 #include "lr0_item.hpp"
00027 #include "state.hpp"
00028
00029 class SLR1Parser {
00030 public:
00031 enum class Action { Shift, Reduce, Accept, Empty };
00032
00033 struct s_action {
00034 const Lr0Item* item;
00035 Action action;
00036 };
00037
00038 using action_table =
00039 std::map<unsigned int, std::map<std::string, SLR1Parser::s_action>>;
00040
00041 using transition_table =
00042 std::map<unsigned int, std::map<std::string, unsigned int>>;
00043
00044 SLR1Parser() = default;
00045 explicit SLR1Parser(Grammar gr);
00046
00047 std::unordered_set<Lr0Item> AllItems() const;
00048
00049 void Closure(std::unordered_set<Lr0Item>& items);
00050
00051 void ClosureUtil(std::unordered_set<Lr0Item>& items, unsigned int size,
00052 std::unordered_set<std::string>& visited);
00053
00054 std::unordered_set<Lr0Item> Delta(const std::unordered_set<Lr0Item>& items,
00055 const std::string& str);
00056
00057 bool SolveLRConflicts(const state& st);
00058
00059 void First(std::span<const std::string> rule,
00060 std::unordered_set<std::string>& result);
00061
00062 void ComputeFirstSets();
00063
00064 void ComputeFollowSets();
00065
00066 std::unordered_set<std::string> Follow(const std::string& arg);
00067
00068 void MakeInitialState();
00069
00070 bool MakeParser();
00071
00072 std::string PrintItems(const std::unordered_set<Lr0Item>& items) const;
00073
00074 Grammar gr_;
00075
00076 std::unordered_map<std::string, std::unordered_set<std::string>>
00077 first_sets_;
00078
00079 std::unordered_map<std::string, std::unordered_set<std::string>>
00080 follow_sets_;
00081
00082 action_table actions_;
00083
00084 transition_table transitions_;
00085
00086 std::unordered_set<state> states_;
00087 };

```

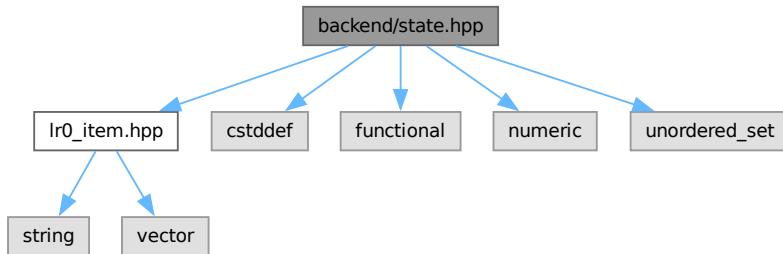
## 7.16 backend/state.hpp File Reference

```

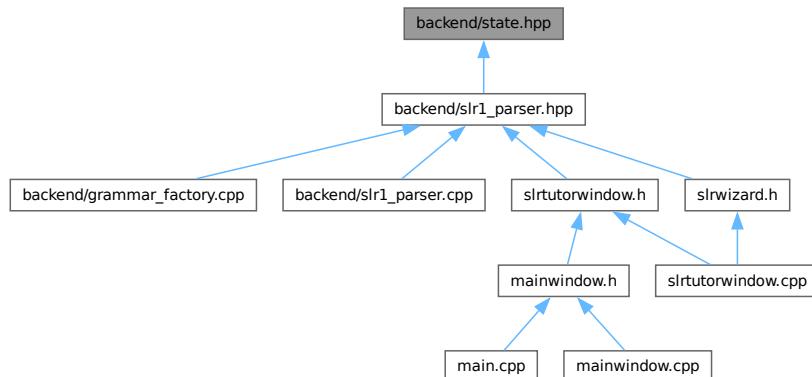
#include "lr0_item.hpp"
#include <cstddef>
#include <functional>
#include <numeric>
#include <unordered_set>

```

Include dependency graph for state.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct `state`

Represents a state in the LR(0) automaton.

## 7.17 state.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #pragma once
00020 #include "lr0_item.hpp"

```

```

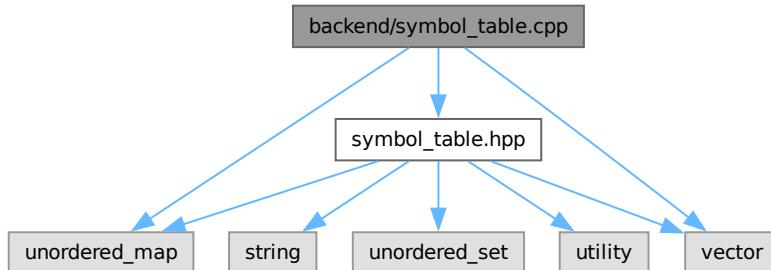
00021 #include <cstddef>
00022 #include <functional>
00023 #include <numeric>
00024 #include <unordered_set>
00025
00033 struct state {
00037 std::unordered_set<Lr0Item> items_;
00038
00042 unsigned int id_;
00043
00049 bool operator==(const state& other) const { return other.items_ == items_; }
00050 };
00051
00052 namespace std {
00053 template <> struct hash<state> {
00054 size_t operator()(const state& st) const {
00055 size_t seed =
00056 std::accumulate(st.items_.begin(), st.items_.end(), 0,
00057 [(size_t acc, const Lr0Item& item) {
00058 return acc ^ (std::hash<Lr0Item>()(item));
00059 });
00060 return seed;
00061 }
00062 };
00063 } // namespace std

```

## 7.18 backend/symbol\_table.cpp File Reference

```
#include "symbol_table.hpp"
#include <unordered_map>
#include <vector>
```

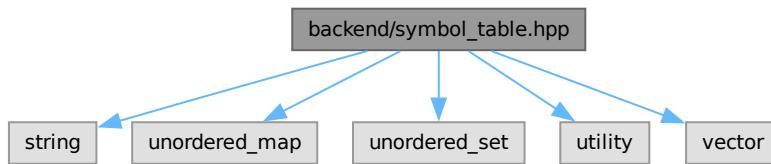
Include dependency graph for symbol\_table.cpp:



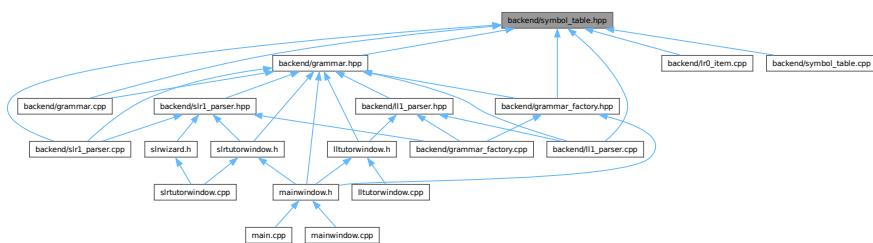
## 7.19 backend/symbol\_table.hpp File Reference

```
#include <string>
#include <unordered_map>
#include <unordered_set>
#include <utility>
#include <vector>
```

Include dependency graph for symbol\_table.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct **SymbolTable**

Stores and manages grammar symbols, including their classification and special markers.

## Enumerations

- enum class **symbol\_type** { **NO\_TERMINAL** , **TERMINAL** }

Represents the type of a grammar symbol.

### 7.19.1 Enumeration Type Documentation

#### 7.19.1.1 symbol\_type

enum class **symbol\_type** [strong]

Represents the type of a grammar symbol.

This enum distinguishes between terminal and non-terminal symbols within the grammar and the symbol table.

Enumerator

|                    |  |
|--------------------|--|
| <b>NO_TERMINAL</b> |  |
| <b>TERMINAL</b>    |  |

## 7.20 symbol\_table.hpp

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)

```

```

00004 *
00005 * This program is free software; you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #pragma once
00020 #include <string>
00021 #include <unordered_map>
00022 #include <unordered_set>
00023 #include <utility>
00024 #include <vector>
00025
00026 enum class symbol_type { NO_TERMINAL, TERMINAL };
00027
00028 struct SymbolTable {
00029 std::string EOL_{"$"};
00030
00031 std::string EPSILON_("EPSILON");
00032
00033 std::unordered_map<std::string, symbol_type> st_{
00034 {EOL_, symbol_type::TERMINAL}, {EPSILON_, symbol_type::TERMINAL}};
00035
00036 std::unordered_set<std::string> terminals_{EOL_};
00037
00038 std::unordered_set<std::string> terminals_wtho_eol_{};
00039
00040 std::unordered_set<std::string> non_terminals_;
00041
00042 void PutSymbol(const std::string& identifier, bool isTerminal);
00043
00044 bool In(const std::string& s) const;
00045
00046 bool IsTerminal(const std::string& s) const;
00047
00048 bool IsTerminalWthoEol(const std::string& s) const;
00049
00050 };
00051
00052
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104 };

```

## 7.21 CHANGELOG.md File Reference

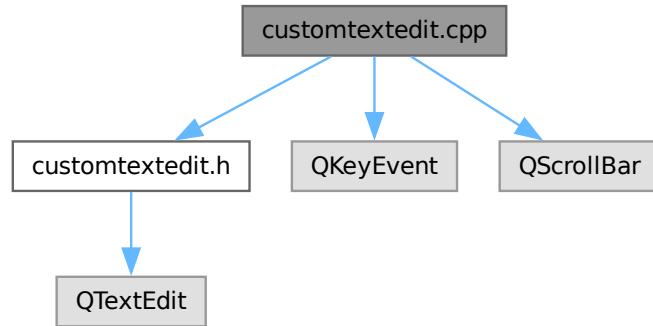
## 7.22 customtextedit.cpp File Reference

```

#include "customtextedit.h"
#include <QKeyEvent>
#include <QScrollBar>

```

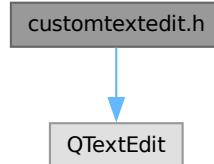
Include dependency graph for customtextedit.cpp:



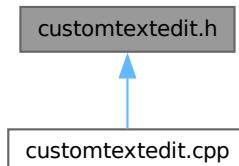
## 7.23 customtextedit.h File Reference

```
#include <QTextEdit>
```

Include dependency graph for customtextedit.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [CustomTextEdit](#)

## 7.24 customtextedit.h

[Go to the documentation of this file.](#)

```

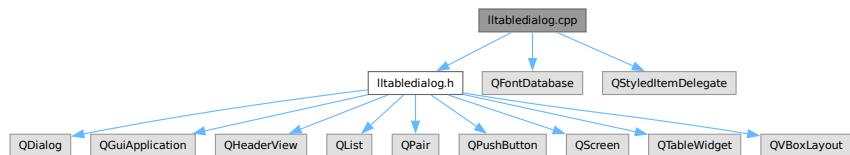
00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #ifndef CUSTOMTEXTEDIT_H
00020 #define CUSTOMTEXTEDIT_H
00021
00022 #include <QTextEdit>
00023
00024 class CustomTextEdit : public QTextEdit {
00025 Q_OBJECT
00026 public:
00027 explicit CustomTextEdit(QWidget* parent = nullptr);
00028
00029 signals:
00030 void sendRequested();
00031
00032 protected:
00033 void keyPressEvent(QKeyEvent* event) override;
00034 };
00035
00036 #endif // CUSTOMTEXTEDIT_H

```

## 7.25 lltabledialog.cpp File Reference

```
#include "lltabledialog.h"
#include <QFontDatabase>
#include <QStyledItemDelegate>
```

Include dependency graph for lltabledialog.cpp:



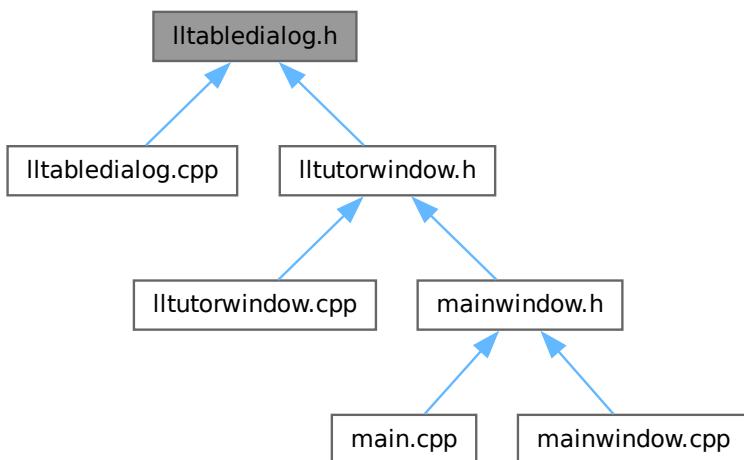
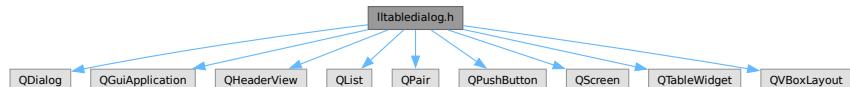
### Classes

- class [CenterAlignDelegate](#)

## 7.26 lltabledialog.h File Reference

```
#include <QDialog>
#include <QGuiApplication>
#include <QHeaderView>
#include <QList>
#include <QPair>
#include <QPushButton>
#include <QScreen>
```

```
#include <QTableWidget>
#include <QVBoxLayout>
Include dependency graph for lltabledialog.h:
```



## Classes

- class [LLTableDialog](#)  
Dialog for filling and submitting an LL(1) parsing table.

## 7.27 lltabledialog.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #ifndef LLTABLEDIALOG_H
00020 #define LLTABLEDIALOG_H
00021

```

```

00022 #include <QDialog>
00023 #include <QGuiApplication>
00024 #include <QHeaderView>
00025 #include <QList>
00026 #include <QPair>
00027 #include <QPushButton>
00028 #include <QScreen>
00029 #include <QTableWidget>
00030 #include <QVBoxLayout>
00031
00041 class LLTableDialog : public QDialog {
00042 Q_OBJECT
00043 public:
00052 LLTableDialog(const QStringList& rowHeaders, const QStringList& colHeaders,
00053 QWidget* parent,
00054 QVector<QVector<QString>>* initialData = nullptr);
00055
00060 QVector<QVector<QString>> getTableData() const;
00061
00070 void setInitialData(const QVector<QVector<QString>& data);
00071
00076 void highlightIncorrectCells(const QList<QPair<int, int>& coords);
00077
00078 signals:
00083 void submitted(const QVector<QVector<QString>& data);
00084
00085 private:
00086 QTableWidget* table;
00087 QPushButton* submitButton;
00088 };
00089
00090 #endif // LLTABLEDIALOG_H

```

## 7.28 lltutorwindow.cpp File Reference

```

#include "lltutorwindow.h"
#include "tutorialmanager.h"
#include "ui_lltutorwindow.h"
#include <QAbstractButton>
#include <QFontDatabase>
#include <QRandomGenerator>
#include <QRegularExpression>
#include <QWheelEvent>

```

Include dependency graph for lltutorwindow.cpp:



## 7.29 lltutorwindow.h File Reference

```

#include <QAbstractItemView>
#include <QDialog>
#include <QFileDialog>
#include <QGraphicsColorizeEffect>
#include <QGraphicsScene>
#include <QGraphicsTextItem>
#include <QGraphicsView>
#include <QListWidgetItem>
#include <QMainWindow>
#include <QMessageBox>
#include <QPainter>
#include <QPropertyAnimation>
#include <QPushButton>
#include <QScrollBar>
#include <QShortcut>

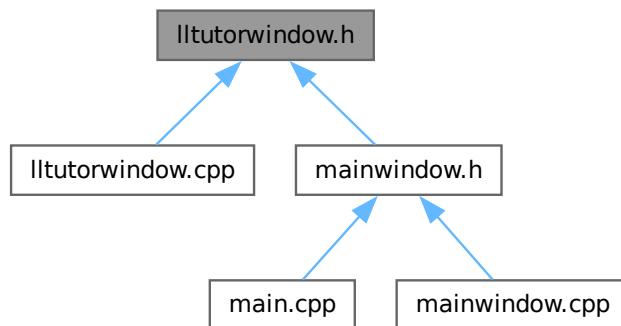
```

```
#include <QTableWidget>
#include <QTextDocument>
#include <QTextEdit>
#include <QTime>
#include <QTimer>
#include <QTreeWidgetItem>
#include <QVBoxLayout>
#include <QtPrintSupport/QPrinter>
#include "backend/grammar.hpp"
#include "backend/ll1_parser.hpp"
#include "lltabledialog.h"
```

Include dependency graph for lltutorwindow.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [LLTutorWindow](#)  
Main window for the LL(1) interactive tutoring mode in SyntaxTutor.
- struct [LLTutorWindow::TreeNode](#)  
[TreeNode](#) structure used to build derivation trees.

## Enumerations

- enum class [State](#) {
 [A](#) , [A1](#) , [A2](#) , [A\\_prime](#) ,
 [B](#) , [B1](#) , [B2](#) , [B\\_prime](#) ,
 [C](#) , [C\\_prime](#) , [fin](#) }

### 7.29.1 Enumeration Type Documentation

#### 7.29.1.1 State

enum class [State](#) [strong]

Enumerator

|         |  |
|---------|--|
| A       |  |
| A1      |  |
| A2      |  |
| A_prime |  |
| B       |  |
| B1      |  |
| B2      |  |
| B_prime |  |
| C       |  |
| C_prime |  |
| fin     |  |

## 7.30 lltutorwindow.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #ifndef LLTUTORWINDOW_H
00020 #define LLTUTORWINDOW_H
00021
00022 #include <QAbstractItemView>
00023 #include <QDialog>
00024 #include <QFileDialog>
00025 #include <QGraphicsColorizeEffect>
00026 #include <QGraphicsScene>
00027 #include <QGraphicsTextItem>
00028 #include <QGraphicsView>
00029 #include <QListWidgetItem>
00030 #include <QMainWindow>
00031 #include <QMessageBox>
00032 #include <QPainter>
00033 #include <QPropertyAnimation>
00034 #include <QPushButton>
00035 #include <QScrollBar>
00036 #include <QShortcut>
00037 #include <QTableWidget>
00038 #include <QTextDocument>
00039 #include <QTextEdit>
00040 #include <QTime>
00041 #include <QTimer>
00042 #include <QTreeWidgetItem>
00043 #include <QVBoxLayout>
00044 #include <QtPrintSupport/QPrinter>
00045
00046 #include "backend/grammar.hpp"
00047 #include "backend/ll1_parser.hpp"
00048 #include "lltabledialog.h"
00049
00050 class TutorialManager;
00051
00052 namespace Ui {
00053 class LLTutorWindow;
00054 }
00055
00056 // ===== LL(1) Tutor States =====

```

```

00057 enum class State { A, A1, A2, A_prime, B, B1, B2, B_prime, C, C_prime, fin };
00058
00059 // ===== LL(1) Tutor Main Class =====
00079 class LLTutorWindow : public QMainWindow {
00080 Q_OBJECT
00081
00082 public:
00083 // ===== Derivation Tree (used in TeachFirst) =====
00084 struct TreeNode {
00085 QString label;
00086 std::vector<std::unique_ptr<TreeNode>> children;
00087 };
00088
00089 // ===== Constructor / Destructor =====
00090 explicit LLTutorWindow(const Grammar& grammar,
00091 TutorialManager* tm = nullptr,
00092 QWidget* parent = nullptr);
00093 ~LLTutorWindow();
00094
00095 // ===== State Machine & Question Logic =====
00096 QString generateQuestion();
00097
00098 void updateState(bool isCorrect);
00099
00100 QString FormatGrammar(const Grammar& grammar);
00101
00102 // ===== UI Interaction =====
00103 void addMessage(const QString& text,
00104 bool isUser);
00105 void addWidgetMessage(QWidget* widget);
00106 void exportConversationToPdf(const QString& filePath);
00107 void showTable();
00108 void showTableForCPrime();
00109 void updateProgressPanel(); // Update progress panel
00110
00111 // ===== Visual Feedback / Animations =====
00112 void animateLabelPop(QLabel* label);
00113 void animateLabelColor(QLabel* label, const QColor& flashColor);
00114 void wrongAnimation();
00115 void wrongUserResponseAnimation();
00116 void markLastUserIncorrect();
00117
00118 // ===== Tree Generation (TeachFirst mode) =====
00119 void TeachFirstTree(std::vector<std::string>& symbols,
00120 std::unordered_set<std::string>& first_set, int depth,
00121 std::unordered_set<std::string>& processing,
00122 QTreeWidgetItem* parent);
00123
00124 std::unique_ptr<TreeNode>
00125 buildTreeNode(const std::vector<std::string>& symbols,
00126 std::unordered_set<std::string>& first_set, int depth,
00127 std::vector<std::pair<std::string, std::vector<std::string>>&
00128 active_derivations);
00129
00130 int computeSubtreeWidth(const std::unique_ptr<TreeNode>& node,
00131 int hSpacing);
00132 void drawTree(const std::unique_ptr<TreeNode>& root, QGraphicsScene* scene,
00133 QPointF pos, int hSpacing, int vSpacing);
00134
00135 void showTreeGraphics(
00136 std::unique_ptr<TreeNode> root); // Display derivation tree visually
00137
00138 // ===== User Response Verification =====
00139 bool verifyResponse(const QString& userResponse); // Delegates to current
00140 // state's verification
00141 bool verifyResponseForA(const QString& userResponse);
00142 bool verifyResponseForA1(const QString& userResponse);
00143 bool verifyResponseForA2(const QString& userResponse);
00144 bool verifyResponseForB(const QString& userResponse);
00145 bool verifyResponseForB1(const QString& userResponse);
00146 bool verifyResponseForB2(const QString& userResponse);
00147 bool verifyResponseForC(); // C is non-textual (checks internal table)
00148
00149 // ===== Expected Solutions (Auto-generated) =====
00150 QString solution(const std::string& state);
00151 QStringList solutionForA();
00152 QString solutionForA1();
00153 QString solutionForA2();
00154 QSet<QString> solutionForB();
00155 QSet<QString> solutionForB1();
00156 QSet<QString> solutionForB2();
00157
00158 // ===== Feedback (Corrective Explanations) =====
00159 QString feedback(); // Delegates by state
00160 QString feedbackForA();

```

```

00185 QString feedbackForA1();
00186 QString feedbackForA2();
00187 QString feedbackForAPrime();
00188 QString feedbackForB();
00189 QString feedbackForB1();
00190 QString feedbackForB2();
00191 QString feedbackForBPrime();
00192 QString feedbackForC();
00193 QString feedbackForCPrime();
00194 void feedbackForB1TreeWidget(); // TreeWidget of Teach (LL1 TeachFirst)
00195 void feedbackForB1TreeGraphics(); // Show derivation tree
00196 QString TeachFollow(const QString& nt);
00197 QString TeachPredictionSymbols(const QString& ant,
00198 const production& consequ);
00199 QString TeachLL1Table();
00200
00201 void handleTableSubmission(const QVector<QString>& raw,
00202 const QStringList& colHeaders);
00203 private slots:
00204 void on_confirmButton_clicked();
00205 void on_userResponseTextChanged();
00206
00207 signals:
00208 void sessionFinished(int cntRight, int cntWrong);
00209
00210 protected:
00211 void closeEvent(QCloseEvent* event) override {
00212 emit sessionFinished(cntRightAnswers, cntWrongAnswers);
00213 QWidget::closeEvent(event);
00214 }
00215
00216 bool eventFilter(QObject* obj, QEvent* event) override;
00217
00218 private:
00219 // ===== Core Objects =====
00220 Ui::LLTutorWindow* ui;
00221 Grammar grammar;
00222 LL1Parser ll1;
00223
00224 // ===== State & Grammar Tracking =====
00225 State currentState;
00226 size_t currentRule = 0;
00227 const unsigned kMaxHighlightTries = 3;
00228 const unsigned kMaxTotalTries = 5;
00229 unsigned lltries = 0;
00230 unsigned cntRightAnswers = 0, cntWrongAnswers = 0;
00231
00232 using Cell = std::pair<QString, QString>;
00233 std::vector<Cell> lastWrongCells;
00234 LLTableDialog* currentDlg = nullptr;
00235
00236 QVector<QString> sortedNonTerminals;
00237 QVector<QPair<QString, QVector<QString>>> sortedGrammar;
00238 QString formattedGrammar;
00239
00240 QMap<QString, QMap<QString, QVector<QString>>> lltable;
00241 QVector<QVector<QString>> rawTable;
00242 QSet<QString> solutionSet;
00243
00244 // ===== Conversation Logging =====
00245 struct MessageLog {
00246 QString message;
00247 bool isUser;
00248 bool isCorrect = true;
00249 MessageLog(const QString& message, bool isUser)
00250 : message(message), isUser(isUser) {}
00251 void toggleIsCorrect() { isCorrect = !isCorrect; }
00252 };
00253
00254 QVector<MessageLog> conversationLog;
00255 QWidget* lastUserMessage = nullptr;
00256 qsizetype lastUserMessageLogIdx = -1;
00257
00258 QMap<QString, QString> userCAB;
00259 QMap<QString, QString> userSIG;
00260 QMap<QString, QString> userSD;
00261
00262 // ===== Helper Conversions =====
00263 std::vector<std::string> qvectorToStdVector(const QVector<QString>& qvec);
00264 QVector<QString> stdVectorToQVector(const std::vector<std::string>& vec);
00265 QSet<QString>
00266 stdUnorderedSetToQSet(const std::unordered_set<std::string>& uset);
00267 std::unordered_set<std::string>
00268 qsetToStdUnorderedSet(const QSet<QString>& qset);
00269
00270 void setupTutorial();
00271

```

```

00272 void
00273 fillSortedGrammar(); // Populate sortedGrammar from internal representation
00274
00275 QPropertyAnimation* m_shakeAnimation =
00276 nullptr; // For interrupting userResponse animation if they spam enter
00277 // key
00278
00279 TutorialManager* tm = nullptr;
00280
00281 QRegularExpression re{ "^\s+|\s+$"};
00282 };
00283
00284 #endif // LLTUTORWINDOW_H

```

## 7.31 main.cpp File Reference

```
#include "mainwindow.h"
#include <QApplication>
#include <QFont>
#include <QFontDatabase>
#include <QImageReader>
#include <QSettings>
#include <QTranslator>
```

Include dependency graph for main.cpp:



### Functions

- void `loadFonts ()`
- int `main (int argc, char *argv[])`

#### 7.31.1 Function Documentation

##### 7.31.1.1 `loadFonts()`

`void loadFonts ()`

Here is the caller graph for this function:

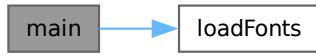


##### 7.31.1.2 `main()`

```
int main (
 int argc,
 char * argv[])

```

Here is the call graph for this function:



## 7.32mainwindow.cpp File Reference

```
#include "mainwindow.h"
#include "tutorialmanager.h"
#include "ui_mainwindow.h"
#include <QMessageBox>
#include <QPixmap>
#include <QProcess>
Include dependency graph for mainwindow.cpp:
```

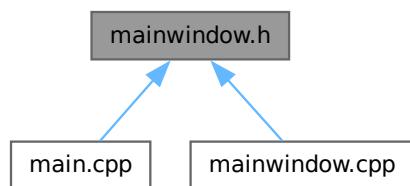


## 7.33mainwindow.h File Reference

```
#include "backend/grammar.hpp"
#include "backend/grammar_factory.hpp"
#include "lltutorwindow.h"
#include "slrtutorwindow.h"
#include "tutorialmanager.h"
#include < QMainWindow>
#include < QSettings>
Include dependency graph for mainwindow.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [MainWindow](#)

Main application window of SyntaxTutor, managing levels, exercises, and UI state.

## 7.34 mainwindow.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #ifndef MAINWINDOW_H
00020 #define MAINWINDOW_H
00021
00022 #include "backend/grammar.hpp"
00023 #include "backend/grammar_factory.hpp"
00024 #include "littutorwindow.h"
00025 #include "slrtutorwindow.h"
00026 #include "tutorialmanager.h"
00027 #include <QMainWindow>
00028 #include <QSettings>
00029
00030 static const QVector<QString> levelColors = {
00031 "#2C3E50", // 1: Navy oscuro
00032 "#2980B9", // 2: Azul brillante
00033 "#16A085", // 3: Teal
00034 "#27AE60", // 4: Verde esmeralda
00035 "#8E44AD", // 5: Púrpura medio
00036 "#9B59B6", // 6: Púrpura claro
00037 "#E67E22", // 7: Naranja
00038 "#D35400", // 8: Naranja oscuro
00039 "#CD7F32", // 9: Bronce
00040 "#FFD700" // 10: Oro puro
00041 };
00042
00043 QT_BEGIN_NAMESPACE
00044 namespace Ui {
00045 class MainWindow;
00046 }
00047 QT_END_NAMESPACE
00048
00049 class MainWindow : public QMainWindow {
00050 Q_OBJECT
00051 Q_PROPERTY(unsigned userLevel READ userLevel WRITE setUserLevel NOTIFY
00052 userLevelChanged)
00053
00054 public:
00055 MainWindow(QWidget* parent = nullptr);
00056 ~MainWindow();
00057
00058 unsigned thresholdFor(unsigned level) { return BASE_THRESHOLD * level; }
00059
00060 unsigned userLevel() const { return m_userLevel; }
00061
00062 void setUserLevel(unsigned lvl) {
00063 unsigned clamped = qMin(lvl, MAX_LEVEL);
00064 if (m_userLevel == clamped)
00065 return;
00066 m_userLevel = clamped;
00067 emit userLevelChanged(clamped);
00068 }
00069
00070 private slots:
00071 void on_lv1Button_clicked(bool checked);
00072 void on_lv2Button_clicked(bool checked);
00073 void on_lv3Button_clicked(bool checked);
00074
00075 }
```

```

00109 void on_pushButton_clicked();
00110
00114 void on_pushButton_2_clicked();
00115
00119 void on_tutorial_clicked();
00120
00124 void on_actionSobre_la_aplicaci_n_triggered();
00125
00129 void on_actionReferencia_LL_1_triggered();
00130
00134 void on_actionReferencia_SLR_1_triggered();
00135
00139 void on_idiom_clicked();
00140
00141 signals:
00146 void userLevelChanged(unsigned lvl);
00147
00152 void userLevelUp(unsigned newLevel);
00153
00154 private:
00158 void setupTutorial();
00159
00163 void restartTutorial();
00164
00170 void handleTutorFinished(int cntRight, int cntWrong);
00171
00175 void saveSettings();
00176
00180 void loadSettings();
00181
00182 Ui::MainWindow* ui;
00183 GrammarFactory factory;
00184 int level = 1;
00185 TutorialManager* tm = nullptr;
00186
00187 static constexpr unsigned MAX_LEVEL = 10;
00188 static constexpr unsigned MAX_SCORE = 999;
00189
00190 unsigned m_userLevel = 1;
00191 unsigned userScore = 0;
00192 QSettings settings;
00193
00194 const unsigned BASE_THRESHOLD = 10;
00195 };
00196 #endif // MAINWINDOW_H

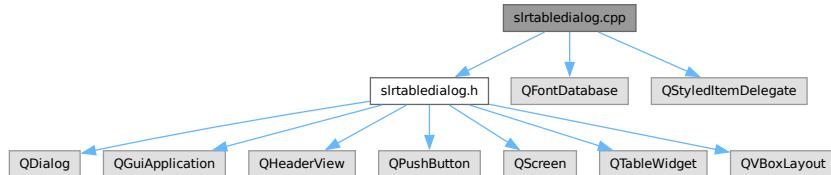
```

## 7.35 README.md File Reference

## 7.36 slrtabledialog.cpp File Reference

```
#include "slrtabledialog.h"
#include <QFontDatabase>
#include <QStyledItemDelegate>
```

Include dependency graph for slrtabledialog.cpp:

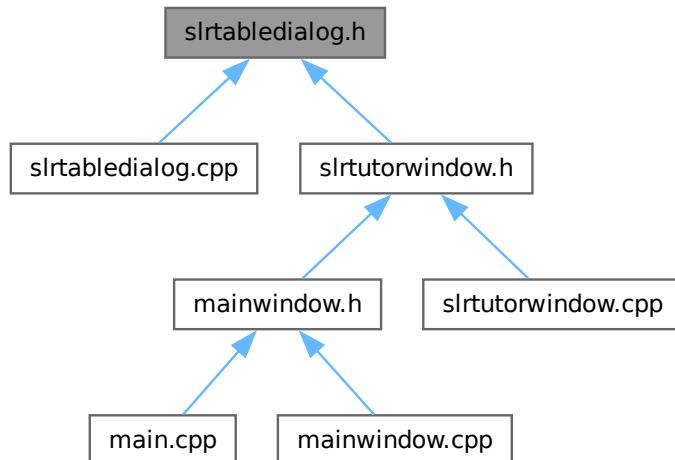
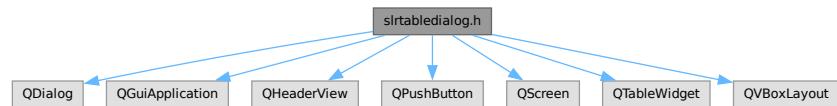


## Classes

- class [CenterAlignDelegate](#)

## 7.37 slrtabledialog.h File Reference

```
#include <QDialog>
#include <QGuiApplication>
#include <QHeaderView>
#include <QPushButton>
#include <QScreen>
#include <QTableWidget>
#include <QVBoxLayout>
Include dependency graph for slrtabledialog.h:
```



### Classes

- class [SLRTTableDialog](#)  
Dialog window for completing and submitting an SLR(1) parsing table.

## 7.38 slrtabledialog.h

[Go to the documentation of this file.](#)

```
00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
```

```

00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #ifndef SLRTABLEDIALOG_H
00020 #define SLRTABLEDIALOG_H
00021
00022 #include <QDialog>
00023 #include <QGuiApplication>
00024 #include <QHeaderView>
00025 #include <QPushButton>
00026 #include <QScreen>
00027 #include <QTableWidget>
00028 #include <QVBoxLayout>
00029
00030
00031 class SLRTableDialog : public QDialog {
00032 Q_OBJECT
00033 public:
00034 SLRTableDialog(int rowCount, int colCount, const QStringList& colHeaders,
00035 QWidget* parent = nullptr,
00036 QVector<QVector<QString>>* initialData = nullptr);
00037
00038 QVector<QVector<QString>> getTableData() const;
00039
00040 void setInitialData(const QVector<QVector<QString>& data);
00041
00042 private:
00043 QTableWidget* table;
00044 QPushButton* submitButton;
00045 };
00046
00047 #endif // SLRTABLEDIALOG_H

```

## 7.39 slrtutorwindow.cpp File Reference

```

#include "slrtutorwindow.h"
#include "tutorialmanager.h"
#include "ui_slrtutorwindow.h"
#include <QEasingCurve>
#include <QFontDatabase>
#include <sstream>
#include "slrwizard.h"

```

Include dependency graph for slrtutorwindow.cpp:



## 7.40 slrtutorwindow.h File Reference

```

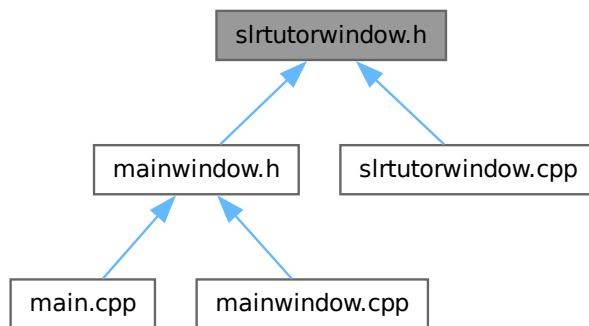
#include "UniqueQueue.h"
#include "backend/grammar.hpp"
#include "backend/slrl_parser.hpp"
#include "slrtabledialog.h"
#include <QAbstractItemView>
#include <QDialog>
#include <QFileDialog>
#include <QGraphicsColorizeEffect>
#include <QListWidgetItem>
#include <QMainWindow>
#include <QMessageBox>
#include <QPropertyAnimation>
#include <QPushButton>

```

```
#include <QRegularExpression>
#include <QScrollBar>
#include <QShortcut>
#include <QTableWidget>
#include <QTextDocument>
#include <QTextEdit>
#include <QTime>
#include <QTimer>
#include <QVBoxLayout>
#include <QtPrintSupport/QPrinter>
Include dependency graph for slrtutorwindow.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [SLRTutorWindow](#)

Main window for the SLR(1) interactive tutoring mode in SyntaxTutor.

## Enumerations

- enum class [StateSlr](#) {
 [A](#) , [A1](#) , [A2](#) , [A3](#) ,
 [A4](#) , [A\\_prime](#) , [B](#) , [C](#) ,
 [CA](#) , [CB](#) , [D](#) , [D1](#) ,
 [D2](#) , [D\\_prime](#) , [E](#) , [E1](#) ,
 [E2](#) , [F](#) , [FA](#) , [G](#) ,
 [H](#) , [H\\_prime](#) , [fin](#) }

### 7.40.1 Enumeration Type Documentation

#### 7.40.1.1 StateSlr

enum class [StateSlr](#) [strong]

## Enumerator

|         |  |
|---------|--|
| A       |  |
| A1      |  |
| A2      |  |
| A3      |  |
| A4      |  |
| A_prime |  |
| B       |  |
| C       |  |
| CA      |  |
| CB      |  |
| D       |  |
| D1      |  |
| D2      |  |
| D_prime |  |
| E       |  |
| E1      |  |
| E2      |  |
| F       |  |
| FA      |  |
| G       |  |
| H       |  |
| H_prime |  |
| fin     |  |

## 7.41 slrtutorwindow.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #ifndef SLRTUTORWINDOW_H
00020 #define SLRTUTORWINDOW_H
00021
00022 #include "UniqueQueue.h"
00023 #include "backend/grammar.hpp"
00024 #include "backend/slr1_parser.hpp"
00025 #include "slrtabledialog.h"
00026 #include <QAbstractItemView>
00027 #include <QDialog>
00028 #include <QFileDialog>
00029 #include <QGraphicsColorizeEffect>
00030 #include <QListWidgetItem>
00031 #include <QMainWindow>
00032 #include <QMessageBox>
00033 #include <QPropertyAnimation>
00034 #include <QPushButton>
00035 #include <QRegularExpression>
00036 #include <QScrollBar>
```

```

00037 #include <QShortcut>
00038 #include <QTableWidget>
00039 #include <QTextDocument>
00040 #include <QTextEdit>
00041 #include <QTime>
00042 #include <QTimer>
00043 #include <QVBoxLayout>
00044 #include <QtPrintSupport/QPrinter>
00045
00046 namespace Ui {
00047 class SLRTutorWindow;
00048 }
00049
00050 // ===== SLR(1) Tutor States =====
00051 enum class StateSlr {
00052 A,
00053 A1,
00054 A2,
00055 A3,
00056 A4,
00057 A_prime,
00058 B,
00059 C,
00060 CA,
00061 CB,
00062 D,
00063 D1,
00064 D2,
00065 D_prime,
00066 E,
00067 E1,
00068 E2,
00069 F,
00070 FA,
00071 G,
00072 H,
00073 H_prime,
00074 fin
00075 };
00076
00077 class TutorialManager;
00078
00079 // ===== Main Tutor Class for SLR(1) =====
00094 class SLRTutorWindow : public QMainWindow {
00095 Q_OBJECT
00096
00097 public:
00098 // ===== Constructor / Destructor =====
00105 explicit SLRTutorWindow(const Grammar& g, TutorialManager* tm = nullptr,
00106 QWidget* parent = nullptr);
00107 ~SLRTutorWindow();
00108
00109 // ===== Core Flow Control =====
00114 QString generateQuestion();
00115
00120 void updateState(bool isCorrect);
00121 QString
00122 FormatGrammar(const Grammar& grammar);
00123 void fillSortedGrammar();
00124
00125 // ===== UI Interaction =====
00126 void addMessage(const QString& text, bool isUser);
00127 void exportConversationToPdf(
00128 const QString& filePath);
00129 void showTable();
00130 void launchSLRWizard();
00131 void updateProgressPanel();
00132 void addUserState(unsigned id);
00133 void addUserTransition(unsigned fromId, const std::string& symbol,
00134 unsigned toId); // Register a user-created transition
00135
00136 // ===== Visual Feedback & Animations =====
00137 void animateLabelPop(QLabel* label);
00138 void animateLabelColor(QLabel* label, const QColor& flashColor);
00139 void wrongAnimation(); // Label animation for incorrect answer
00140 void wrongUserResponseAnimation(); // Message widget animation for incorrect
00141 // answer
00142 void markLastUserIncorrect();
00143
00144 // ===== Response Verification =====
00145 bool verifyResponse(const QString& userResponse);
00146 bool verifyResponseForA(const QString& userResponse);
00147 bool verifyResponseForA1(const QString& userResponse);
00148 bool verifyResponseForA2(const QString& userResponse);
00149 bool verifyResponseForA3(const QString& userResponse);
00150 bool verifyResponseForA4(const QString& userResponse);
00151 bool verifyResponseForB(const QString& userResponse);

```

```

00152 bool verifyResponseForC(const QString& userResponse);
00153 bool verifyResponseForCA(const QString& userResponse);
00154 bool verifyResponseForCB(const QString& userResponse);
00155 bool verifyResponseForD(const QString& userResponse);
00156 bool verifyResponseForD1(const QString& userResponse);
00157 bool verifyResponseForD2(const QString& userResponse);
00158 bool verifyResponseForE(const QString& userResponse);
00159 bool verifyResponseForE1(const QString& userResponse);
00160 bool verifyResponseForE2(const QString& userResponse);
00161 bool verifyResponseForF(const QString& userResponse);
00162 bool verifyResponseForFA(const QString& userResponse);
00163 bool verifyResponseForG(const QString& userResponse);
00164 bool verifyResponseForH();
00165
00166 // ===== Correct Solutions (Auto-generated) =====
00167 QString solution(const std::string& state);
00168 std::unordered_set<Lr0Item> solutionForA();
00169 QString solutionForA1();
00170 QString solutionForA2();
00171 std::vector<std::pair<std::string, std::vector<std::string>>>
00172 solutionForA3();
00173 std::unordered_set<Lr0Item> solutionForA4();
00174 unsigned solutionForB();
00175 unsigned solutionForC();
00176 QStringList solutionForCA();
00177 std::unordered_set<Lr0Item> solutionForCB();
00178 QStringList solutionForD();
00179 QString solutionForD1();
00180 QString solutionForD2();
00181 std::ptrdiff_t solutionForE();
00182 QSet<unsigned> solutionForE1();
00183 QMap<unsigned, unsigned> solutionForE2();
00184 QSet<unsigned> solutionForF();
00185 QSet<QString> solutionForFA();
00186 QSet<QString> solutionForG();
00187
00188 // ===== Pedagogical Feedback =====
00189 QString feedback(); // Delegates to appropriate feedback based on state
00190 QString feedbackForA();
00191 QString feedbackForA1();
00192 QString feedbackForA2();
00193 QString feedbackForA3();
00194 QString feedbackForA4();
00195 QString feedbackForAPrime();
00196 QString feedbackForB();
00197 QString feedbackForB1();
00198 QString feedbackForB2();
00199 QString feedbackForBPrime();
00200 QString feedbackForC();
00201 QString feedbackForCA();
00202 QString feedbackForCB();
00203 QString feedbackForD();
00204 QString feedbackForD1();
00205 QString feedbackForD2();
00206 QString feedbackForDPrime();
00207 QString feedbackForE();
00208 QString feedbackForE1();
00209 QString feedbackForE2();
00210 QString feedbackForF();
00211 QString feedbackForFA();
00212 QString feedbackForG();
00213 QString TeachDeltaFunction(const std::unordered_set<Lr0Item>& items,
00214 const QString& symbol);
00215 void TeachClosureStep(std::unordered_set<Lr0Item>& items, unsigned int size,
00216 std::unordered_set<std::string>& visited, int depth,
00217 QString& output);
00218 QString TeachClosure(const std::unordered_set<Lr0Item>& initialItems);
00219 private slots:
00220 void on_confirmButton_clicked();
00221 void on_userResponseTextChanged();
00222 signals:
00223 void sessionFinished(int cntRight, int cntWrong);
00224
00225 protected:
00226 void closeEvent(QCloseEvent* event) override {
00227 emit sessionFinished(cntRightAnswers, cntWrongAnswers);
00228 QWidget::closeEvent(event);
00229 }
00230
00231
00232 private:
00233 // ===== Helper Functions =====
00234 std::vector<std::string> qvectorToStdVector(const QVector<QString>& qvec);
00235 QVector<QString> stdVectorToQVector(const std::vector<std::string>& vec);
00236 QSet<QString>
00237 stdUnorderedSetToQSet(const std::unordered_set<std::string>& uset);
00238 std::unordered_set<std::string>

```

```

00239 qsetToStdUnorderedSet(const QSet<QString>& qset);
00240 std::unordered_set<Lr0Item> ingestUserItems(const QString& userResponse);
00241 std::vector<std::pair<std::string, std::vector<std::string>>
00242 ingestUserRules(const QString& userResponse);
00243 void setupTutorial();
00244 // ===== Core Components =====
00245 Ui::SLRTutorWindow* ui;
00246 Grammar grammar;
00247 SLR1Parser slr1;
00248
00249 // ===== State and Grammar Tracking =====
00250 StateSlr currentState;
00251 QVector<QString> sortedNonTerminals;
00252 QVector<QPair<QString, QVector<QString>> sortedGrammar;
00253 QString formattedGrammar;
00254
00255 unsigned cntRightAnswers = 0;
00256 unsigned cntWrongAnswers = 0;
00257
00258 // ===== State Machine Runtime Variables =====
00259 std::unordered_set<state> userMadeStates; // All states the user has created
00260 std::unordered_map<unsigned, std::unordered_map<std::string, unsigned>>
00261 userMadeTransitions; // Transitions made by the user
00262 UniqueQueue<unsigned>
00263 statesIdQueue; // States to be processed in B-C-CA-CB loop
00264 unsigned currentStateId = 0;
00265 state currentSlrState;
00266
00267 QStringList followSymbols; // Used in CA-CB loop
00268 qsizetype currentFollowSymbolsIdx = 0;
00269 unsigned int nextStateId = 0;
00270
00271 QVector<const state*> statesWithLr0Conflict; // Populated in F
00272 std::queue<unsigned> conflictStatesIdQueue;
00273 unsigned currentConflictStateId = 0;
00274 state currentConflictState;
00275
00276 std::queue<unsigned>
00277 reduceStatesIdQueue; // States without conflicts but with reduce
00278 unsigned currentReduceStateId = 0;
00279 state currentReduceState;
00280
00281 struct ActionEntry {
00282 enum Type { Shift, Reduce, Accept, Goto } type;
00283 int target;
00284 static ActionEntry makeShift(int s) { return {Shift, s}; }
00285 static ActionEntry makeReduce(int r) { return {Reduce, r}; }
00286 static ActionEntry makeAccept() { return {Accept, 0}; }
00287 static ActionEntry makeGoto(int g) { return {Goto, g}; }
00288 };
00289
00290 QMap<int, QMap<QString, ActionEntry>> slrtable;
00291 QVector<QVector<QString>> rawTable;
00292
00293 // ===== Conversation Log =====
00294 struct MessageLog {
00295 QString message;
00296 bool isUser;
00297 bool isCorrect = true;
00298
00299 MessageLog(const QString& message, bool isUser)
00300 : message(message), isUser(isUser) {}
00301
00302 void toggleIsCorrect() { isCorrect = false; }
00303 };
00304
00305 QVector<MessageLog> conversationLog;
00306 QWidget* lastUserMessage = nullptr;
00307 qsizetype lastUserMessageLogIdx = -1;
00308
00309 QPropertyAnimation* m_shakeAnimation =
00310 nullptr; // For interrupting userResponse animation if they spam enter
00311 // key
00312
00313 TutorialManager* tm;
00314
00315 QRegularExpression re{ "^\s+|\s+\$"};
00316 };
00317
00318 #endif // SLRTUTORWINDOW_H

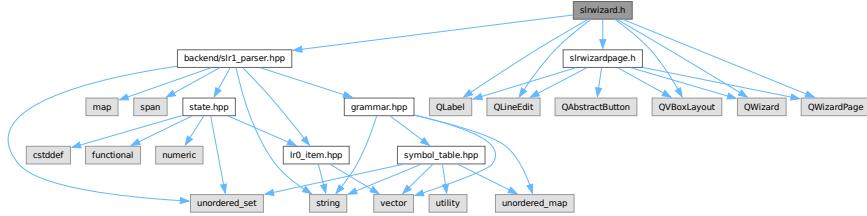
```

## 7.42 slrwizard.h File Reference

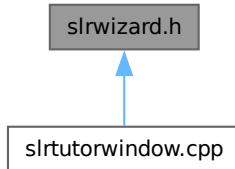
```
#include "backend/slrl_parser.hpp"
#include "slrwizardpage.h"
```

```
#include <QLabel>
#include <QLineEdit>
#include <QVBoxLayout>
#include <QWizard>
#include <QWizardPage>
```

Include dependency graph for slrwizard.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [SLRWizard](#)

Interactive assistant that guides the student step-by-step through the SLR(1) parsing table.

## 7.43 slrwizard.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #ifndef SLRWIZARD_H
00020 #define SLRWIZARD_H

```

```

00021
00022 #include "backend/slr1_parser.hpp"
00023 #include "slrwizardpage.h"
00024 #include <QLabel>
00025 #include <QLineEdit>
00026 #include <QVBoxLayout>
00027 #include <QWizard>
00028 #include <QWizardPage>
00029
00030 class SLRWizard : public QWizard {
00031 Q_OBJECT
00032 public:
00033 SLRWizard(SLR1Parser& parser, const QVector<QString>& rawTable,
00034 const QStringList& colHeaders,
00035 const QVector<QPair<QString, QVector<QString>>& sortedGrammar,
00036 QWidget* parent = nullptr)
00037 : QWizard(parent) {
00038 setWindowTitle(tr("Ayuda interactiva: Tabla SLR(1")));
00039
00040 const int nTerm =
00041 parser.gr_st_.terminals_.contains(parser.gr_st_.EPSILON_) ?
00042 parser.gr_st_.terminals_.size() - 1
00043 : parser.gr_st_.terminals_.size();
00044 SLRWizardPage* last = nullptr;
00045 // Generar explicación y páginas
00046 int rows = rawTable.size();
00047 int cols = colHeaders.size();
00048 for (int i = 0; i < rows; ++i) {
00049 for (int j = 0; j < cols; ++j) {
00050 QString sym = colHeaders[j];
00051 QString expected;
00052 QString explanation;
00053 if (j < nTerm) {
00054 auto itAct = parser.actions_.at(i).find(sym.toStdString());
00055 SLR1Parser::s_action act =
00056 (itAct != parser.actions_.at(i).end() ?
00057 itAct->second
00058 : SLR1Parser::s_action{nullptr,
00059 SLR1Parser::Action::Empty});
00060 switch (act.action) {
00061 case SLR1Parser::Action::Shift: {
00062 unsigned to =
00063 parser.transitions_.at(i).at(sym.toStdString());
00064 expected = QString("s%1").arg(to);
00065 explanation = tr("Estado %1: existe transición (%1, "
00066 "'%2'). ¿A qué "
00067 "estado harías shift?")
00068 .arg(i)
00069 .arg(sym);
00070 break;
00071 }
00072 case SLR1Parser::Action::Reduce: {
00073 int idx = -1;
00074 for (int k = 0; k < sortedGrammar.size(); ++k) {
00075 auto& rule = sortedGrammar[k];
00076 if (rule.first.toStdString() ==
00077 act.item->antecedent_ &&
00078 stdVectorToQVector(act.item->consequent_) ==
00079 rule.second) {
00080 idx = k;
00081 break;
00082 }
00083 }
00084 expected = QString("r%1").arg(idx);
00085 // explicación con FOLLOW
00086 std::unordered_set<std::string> F;
00087 F = parser.Follow(act.item->antecedent_);
00088 QStringList followList;
00089 for (auto& t : F)
00090 followList « QString::fromStdString(t);
00091 explanation = tr("Estado %1: contiene el ítem [%2 → "
00092 "...] y '%3' "
00093 "SIG(%2). ¿Qué regla usas para "
00094 "reducir (0, 1, ...)?")
00095 .arg(i)
00096 .arg(QString::fromStdString(
00097 act.item->antecedent_))
00098 .arg(colHeaders[j]);
00099 break;
00100 }
00101 case SLR1Parser::Action::Accept:
00102 expected = "acc";
00103 explanation = tr("Estado %1: contiene [S → A · $.] "
00104 ";Qué palabra clave "
00105 "usas para aceptar?")
00106 .arg(i);
00107 break;
00108 }
00109 }
00110 }
00111 // explicación con FOLLOW
00112 std::unordered_set<std::string> F;
00113 F = parser.Follow(act.item->antecedent_);
00114 QStringList followList;
00115 for (auto& t : F)
00116 followList « QString::fromStdString(t);
00117 explanation = tr("Estado %1: contiene el ítem [%2 → "
00118 "...] y '%3' "
00119 "SIG(%2). ¿Qué regla usas para "
00120 "reducir (0, 1, ...)?")
00121 .arg(i)
00122 .arg(QString::fromStdString(
00123 act.item->antecedent_))
00124 .arg(colHeaders[j]);
00125 break;
00126 }
00127 case SLR1Parser::Action::Accept:
00128 expected = "acc";
00129 explanation = tr("Estado %1: contiene [S → A · $.] "
00130 ";Qué palabra clave "
00131 "usas para aceptar?")
00132 .arg(i);
00133 break;
00134 }
00135 }
```

```

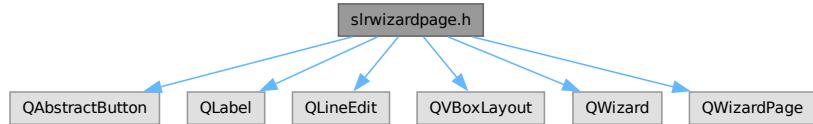
00134 case SLR1Parser::Action::Empty:
00135 default:
00136 continue;
00137 }
00138 } else {
00139 // GOTO sobre no terminal
00140 auto nonT = sym.toStdString();
00141 if (!parser.transitions_.contains(i)) {
00142 continue;
00143 }
00144 auto itGo = parser.transitions_.at(i).find(nonT);
00145 if (itGo != parser.transitions_.at(i).end()) {
00146 expected_ = QString::number(itGo->second);
00147 explanation = tr("Estado %1: (%1, '%2') existe. ¿A "
00148 "qué estado va "
00149 "la transición? (pon solo el número)")
00150 .arg(i)
00151 .arg(sym);
00152 } else {
00153 continue;
00154 }
00155 }
00156
00157 SLRWizardPage* page =
00158 new SLRWizardPage(i, sym, explanation, expected, this);
00159 last = page;
00160 addPage(page);
00161 }
00162 }
00163 if (last) {
00164 last->setFinalPage(true);
00165 }
00166 }
00167
00174 QVector<QString> stdVectorToQVector(const std::vector<std::string>& vec) {
00175 QVector<QString> result;
00176 result.reserve(vec.size());
00177 for (const auto& str : vec) {
00178 result.push_back(QString::fromStdString(str));
00179 }
00180 return result;
00181 }
00182 };
00183
00184 #endif // SLRWIZARD_H

```

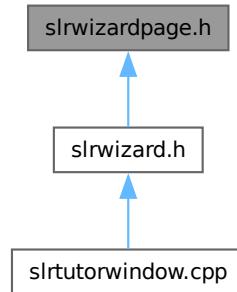
## 7.44 slrwizardpage.h File Reference

```
#include <QAbstractButton>
#include <QLabel>
#include <QLineEdit>
#include <QVBoxLayout>
#include <QWizard>
#include <QWizardPage>
```

Include dependency graph for slrwizardpage.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [SLRWizardPage](#)

A single step in the SLR(1) guided assistant for table construction.

## 7.45 slrwizardpage.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018 #ifndef SLRWIZARDPAGE_H
00019 #define SLRWIZARDPAGE_H
00021
00022 #include <QAbstractButton>
00023 #include <QLabel>
00024 #include <QLineEdit>
00025 #include <QVBoxLayout>
00026 #include <QWizard>
00027 #include <QWizardPage>
00028
00041 class SLRWizardPage : public QWizardPage {
00042 Q_OBJECT
00043 public:
00054 SLRWizardPage(int state, const QString& symbol, const QString& explanation,
00055 const QString& expected, QWidget* parent = nullptr)
00056 : QWizardPage(parent), m_state(state), m_symbol(symbol),
00057 m_expected(expected) {
00058 setTitle(tr("Estado %1, simbolo '%2'").arg(state).arg(symbol));
00059
00060 QLabel* lbl = new QLabel(explanation, this);
00061 lbl->setWordWrap(true);
00062
00063 m_edit = new QLineEdit(this);
00064 m_edit->setPlaceholderText(
00065 tr("Escribe tu respuesta (p.ej. s3, r2, acc, 5)"));
00066
00067 QVBoxLayout* layout = new QVBoxLayout(this);
00068 layout->addWidget(lbl);
00069 layout->addWidget(m_edit);

```

```

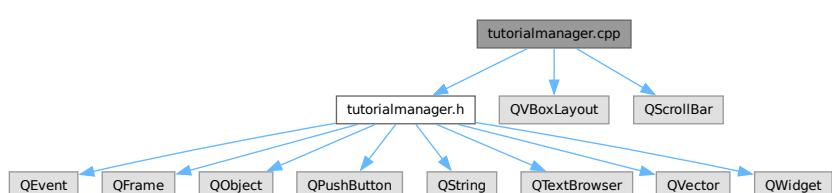
00070 setLayout(layout);
00071
00072 connect(m_edit, &QLineEdit::textChanged, this,
00073 &SLRWizardPage::onTextChanged);
00074 }
00075 private slots:
00076 void onTextChanged(const QString& text) {
00077 bool correct = (text.trimmed() == m_expected);
00078 setComplete(correct);
00079 if (correct) {
00080 setSubTitle(
00081 tr(" Respuesta correcta, pasa a la siguiente pregunta"));
00082 } else {
00083 setSubTitle(tr(" Incorrecto, revisa el enunciado. Consulta los "
00084 "estados que has construido."));
00085 }
00086 wizard()->button(QWizard::NextButton)->setEnabled(correct);
00087 }
00088
00089
00090
00091
00092
00093
00094 private:
00095 void setComplete(bool complete) {
00096 m_isComplete = complete;
00097 emit completeChanged();
00098 }
00099 bool isComplete() const override { return m_isComplete; }
00100
00101 int m_state;
00102 QString m_symbol;
00103 QString m_expected;
00104 QLineEdit* m_edit;
00105 bool m_isComplete =
00106 false;
00107 }
00108
00109 #endif // SLRWIZARDPAGE_H

```

## 7.46 tutorialmanager.cpp File Reference

```
#include "tutorialmanager.h"
#include <QVBoxLayout>
#include <QScrollBar>
```

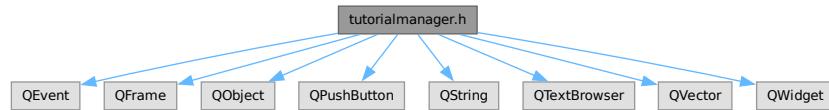
Include dependency graph for tutorialmanager.cpp:



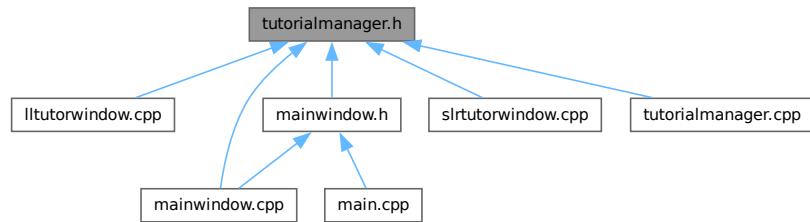
## 7.47 tutorialmanager.h File Reference

```
#include <QEvent>
#include <QFrame>
#include <QObject>
#include <QPushButton>
#include <QString>
#include <QTextBrowser>
#include <QVector>
#include <QWidget>
```

Include dependency graph for tutorialmanager.h:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [TutorialStep](#)  
Represents a single step in the tutorial sequence.
- class [TutorialManager](#)  
Manages interactive tutorials by highlighting UI elements and guiding the user.

## 7.48 tutorialmanager.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software; you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #ifndef TUTORIALMANAGER_H
00020 #define TUTORIALMANAGER_H
00021
00022 #include <QEvent>
00023 #include <QFrame>
00024 #include <QObject>
00025 #include <QPushButton>
00026 #include <QString>
00027 #include <QTextBrowser>
00028 #include <QVector>
00029 #include <QWidget>
00030
00031 struct TutorialStep {
00032 QWidget* target;
00033 QString htmlText;

```

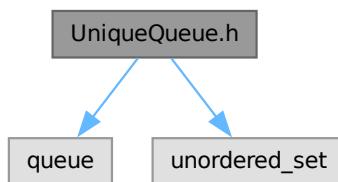
```

00041 };
00042
00053 class TutorialManager : public QObject {
00054 Q_OBJECT
00055 public:
00061 TutorialManager(QWidget* rootWindow);
00062
00068 void addStep(QWidget* target, const QString& htmlText);
00069
00073 void start();
00074
00079 void setRootWindow(QWidget* newRoot);
00080
00084 void clearSteps();
00085
00089 void hideOverlay();
00090
00095 void finishLL1();
00096
00101 void finishSLR1();
00102
00103 protected:
00107 bool eventFilter(QObject* obj, QEvent* ev) override;
00108
00109 signals:
00114 void stepStarted(int index);
00115
00119 void tutorialFinished();
00120
00124 void ll1Finished();
00125
00129 void slr1Finished();
00130
00131 public slots:
00135 void nextStep();
00136
00137 private:
00141 void showOverlay();
00142
00146 void repositionOverlay();
00147
00148 QWidget* m_root;
00149 QVector<TutorialStep> m_steps;
00150 int m_index = -1;
00151
00152 QWidget* m_overlay = nullptr;
00153 QFrame* m_highlight =
00154 nullptr;
00155 QTextBrowser* m_textBox = nullptr;
00156 QPushButton* m_nextBtn = nullptr;
00157 };
00158
00159 #endif // TUTORIALMANAGER_H

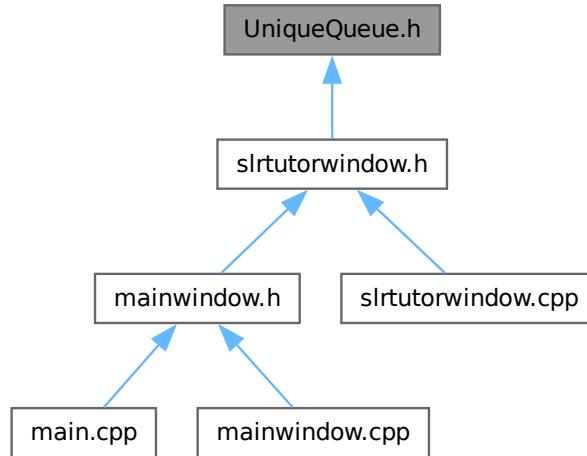
```

## 7.49 UniqueQueue.h File Reference

```
#include <queue>
#include <unordered_set>
Include dependency graph for UniqueQueue.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class `UniqueQueue< T >`  
A queue that ensures each element is inserted only once.

## 7.50 UniqueQueue.h

[Go to the documentation of this file.](#)

```

00001 /*
00002 * SyntaxTutor - Interactive Tutorial About Syntax Analyzers
00003 * Copyright (C) 2025 Jose R. (jose-rzm)
00004 *
00005 * This program is free software: you can redistribute it and/or modify it
00006 * under the terms of the GNU General Public License as published by
00007 * the Free Software Foundation, either version 3 of the License, or
00008 * (at your option) any later version.
00009 *
00010 * This program is distributed in the hope that it will be useful,
00011 * but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 * GNU General Public License for more details.
00014 *
00015 * You should have received a copy of the GNU General Public License
00016 * along with this program. If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 #ifndef UNIQUEQUEUE_H
00020 #define UNIQUEQUEUE_H
00021 #include <queue>
00022 #include <unordered_set>
00023
00024 template <typename T> class UniqueQueue {
00025 public:
00026 void push(const T& value) {
00027 if (seen_.insert(value).second) {
00028 queue_.push(value);
00029 }
00030 }
00031
00032 void pop() {
00033 if (!queue_.empty()) {
00034 queue_.pop();
00035 }
00036 }
00037 const T& front() const { return queue_.front(); }

```

```
00063
00068 bool empty() const { return queue_.empty(); }
00069
00073 void clear() {
00074 while (!queue_.empty())
00075 queue_.pop();
00076 seen_.clear();
00077 }
00078
00079 private:
00080 std::queue<T> queue_;
00081 std::unordered_set<T> seen_;
00082 }
00083 #endif // UNIQUEQUEUE_H
```