Integrated Development Environment for LOOP, WHILE and GOTO

User Manual

The Theo-IDE Developers

Contents

Section 1

Introduction

This document serves as an introduction to the use of Theo-IDE, a program developed for a project named *Integrated Development Enviroment for LOOP, WHILE and GOTO* at the University of Applied Sciences Würzburg-Schweinfurt. LOOP, WHILE and GOTO [1] (hereafter referred to as *the project languages*) are the names of a range of simple concept programming languages used within lectures such as *Foundations of Theoretical Computer Science* and are used for demonstrating a range of concepts and isomorphisms in computability theory. Theo-IDE provides a simple explorative enviroment for developing, running and debugging programs written in these languages.

1.1 Requirements

During the project, great emphasis was placed on accessability and portability. As a result, Theo-IDE is available on a range of different desktop and mobile platforms. We officially support Linux (X11, Wayland), MacOS and Windows desktops and provide an application package for Android (ARMv7, ARMv8). If you're willing to get your hands dirty yourself, the project has also been verified to build for iOS.

1.2 Setup

Binary distributions for most major platforms are available in the project repository at **https://github.com/Theo-IDE/Theo-IDE/**. If there happens to be no binary artifact for your platform, the repository also includes a list of dependencies and build instructions.

Section 2

Basic Functionality

Once the Theo-IDE application is opened, you will be greeted with the user interface as pictured in **Figure 2.1**. Depending on the window width of your platform, the sidebar may be located at the bottom, and some of the buttons in the primary action bar at the top may be hidden in drop-down menus. We will now proceed with an explanation of the functionality of the Theo-IDE.

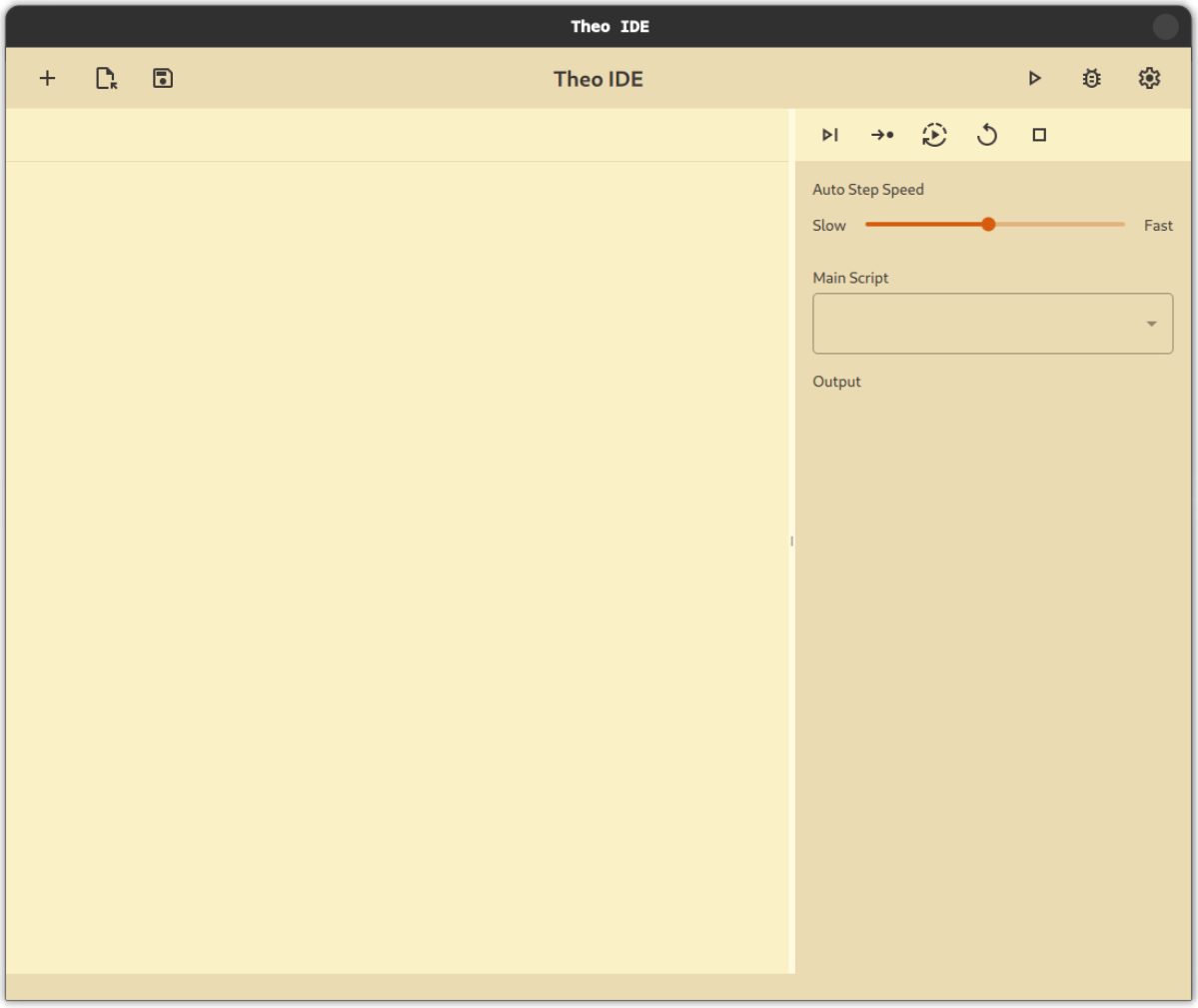

Figure 2.1 User Interface of Theo-IDE

The action bar at the top (which also contains the "Theo-IDE" label) is refereed to as the *primary action bar*. Below the primary action bar the view is split into the *editor* (on the left side in **Figure 2.1**) and the *sidebar*. The sidebar further is segmented into the *secondary action bar* at its top and the *control area* below it.

2.1 Creating, Opening and Saving Files

You may create new program files by clicking on the button with the plus icon in the primary action bar (the leftmost button in **Figure 2.1**). In situations where the window is narrow, this button may be hidden in a drop-down menu, which is accessible from the primary action bar.

Upon clicking this button, a new editor tab named *Temporary-** will open, and you'll be able to enter your source code.

You may load source code from files contained in your file system by clicking on the button with the file icon in the primary action bar (next to the creating button in **Figure 2.1**). In narrow windows, this button is hidden in the same drop-down menu as the opening button.

Upon clicking this button, a file dialog will open, prompting you for the files you wish to load. After confirming your selection of files, one editor tab will open for each of the files you opened, where the name of the tab is the file name in your file system.

Analogously to opening files, you may save modified or created files by clicking on the button with the save icon (to the right of the opening button in **Figure 2.1**).

This action will save whatever file is currently selected in the editor.

2.2 Running Programs

Running programs is a two-step process: First you select the main file of your program in the control area by selecting the name of the editor tab that contains it from the drop-down menu labeled *Main Script*. Secondly, in the primary action bar, you click on the button with the triangular icon.

This action will first compile your program. If any errors occur during compilation you will be shown a message dialog notifying you of the incident. If no errors occur, your program will be executed. When execution finishes successfully, a table showing the final variable contents will appear in the control area under the label *Output*. Should your program not halt for any reason, you may use the stop button in the secondary action bar (the one with the square iconography), to forcefully halt your program.

2.3 Debugging Programs

If you want to step through your program line-by-line or have the program halt on predefined breakpoints, you will want to make use of Theo-IDEs debugging capabilities.

At first, you will follow exactly the same process as when running a program, except that after selecting your main file, you will click on the debugging button (next to the run button in **Figure 2.1**, with the insect icon).

At this point, your program will once again be compiled, and any errors will be reported with a message dialog. But the program *won't* be executed straight away, and you will be able to make use of the buttons in the secondary action bar to influence execution:

- the leftmost button in **Figure 2.1** will execute the program until a breakpoint is encountered or the program ends (whichever happens first)
- the second button from the left in **Figure 2.1** will execute one line of code
- the third button from the left in **Figure 2.1** toggles auto-stepping mode: When enabled, Theo-IDE will automatically execute lines of code with a delay inbetween,

the length of which is altered with the slider titled *Auto Step Speed* in the control area

- the fourth button from the left in **Figure 2.1** resets execution to the beginning of the program
- the fifth button from the left in **Figure 2.1** stops execution

You may activate or deactivate breakpoints by clicking on the line number in the editor at the location where you want the program to pause. During execution, the output table in the control area will always show the current variable states.

References

[1] U. Schöning, *Theoretische Informatik - kurz gefasst* (Spektrum Akademischer Verlag, 2008).