A Rodin Plugin for the Tinker Tool

By Yibo Liang

Ver. 01

Table of Content

Introduction

Tinker is a tool that implements the proof-strategy graph (PSGraph) formalism. Here, tactics are represented as nodes and are linked with each other with edges. Each tactic will consume goals on input edges and generate new sub-goals which are sent to the output edges. The Tinker tool is generic w.r.t. theorem provers, and supports Isabelle and ProofPower.

This plugin aim to provide Rodin Prover users with the functionality of Tinker. It integrates TinkerGUI, Tinker core and Rodin platform to provide PSGraph feature for Rodin users. This plugin is a software system including mainly 2 parts, a Rodin Plugin (Eclipse Plugin) programmed in Java and an extension to Tinker in PolyML.

Definition

Atomic tactic: a tactic that only represent an atomic evaluation of a goal. Note that the model of atomic tactics contains a "tactic" field which does not correspond to their name, but to a complete description of their purpose.

On Hyp Tactic: Tactics that only applies on hypothesis in Rodin platform, such as deducing a conjunction in a predicate of hypothesis.

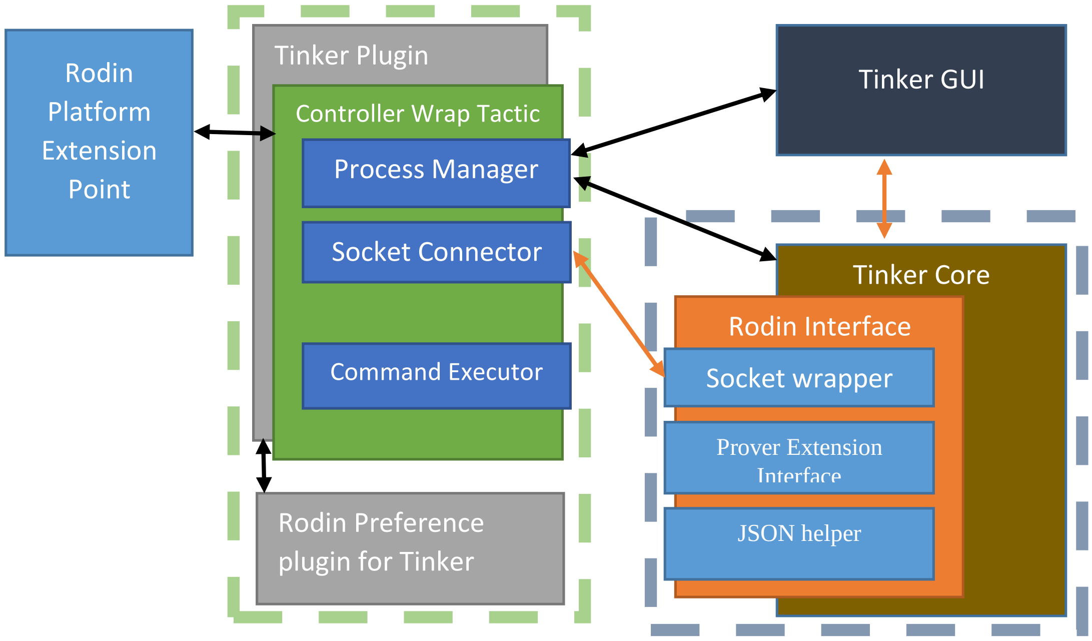
On Goal Tactic: Tactics that only applies on goal in Rodin platform, such as case splitting a disjunction in a predicate of goals.

Auto tactics: Automatic Tactics that are originally Rodin built in, such as newPP and Lasoo. Mostly are a combination of different atomic tactics or other auto tactics with predefined applying rules and order. The auto tactics include combinations of atomic tactics of which type of outcomes are non-deterministic, since PSGraph is a typed proof model, user must be very certain of their work before they use this type of tactics.

Proof Node: The node displayed in Proof Tree View in Rodin Platform. Each node represent a goal that may need to be proved.

High Level Overview

The Tinker plugin is made of two main groups of components, a Rodin side Eclipse based Plugins written in Java and a Tinker side interface written in PolyML.



As you can see from the figure, there are two plugins implemented on Rodin side. On Rodin Platform, Tinker Plugin implements the proof tactic extension point that wraps the controller for the plugin.

On the left side, the controller as an implementation of a tactic, warps all the program flow in its inherited “apply” method. It has a process manager that launches, monitors and terminates the Tinker Core and GUI process. A socket connector class is used to manage the connection and communication between the Plugin and Tinker Core. Command Executor class reads the JSON command from Tinker Core and makes corresponding changes to the proof node.

Components

Rodin Side

The Tinker plugin

The Tinker Plugin for Rodin Platform is technically an Eclipse plugin, because Rodin Platform is built on Eclipse platform. A plugin activator is needed for the platform to control the life cycle of the plugin. Then there is a tactic provider class that extend a Rodin class named DefaultTacticProvider which is used by Rodin tactic manager. Each tactic provider will add a tactic button on the Proof Control View tool bar on Rodin Platform. The interfaces are implemented in the following files in package tinker.core.plugin:

- *PluginActivator.java*: the activator class used by Eclipse platform to control the plugin’s life cycle.
- *TinkerDebugMode.java*: the class that offers all current Tinker functionality by using *TinkerTactic.java*.

- *TinkerAutoProveMode.java*: the class designed to use Tinker as an auto tactic like other auto tactics in Rodin. This is disabled in this version because the functionality is not integrated with TinkerGUI.

The Preference Plugin

This is a separate plugin that add a preference page to Rodin Platform for Tinker plugin. It includes 3 files in package tinker.core.preference:

- *PreferenceConstants.java*: a class that holds all string names of the preference options.
- *PreferenceInitializer.java*: a class used by Eclipse Platform to initialise the preference page.
- *TinkerPreferencePage.java*: the class that implements the view of the preference page, such as adding textbox and buttons.

The controller

The controller includes 3 parts, they are:

Controller itself

with files in package *tinker.core.protocol*:

- *TinkerTactic.Java*: the class that implements **ITactic** class of Rodin platform. Unlike other class implements **ITactic** class that are originally implemented in Rodin Platform, Its "apply" method does not make any change to the proof node in Rodin directly, nevertheless it is actually the wrapper of the controller of the plugin. The modification to proof node tree is done by Command Executor.
- *TinkerSession.java*: the model class that holds protocol states of current session plus the information needed for the connection to Tinker Core such as the PSGraph file path.
- *PluginStates*.java: the static class that holds the constant representing plugin states.
- *SocketStates*.java: the static class that holds the constant representing Socket states.
- *TacticStates*.java: the static class that holds the constant representing Tinker Tactic states, which are basically the controller states.

The states are used to by the controller to control the program flow and keep the consistence of the communication between processes.

Process Manager with files:

- *TinkerGUIProcess.java*: the singleton class that handles the Tinker GUI process.
- *TinkerProcess.java*: the singleton class that handles the Tinker Core process.

Socket Connector with file:

- *TinkerConnector.java*: the socket wrapper class, which also reports current socket state to the controller.

Tinker Side

On Tinker side of the plugin, the added files all together serves as the interface that handles the communication between Tinker Core and Rodin. All files were developed in Isabelle 2014 platform and later were compiled independent of Isabelle.

Socket Wrapper

- *interface/wsock.sig.ML*
- *build/rodin_socket_struct.ML*
- *build/rodin_text_socket.ML*

The socket connection is built on top of the textsocket.ML file from Quantomatics. Above 3 files together wraps the textsocket.ML and offers convenience for further usage.

JSON Wrapper

- *build/rodin_json_protocol.struct.ML*
- *interface/json_socket.sig.ML*

The above 2 files helps to create and parse JSON string for socket communication.

Protocol Wrapper

- *interface/tpp_protocol.sig.ML*
- *build/rodin_protocol.ML*

These 2 files implements the tinker side protocol of communication.

Compiling Tinker

The Tinker Core for Rodin is compiled as an independent executable. The process of compiling is complicated and will not be explained in detail in this manual. To develop an automatic compiler for Tinker core with a user manual will be a part of future work.

Briefly speaking, because the Tinker Core was built in Isabelle, where many .ML files are connected and referenced by Isabelle theory file .thy, these .thy files are all recursively parsed and turned into .ML file in a bottom-up style using a small Java Compiler implemented myself. Then the dependencies are manually solved by only keeping the necessary file reference, because Tinker Core fore Rodin does not need the functionality used by ProofPower or Isabelle, whereas Tinker Core depends on Quantomatics of which only part functionality is used. Then the .ML files are optimised by deleting repetitive reference. Finally the files are chained together and loaded into PolyML, and by using the compiling functionality built-in PolyML the Core is compiled into an .o file. Then GCC is used to turn the .o file into an executable.

Protocol

Overview

The following table shows the overview communication flow between Tinker Core and Rodin Plugin from top to bottom:

Step	Tinker GUI	Tinker Core	Rodin Plugin
1			User clicks “Tinker Tactic” button, select PSGraph file, lauches GUI and Tinker Core.

			The Plugin blocks on server socket waiting for connection
2	Launches	Launches, trying to connect to Rodin as a client.	
3			Connection made. Send PSGraph file path as string Blocks on waiting for reply
4		Receives file path, Blocks on trying connect to GUI as client	
5	User click “connect” on GUI, Start server socket		
6		Connection made with GUI Sends information with GUI Blocks on waiting for GUI	
7	GUI load PSGraph file Waiting for user actions		
8	User Action Sends corresponding information to Tinker Core		
9		Tinker Core send instructions to Plugin in JSON Blocks on waiting for Plugin	
10			Plugin receives instruction, performs some operation, Send back result, Blocks on waiting for reply
11	GUI exchange information with core. Perform visual change	According to result send by Plugin Exchanges information with GUI Send new instruction to Plugin	
	Step 10 and 11 are repeated until the user click stop on GUI.		
12		On receiving stop from GUI Send “STOP” instruction to Plugin	
13			Receive “Stop”

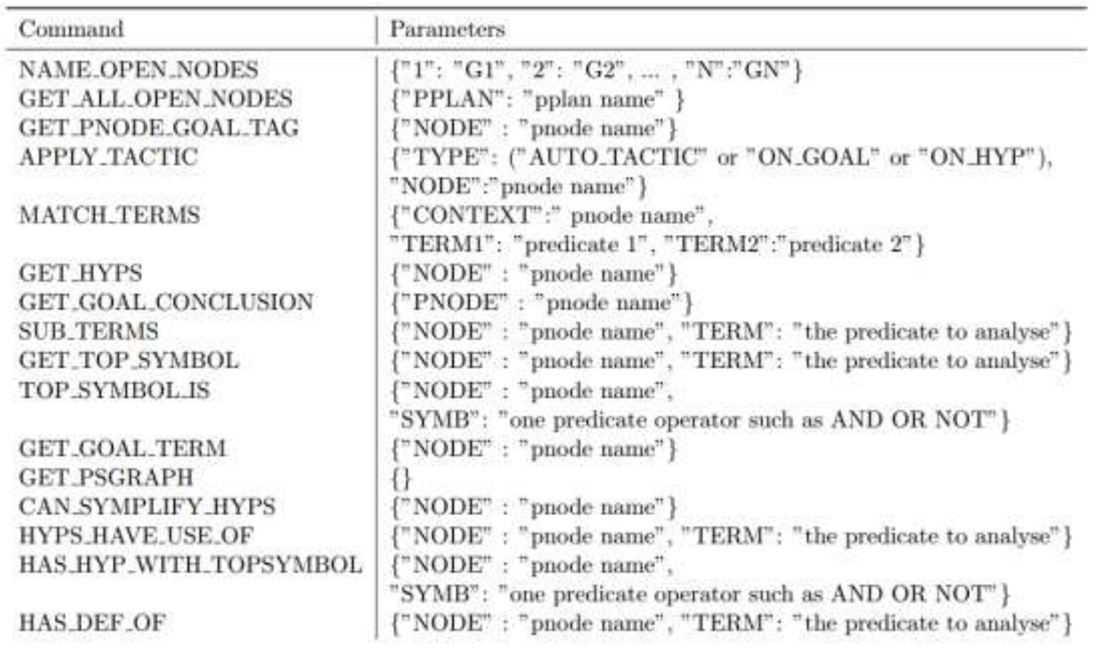
			Send “STOP_RECEIVED” Blocks on waiting for Disconnection
14		Receive “STOP_RECEIVED” Disconnect Socket with Plugin	
15			Disconnect Socket Tactic completes
16	Remains Open	Exit in 15 second if there is no further connection from Plugin	

Detail

The following are the collection of instructions and replies that Tinker and Rodin send to each other. The communication is simple:

ONE request from core is answered with ONE result, e.g. GET_HYPS <= GET_HYPS_RESULT

Table 1: Core to Plugin



Command	Parameters
NAME_OPEN_NODES	{"1": "G1", "2": "G2", ... , "N":"GN"}
GET_ALL_OPEN_NODES	{"PPLAN": "pplan name" }
GET_PNODE_GOAL_TAG	{"NODE" : "pnode name"}
APPLY_TACTIC	{"TYPE": ("AUTO_TACTIC" or "ON_GOAL" or "ON_HYP"), "NODE":"pnode name"}
MATCH_TERMS	{"CONTEXT":" pnode name", "TERM1": "predicate 1", "TERM2":"predicate 2"}
GET_HYPS	{"NODE" : "pnode name"}
GET_GOAL_CONCLUSION	{"PNODE" : "pnode name"}
SUB_TERMS	{"NODE" : "pnode name", "TERM": "the predicate to analyse"}
GET_TOP_SYMBOL	{"NODE" : "pnode name", "TERM": "the predicate to analyse"}
TOP_SYMBOL_IS	{"NODE" : "pnode name", "SYMB": "one predicate operator such as AND OR NOT"}
GET_GOAL_TERM	{"NODE" : "pnode name"}
GET_PSGRAPH	{}
CAN_SYMPLIFY_HYPS	{"NODE" : "pnode name"}
HYPS_HAVE_USE_OF	{"NODE" : "pnode name", "TERM": "the predicate to analyse"}
HAS_HYP_WITH_TOPSYMBOL	{"NODE" : "pnode name", "SYMB": "one predicate operator such as AND OR NOT"}
HAS_DEF_OF	{"NODE" : "pnode name", "TERM": "the predicate to analyse"}

Table 2: Plugin to Core

Command	Parameters
ERROR	{"INFO": "some error info"}
NAMING_COMPLETE	{ "0": "G1", "1": "G2", ... , "N":"GN"}
NEED_NAMING	{"NUM": "number of new nodes", "PARENT": "parent node name"}
NAMES	{ "0": "G1", "1": "G2", ... , "N":"GN"}
RETURN_TAG	{"TAG": "pnode goal tag"}
NODE_CLOSED	{}
MATCH_RESULT	{"RESULT" : "true or false"}
GET_HYPS_RESULT	{"1":"hypothesis 1", "2":"hypothesis 2", ... }
GET_GOAL_CONCLUSION_RESULT	{"GOAL":"goal term"}
SUB_TERMS	{"1":"subterm 1", "2":"subterm 2", ...}
GET_TOP_SYMBOL_RESULT	{"TAG": "some integer"}
TOP_SYMBOL_CHECK_RESULT	{"RESULT":"true or false"}
GET_GOAL_TERM_RESULT	{"TERM": "goal term"}
PS_GRAPH	{"PS":"psgraph file path"}
CAN_SYMPIFY_HYPS_RESULT	{"RESULT":"true or false"}
RESULT_HYPS_HAVE_USE_OF	{"RESULT":"true or false"}
HAS_HYP_WITH_TOPSYMBOL_RESULT	{"RESULT":"true or false"}
HAS_DEF_RESULT	{"RESULT":"true or false"}

Conclusion & Future work

Rodin can currently work with tinker with easy setup. With independently compiled Tinker version for Rodin, users only need one click to start using Tinker for Rodin proofs.

Future works

1. Need to solve exception handling problems for the connection between Tinker Core and Rodin. Currently when an exception is caught and if this exception is a socket connection exception, then the connection is assumed to be disconnected by the Plugin and Tactic will end to avoid dead lock and lose of work.
2. Need to integrate launch command with Tinker GUI, currently, user will need to click on Rodin to start GUI and click “connect” on GUI in order for the plugin to work. In future work, we aim to simplify this operation and user will only need to click one button to use everything.
3. Solve the exception handling problem for Tinker Core. Tinker core uses a socket implementation from Quantomatics. This implementation causes the thread to block permanently if there are 60 or more exception thrown while trying to connecting. This limits down the time we can retry to connect the Core to the plugin and it is the reason why the tinker core process has to be shut down after each session.
4. Solve the problem that users need to use tactic editor in GUI in order for Rodin to work with PSGraph.

Installation Guidelines

Download

Please download Compiled Tinker Core at link:

Linux

http://ggrov.github.io/tinker/abz2016/release/TinkerCore.tar.gz

Windows

http://ggrov.github.io/tinker/abz2016/release/TinkerCore.7z

Please download TinkerGUI at link:

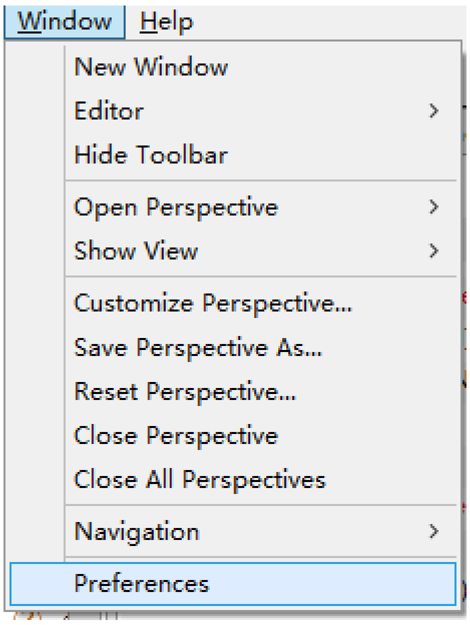
http://ggrov.github.io/tinker/abz2016/release/tinkerGUI.zip

Please download the plugin at link:

http://ggrov.github.io/tinker/abz2016/release/tinker.for.rodin.zip

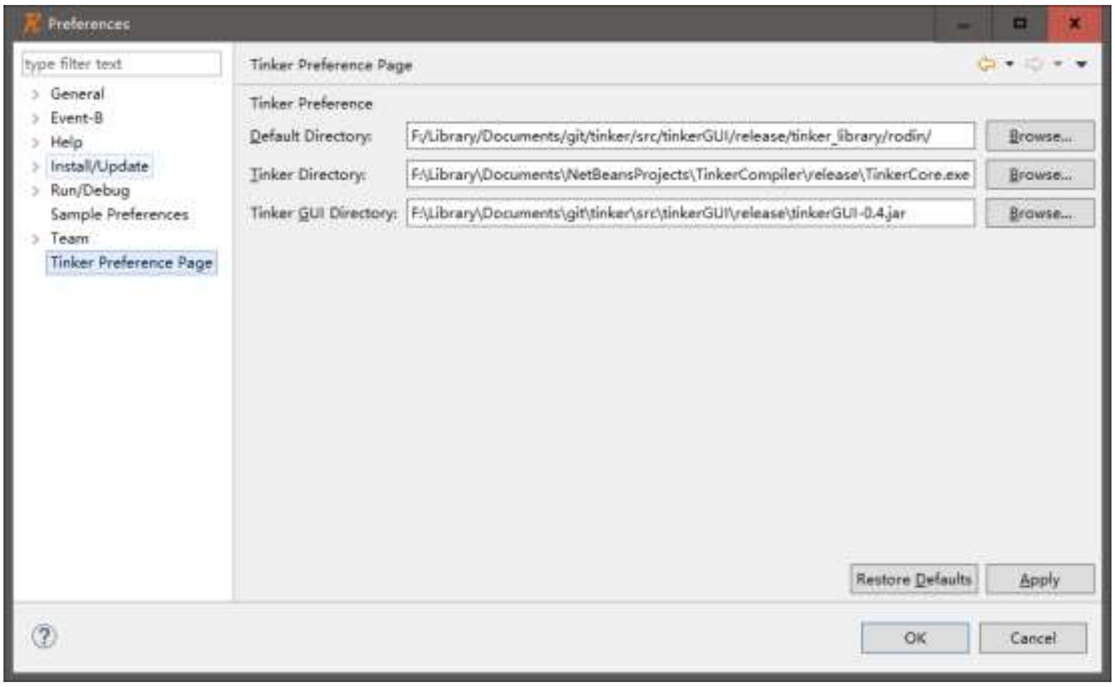
Installation

1. Unzip Tinker Core and Tinker GUI to a folder
2. Extract *tinker.for.rodin.zip* and copy *tinker.for.rodin.jar* to **plugins** folder under Rodin installation path.
3. Lauch Rodin, you should see a small T button in the toolbar in the Proof Control View
4. Click top menu Window -> Preference -> Tinker Preference Page



a. Set default path as any folder you would like to save your PSGraph folder. Do not worry about the slash direction, both directions will work regardless of the operating system.
b. Set tinker Directory as the executable file of Tinker Core. If you are using Windows System, the value should be like c:\Tinker\tinker.core.exe as shown in the image.
c. Set tinker GUI directory as the .jar file of the tinkerGUI executable.

Now you have everything set, and the plugin is ready for work.



Tinker Preference Page

Tinker Preference

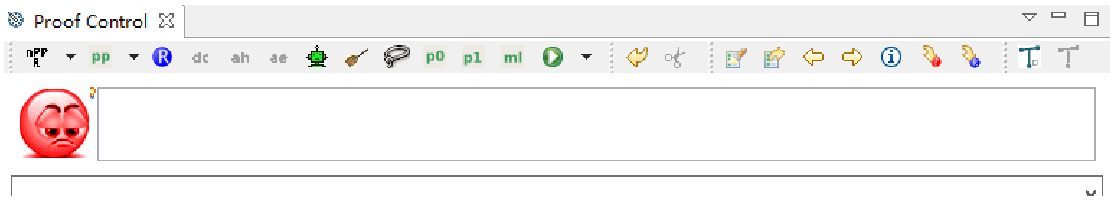
Default Directory: F:/Library/Documents/git/tinker/src/tinkerGUI/release/tinker_library/rodin/

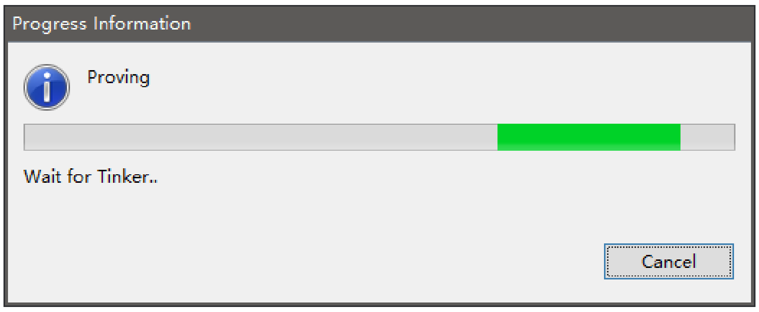
Tinker Directory: F:\Library\Documents\NetBeansProjects\TinkerCompiler\release\TinkerCore.exe

Tinker GUI Directory: F:\Library\Documents\git\tinker\src\tinkerGUI\release\tinkerGUI-0.4.jar

User Guide

1. Open any Rodin project, if there is an obligation that need to be proven, the T button should appear enabled. Click on it and a progress bar should show up like all other tactics.



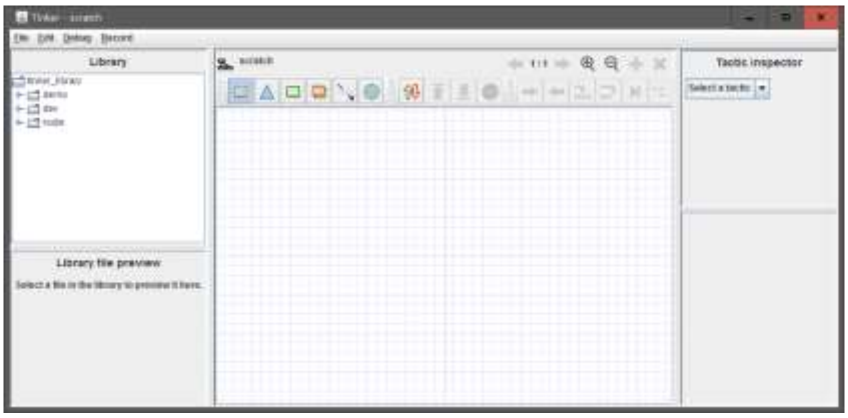


Progress Information

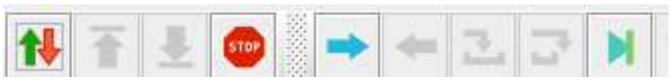
Proving

Wait for Tinker..

2. If this is the first time you use, a file selection window will pop up, requiring you to select a PSGraph file. For more information of how to create PSGraph file, please see
    - Link to GUI manual
3. Wait until the Tinker GUI launches. The Tinker GUI should within few seconds.



4. After the GUI shows up, click on the connect button. And your GUI should show up with connected button



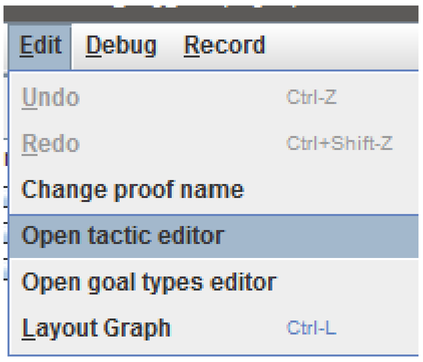
5. You can now use Tinker GUI to guide Rodin and prove your obligation

Important Notice

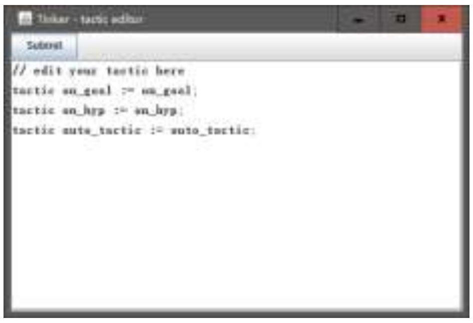
A. When you are creating any new PSGraph with TinkerGUI or making any modification, please ensure the proof session is closed (Clicked "STOP"
B. Every time you create a new PSGraph to work with Rodin, please
    a. Copy the following text

```
tactic on_goal := on_goal;
tactic on_hyp := on_hyp;
tactic auto_tactic := auto_tactic;
```

    b. In Tinker GUI, click menu -> Edit -> Open tactic editor



    c. And paste in the opened window as shown in the image:



```
// edit your tactic here
tactic on_goal := on_goal;
tactic on_hyp := on_hyp;
tactic auto_tactic := auto_tactic;
```

d. Click submit
e. File -> Save, and your PSGraph file is initialised.

The notice B is temporary, and will be changed in the future version.