

RoboTool

RoboChart Tool Manual

James Baxter

Alvaro Miyazawa

Pedro Ribeiro

Kangfeng Ye

April 2, 2025

This is the manual for the *RoboChart* tool. It describes the requirements, installation process and usage of the tool.

Contents

1	Requirements	5
2	Installation	7
2.1	Recommended approach	7
2.2	Alternative using the Eclipse installer	7
2.3	Using the update site	8
2.4	Updating the tool	12
3	RoboChart: usage	14
3.1	Creating a new project	14
3.2	Creating a new RoboChart package	15
3.3	Deleting a RoboChart package	17
3.4	Editing a RoboChart diagram	17
3.5	Deleting element from diagrams	19
3.6	Exporting diagrams as figures	19
3.7	Exporting projects	20
3.8	Importing projects	21
4	RoboSim: usage	22
4.1	Creating a new RoboSim package	22
4.2	Deleting a RoboSim package	23
5	Model Checking with FDR	24
5.1	Adapting the generated semantics	25
5.1.1	RoboChart semantics: CSP _M conventions	26
5.2	Opening FDR directly from Eclipse	27
5.3	RoboChart assertions language	27
5.3.1	Assertions	27
5.3.2	Optimisations	30
5.4	RoboSim assertions language	31
5.4.1	Assertions	31
5.4.2	Optimisations	33
5.5	Running FDR	34

5.6	Forbidden trace generation via FDR	34
6	Model Checking with PRISM	37
6.1	Invoking the translator	37
6.2	Configuration File	38
6.3	Opening PRISM directly from Eclipse	40
6.4	Probabilistic assertion language	40
7	Mutation Testing with Wodel	42
7.1	Exporting a RoboChart Model to XMI	43
7.2	Obtaining the RoboChart Metamodel	43
7.3	Writing Mutation Operators	44
7.4	Running Wodel	45
7.5	Importing Mutants into RoboTool	46
7.6	Generating RoboChart CSP Semantics	46
7.7	Generating Traces from Comparison with Mutants	47
A	Deviations from RoboChart reference manual	49
A.1	Well-formedness condition	50
A.2	Semantics	50

Requirements

- Linux Ubuntu LTS – Certain aspects of Eclipse seem to work differently in Windows, Linux and macOS. While our plugins may work on any platform that has an Eclipse distribution, we will focus our efforts on fixing bugs that occur on Linux. We recommend using the desktop session using Xorg rather than Wayland.¹ This can be selected in the login screen by clicking on the cog icon on the right-hand side, and selecting *Ubuntu on Xorg* as shown in Figure 1.1.
- Eclipse² 2021-12 – We recommend following the procedure in Section 2.1 using the Eclipse Installer, which installs RoboTool in one go, and therefore also Eclipse 2021-12.
- Java Runtime Environment – Recommended to use Java 17. If using the Eclipse Installer, a JRE can be installed via the Eclipse Installer if one is not already installed.
- FDR4 (optional) – <https://cocotec.io/fdr/> Needed if you want to analyse the automatically generated semantics of the RoboChart specifications.
- PRISM (optional) – <http://www.prismmodelchecker.org/> Needed if you want to analyse the manually generated probabilistic semantics of the RoboChart specifications.

¹Alternatively, if using Wayland, RoboTool can be run under X11 by setting the environment variable `GDK_BACKEND=x11` on the command line. For example: `export GDK_BACKEND=x11 && ./eclipse`

²The graphical editors of RoboChart and RoboSim are currently known to be incompatible with Eclipse 2022 and above.

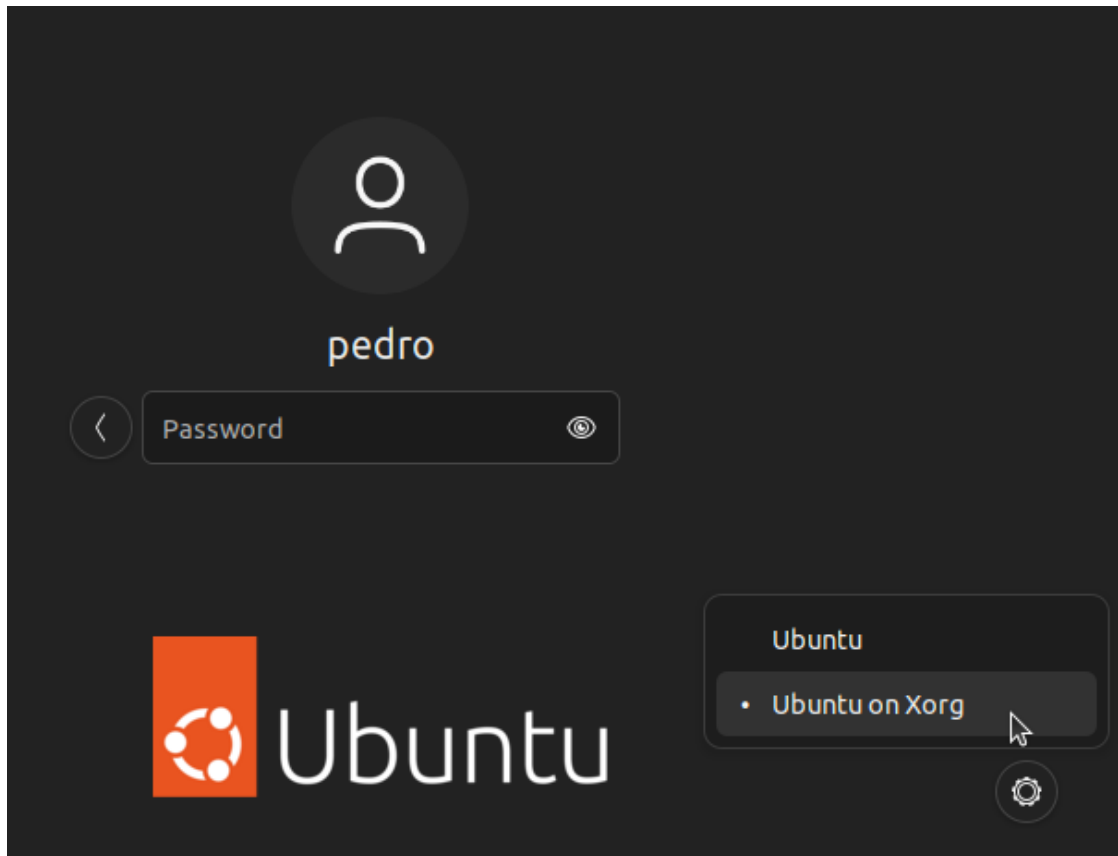


Figure 1.1: Choosing Xorg as the desktop session on Ubuntu 22.04 LTS

Installation

The installation of RoboTool, and Eclipse, can be achieved in different ways. We recommend using the approach detailed the next Section 2.1. An alternative method using an update site is described in Section 2.3 for more advanced users.

2.1 Recommended approach

Download the latest RoboTool release from <https://github.com/UoY-RoboStar/robotool/releases>. Choose the appropriate Operating System (Linux recommended) and architecture. Download the archive file, extract it and execute the Eclipse binary.

2.2 Alternative using the Eclipse installer

1. Download the latest Eclipse installer from <https://www.eclipse.org/downloads/packages/installer>. At the time of writing the latest version is 2023-09 R (internally displayed as 1.31.0 build 313). Afterwards, extract the installer and run it.
2. Click the hamburger button (\equiv) on the top-right corner. Deselect ‘Bundle Pools’ and click on `Advanced mode`.
3. Click the green plus button (+) on the top-right corner. A dialog titled “Add User Products” will appear. Enter the URL <https://robostar.cs.york.ac.uk/robotool.setup> and press `OK`. RoboTool should now appear selected under <User Products>. If not, select it.

4. Select a suitable Java VM (17+). If there is no version suitable in your machine, then select JRE 17.0.9 (or equivalent), so that a compatible Java Runtime Environment (JRE) will be installed. It is also recommended that ‘Bundle Pool’ is left disabled.
5. Click twice. If you wish to choose where to install RoboTool then select ‘Show all variables’ and configure the installation folders as appropriate, otherwise a default folder hierarchy will be chosen.
6. Click , and finally click .
7. During the installation, which typically takes 2 to 3 minutes depending on the network, you will be prompted to:
 - Accept the EPL license: we recommended toggling ‘Remember accepted licenses’.
 - Trust ‘unsigned content of unknown origin’: you will need to click on followed by .
8. At the end of this process, if everything succeeds, RoboTool will be launched automatically. You can then click on to quit the Eclipse installer. This RoboTool installation will contain every RoboTool plug-in and required dependencies (eg. Epsilon, Sirius, Xtext).

2.3 Using the update site

1. Download Eclipse 2021-12 (<https://www.eclipse.org/>) and install it. You can choose the *Eclipse Modeling Tools* package¹, available from <https://www.eclipse.org/downloads/packages/release/2021-12/r>. Eclipse can also be installed using the **Eclipse Installer**, that also provides the facility to install Java. At the time of writing, Java 17 is recommended. Further instructions are available at <https://wiki.eclipse.org/Eclipse/Installation> regarding installation of Eclipse;
2. Select

¹If a different Eclipse package is desired, for example, the *Eclipse IDE for Java Developers*, then before installing RoboTool you should ensure that Eclipse’s *Software Sites* are configured to consider versions of dependencies from the same Eclipse release. This can be set by going to , selecting and deselecting *Latest Eclipse Simultaneous Release*. See https://bugs.eclipse.org/bugs/show_bug.cgi?id=579806 for further details.

3. Type `https://robo-star.cs.york.ac.uk/robotool/update` in the `Work with` textbox and press `Enter`;

4. Select the plugins to suit your needs. If in doubt, you can install all available plugins. Currently, the following plugins are available, grouped by category as:

- **PRISM**

- **PRISM Metamodel** ECore metamodel targetting PRISM. This plug-in is required if using the facilities to generate PRISM artefacts for analysis of probabilistic models;

- **RoboCert:**

- **RoboCert Metamodel** ECore metamodel for working with RoboCert specifications;
- **RoboCert Textual Editor** provides support for writing RoboCert specifications;

- **RoboChart:**

- **RoboChart Assertions** for writing properties over RoboChart models for analysis using FDR and PRISM;
- **RoboChart Graphical Editor** is needed if you use the graphical notation of RoboChart;
- **RoboChart Metamodel** and **RoboChart Textual Editor** are essential for working with RoboChart models;

- **RoboChart Generator:**

- **RoboChart CSP Generator** for verification of RoboChart models using FDR;
- **RoboChart PRISM Generator** for verification of RoboChart models using PRISM. Please note that the **PRISM Metamodel** plug-in under the category **PRISM** must also be installed;

- **RoboChart Extras:**

- **RoboChart CSP Semantics Comparison** essential for mutation testing with Wodel;
- **RoboChart Model Exporter** and **RoboChart Model Importer** are also required for interfacing with Wodel;
- **RoboChart2Sim Metamodel** required for model transformation from RoboChart to RoboSim;

- **RoboSim:**

- **RoboSim Assertions** for writing properties over RoboSim models for analysis using FDR;
- **RoboSim Graphical Editor** is needed if you use the graphical notation of RoboSim;
- **RoboSim Metamodel** and **RoboSim Textual Editor** are **essential** for working with RoboSim models;
- **RoboSim Generator:**
 - **RoboSim CSP Generator** is required for analysis of RoboSim models with FDR;
- **RoboSim Physical Modelling:**
 - **Physical Modelling Graphical Editor** is required if you use the graphical **RoboSim** language for modelling physical platforms;
 - **Physical Modelling Metamodel** and **Physical Modelling Textual Editor** are **essential** if using the physical modelling language of RoboSim;
- **RoboSim Physical Modelling Generator:**
 - **PhysMod CyPhyCircus Generator** allows generating CyPhyCircus models from RoboSim models;
 - **PhysMod SDF Generator** allows generating SDF models from physical RoboSim models;
- **RoboTool Extras:**
 - **RoboTool Versioning** automatically documents the version of RoboTool plug-ins used in a project, suitable for archival;

Please note that occasionally upon installation of RoboTool plug-ins there may be an error reported similar to the following:

```
An error occurred while collecting items to be installed
session context was:(profile=epp.package.java,
phase=org.eclipse.equinox.internal.p2.engine.phases.Collect, operand=, action=).
No repository found containing: osgi.bundle,org.eclipse.epsilon.common,2.4.0.202203041826
No repository found containing: osgi.bundle,org.eclipse.epsilon.common.dt,2.4.0.202203041..
```

This is an intermittent issue with the installation of Epsilon, a plug-in that is required by the RoboTool plug-ins **RoboChart PRISM Generator** and **RoboChart2Sim Metamodel**. In this case we suggest deselecting such plug-ins and installing Epsilon separately (see installation step 11), if required, before those that depend on Epsilon.

5. Click ;
6. Click ;
7. Accept the license and click ;
8. A security warning will appear; click the button if you wish to continue with installation;
9. Once the plugins are installed, a popup box will appear requesting Eclipse to be restarted. Click .
10. Verify you installation by:
 - a) Click the menu item;
 - b) Click the ;
 - c) Look for *RoboChart Feature* in the list of Installed Software;
 - d) Click the arrow to the left of *RoboChart Feature*;
 - e) The *RoboChart Feature* should contain four sub-items: Sirius Core Runtime, Sirius Runtime IDE, Sirius Runtime Support AQL, and Sirius Specifier Environment.
11. For verification with PRISM, an additional plugin Epsilon is needed.

- a) Select
 - b) Type `https://download.eclipse.org/epsilon/updates` in the textbox and press
 - c) Select “Epsilon Core”, “Epsilon Core Development Tools”, and “Epsilon Development Tools for EMF” under “Epsilon EMF Integration”;
 - d) Click
 - e) Click
 - f) Accept the license and click
 - g) A security warning will appear; click the button if you wish to continue with installation;
 - h) Once the plugins are installed, a popup box will appear requesting Eclipse to be restarted. Click .
12. You are done. If you have any problems with the installation or usage, check for an associated issue or create a new issue on <https://github.com/UoY-RoboStar/robochart-textual>.

2.4 Updating the tool

In order to update the tool, there are a few alternative ways:

1. Click
2. If there are updates available, the installed features will be available; select them and click twice
3. Accept the license agreement, and click
4. A security warning will appear; click the button if you wish to continue with installation;
5. Once the plugins are installed, a popup box will appear requesting Eclipse to be restarted. Click .

While convenient, the feature looks for all updates, not just RoboChart updates, and it can take longer. Additionally, it may lead to accidental updates. If you want to update only the RoboChart tool, follow these steps:

1. Click
2. Click the button
3. In the tab, find the installed features and select one of them;
4. Click the button . This will search for updates specific to the selected feature;
5. If an update is found, a details page will be shown, click ;
6. Accept the license agreement, and click ;
7. A security warning will appear; click the button if you wish to continue with installation;
8. Once the plugins are installed, a popup box will appear requesting Eclipse to be restarted. Click .

RoboChart: usage

The RoboChart plugins are based on the Sirius framework. A generic user's manual for Sirius based tools is available in <http://www.eclipse.org/sirius/doc/user/Sirius%20User%20Manual.html>. The next sections describe basic activities specific to our tool.

A RoboChart specification is a collection of named or anonymous packages (RCPackage in the abstract syntax). In the tool, we associate a RoboChart specification with a Modeling project, and a package with a file with the extension `rct`. An `rct` file records information about the elements defined in a package, but not how they are represented graphically. Packages recorded as `rct` files can be visualised and edited through a graphical editor. These diagrams are recorded in an `aird` file. For this reason, we first create a `rct` file, and then create a diagram to represent that file. Note that it is possible to have multiple graphical representations of the same package.

The next section describes how to create a project (i.e., RoboChart specification), and the following section describes how to create new `rct` files (i.e., RoboChart packages).

3.1 Creating a new project

1. Select Modeling perspective, if it is not yet selected:
 - a) Click the `Window > Perspective > Open Perspective > Other ...` menu item;
 - b) Select the item `Modeling` from the list; and
 - c) Click `OK`. The *Modeling* perspective is divided in four main parts:
 - i. *Model Explorer* – top left corner
 - ii. *Outline* – bottom left corner

iii. *Text editor* – top right corner

iv. *Properties, Problems and Console* – bottom right corner

2. Create a new Modeling project

- a) Right click the *Model Explorer* window and select **New** **Project**; alternatively, in the menu click **File** **New** **Project**;
- b) Either type *Modeling* in the wizard text box, or open the *Sirius* folder;
- c) Select the item *Modeling Project* and click **Next**;
- d) Type the project name and click **Finish**



3. In order to browse the contents of the project click the arrow to the left of the project name in the Model Explorer; the new project will contain two initial components:

- a) Project Dependencies – This can be used to add references to external packages, but I have not used it;
- b) *representations.aird* – This file stores the graphical representation of all the diagrams.

3.2 Creating a new RoboChart package

A project can have several packages (*rct* files). Here, we describe how to create a package inside a project.

1. Select the project in which you wish to create a new package;
2. Press **Ctrl+N** and select **General** **File**; or right click the project and select **New** **File**; or in the menu click **File** **New** **File**;
3. Type the name of the file with extension *.rct* and press **Finish**;
4. The first time you create a package in a new project, a popup will appear asking to convert the project into an Xtext project. This is necessary, so click **Yes**;

5. The first time you create a package in a new project, right click the project and select *Viewpoint Selection*. A popup window will appear with an item *RoboChart*; tick this item and click ;
6. In order to open a diagram follow the following steps:
 - a) Click the arrow to the left of the name of the *rct* file; there should be a subitem *RCPackage* under the file name¹;
 - i. If there is no arrow to the left of *RCPackage*, or no subitem of type diagram (marked by the icon , right click *RCPackage* and click , where {name} is the name of the file without the extension². Finally, click (this creates a new diagram associated with the *rct* file);
 - ii. If there is already a diagram under *RCPackage*, do nothing.
 - b) Finally, double click a diagram (identified by the icon ) to open it.

An alternative, more automated way of creating a RoboChart involves the use of the RoboChart Wizard. Notice that this is a newer feature and more prone to errors, so if it fails, try the steps above.

1. Select the project in which you wish to create a new package;
2. Press , or right click the project and select ; or in the menu click ;
3. Type *RoboChart* in the wizard text box, or find the RoboChart folder and open it;
4. Select the item *New RoboChart Package* and click ;
5. Change the file name (it must end in *.rct*) and click ;
6. The first time you create a package in a new project, a popup will appear asking to convert the project into an Xtext project. This is necessary, so click ;
7. The wizard will create the file, select the appropriate viewpoint, create the diagram and open it automatically.

¹Sirius does not seem to behave consistently when creating a new file, sometimes it creates the diagram automatically, other times it does not.

²If you wish, you can change the name.

3.3 Deleting a RoboChart package

Since a RoboChart package is associated with an `rct` file and a diagram (in the `aird` file), we need to delete both the `rct` file and the diagram³.

1. In order to delete a RoboChart package, that is, the `rct` file and the associated diagrams:
 - a) Select the file, press `Delete` (or select delete on the context menu) and click `OK`;
 - b) Click the left arrows on the file `representations.aird`, the subitem `RoboChart` and the subitem `Package`. The names of the diagrams associated with the deleted file will be in a lighter font. Select them, press `delete` (or right-click and click `Delete`) and confirm the deletion by clicking `OK`.

3.4 Editing a RoboChart diagram

1. Use the palette to the right to select objects to add to the diagram. For most objects (except connection) you can either drag and drop the object to the diagram, or select and click (if you keep `Ctrl` pressed, it is possible to click multiple times on the diagram to create multiple instances of the object). After creating a new component (e.g., type), make sure to save the diagram (press `Ctrl` + `S` or click `File` >> `Save`) to guarantee the newly created element can be used.

Notice that the palette may not show all tools depending on the size of the screen. It is divided into category boxes that group similar tools. In order to explore the available tools you can click a category box to close it, and click once more to fully expand it.

It is also possible to use the arrows at the bottom and top of the category box (or the scroll wheel on your mouse) to scroll through the available tools.

If you are familiar with the icons, it is possible to reduce the space taken by the tools by removing the text and showing only the icons. To do this, right click the palette and select `Layout` >> `Icons Only`

³We must delete the diagram associated with the RoboChart package, not the whole `aird` file.

2. To use connections, select the appropriate connection tool (e.g., transition), click the source node and then the target node⁴;
3. To change labels (e.g., interface names, transition labels) there are a few options:
 - a) Select the item and start typing. This option will delete the original text; to avoid this use one of the next two options; or
 - b) Press **F2** and the text will become editable; or
 - c) Click once over the text and it will become editable.

Label editing has some limitations that are worth mentioning:

- a) Not all labels editors are implemented yet (if you think label editing for a specific feature is missing, create an issue in the bitbucket issue tracker);
- b) If you type a label with syntax errors, the edit will fail and the previous text will show (potentially empty text). transitions, the failure text will be stored internally and shown again if you press **F2** or click the label (you can then fix the error directly instead of retyping the whole text again).

4. Most textual elements (variable declarations, operations, actions etc) are input through a pop-up window that provides a hint of the syntax and parses the input before creating the element. For example, when creating a new action in a state:
 - a) a pop-up windows will appear requesting the text of the action in the format [(entry|exit|during) Action], that is, one of the keyword **entry**, **exit** or **during** followed by an action⁵
 - i. if you only type entry, for instance, the box will show Cannot parse below the text box and keep the button **OK** disabled
 - ii. if, on the other hand, you type entry skip, the tool parses the action correctly and enables the **OK** button. If you click **OK** the action will be added to the state.
5. To save a diagram, press **Ctrl** + **S** or click **File** >> **Save**.

⁴While you can use **Ctrl** + **Z** to undo changes to the diagram, it can create inconsistencies, so use it carefully.

⁵See the language reference manual for the concrete syntax of actions and other elements.

3.5 Deleting element from diagrams

To delete an object from the diagram, select it and press `delete`; avoid using the option `Right-click` `Edit` `Delete from Model` as it can make the diagram inconsistent.

Be careful when deleting elements:

1. In general, make sure they are not used elsewhere. For example, when deleting an event used as the trigger of a transition, the transition will point to a non-existent event; make sure to remove the reference first;
2. We have implemented controlled deletions for Interfaces and Types, but not for other elements⁶.
 - a) When deleting an interface, the tool will delete all required and provided references and try to replace call to the operations in the interface by other operations of the same name in scope; if this is not possible, operation calls are converted into `skip`
 - b) When deleting a type, the tool will look for uses of that type, if none are found, the type is deleted. Otherwise, a list of types is offered to the user to replace the deleted type.

3.6 Exporting diagrams as figures

It is possible to export the diagrams as figure⁷.

To export all the diagrams at the same time:


1. Click the arrow to the left of the file `representations.aird` to expand it.
2. Expand the items `RoboChart` and `Package`.
3. Use `Ctrl` + `left click` to select multiple diagrams
4. Right click the selection and click `Export representations as images`

⁶If and when I implement more cases, I'll add them to this manual.

⁷The available formats are JPG, PNG, SVG, BMP and GIF. Exporting SVG will also generate a PDF equivalent.

5. A popup window will appear allowing you to select the target directory and the image format; after selecting the directory and the format, click **OK**.

To export a specific diagram, follow these steps:

1. Open the diagram (if it not already open)
2. If any diagram elements are selected, deselect them by pressing **Esc** or clicking in the background of the diagram
3. The toolbar at the top of the diagram contains a camera icon ; click it
4. A popup window will appear allowing you to select the target file and the image format; after selecting the file and the format, click **OK**.

If you right click `representation.aird`, an option **Export representations as images** is available. While this option works and generates images for all diagrams, it may have a side effect of closing all open diagrams, and making the project tree inconsistent. In this case, collapse the project by clicking the arrow to the left of the project name, and expand it again. This should refresh the project and offer to reopen the diagrams.

3.7 Exporting projects

Since the tool is under development, it is advisable to not only frequently save your specifications, but also export them as compressed files. To do that follow these steps:

1. Right-click the project in the Model Explorer and click **Export**
2. Select **General** >> **Archive File** and click **Next**
3. Give a name to the compressed file (or select an existing file using the **Browse** button) and click **Finish**

3.8 Importing projects

To import a RoboChart specification⁸ into eclipse:

1. Right-click the Model Explorer and click (alternatively, click)
2. Select and press
3. Choose the option and press the button
4. Find the compressed file (zip or tar file) that contains the specification and press
5. The project should be selected in the window below. In this case, click . If the project is not selected in the window, it is because a project with the same name already exists in the workspace; either delete the project or rename it before trying to import again.

⁸This should have been exported as described in the previous section

RoboSim: usage

Similarly to RoboChart, the RoboSim plugins are also based on the Sirius framework. This chapter discusses specifics of working with RoboSim models.

A RoboSim specification is a collection of named or anonymous packages (also RCPackages in the abstract syntax). Importantly, RoboSim packages are contained in files with the extension `.rst`. Such files record information about the elements defined in the package, but not how they are represented graphically. Diagrams are recorded in a unique `.aird` file¹ per project. A project may contain both RoboChart and RoboSim packages and diagrams. This is crucial, for example, to verify that a RoboSim simulation is correct with respect to a RoboChart model, as explained later in Section 5.4.



The following sections discuss how to use RoboSim packages. They are similar in nature to RoboChart packages, so their usage is similar to that described in Chapter 3. Here, we focus on aspects that are specific to RoboSim packages.

4.1 Creating a new RoboSim package

A project can have several packages (`.rst` files). Here, we describe how to create a RoboSim package inside a project.

1. Select the project in which you wish to create a new package;
2. Press `Ctrl+N` and select `General >> File`; or right click the project and select `New >> File`; or in the menu click `File >> New >> File`;

¹Sirius has an experimental mode that allows each diagram to be saved in separate files within a folder named `.representations`. This can be enabled by setting the system property `createLocalRepresentationInSeparateResource` to `true`, for example, by passing it as a parameter to the Java VM as `-DcreateLocalRepresentationInSeparateResource=true`. For further details see https://www.eclipse.org/sirius/doc/6.0.x/developer/representations_lazy_loading.html.

3. Type the name of the file with extension *.rst* and press ;
4. The first time you create a package in a new project, a popup will appear asking to convert the project into an Xtext project. This is necessary, so click ;
5. The first time you create a package in a new project, right click the project and select *Viewpoint Selection*. A popup window will appear with an item *RoboSim*; tick this item and click ;
6. In order to open a diagram follow the following steps:
 - a) Click the arrow to the left of the name of the *.rst* file; there should be a subitem *RCPackage* under the file name;
 - i. If there is no arrow to the left of *RCPackage*, or no subitem of type diagram (marked by the icon ) , right click *RCPackage* and click , where {name} is the name of the file without the extension². Finally, click (this creates a new diagram associated with the *.rst* file);
 - ii. If there is already a diagram under *RCPackage*, do nothing.
 - b) Finally, double click a diagram (identified by the icon ) to open it.

4.2 Deleting a RoboSim package

Since a RoboSim package is associated with an *.rst* file and a diagram (in the *aird* file), we need to delete both the *.rst* file and the diagram³.

1. In order to delete a RoboChart package, that is, the *.rst* file and the associated diagrams:
 - a) Select the file, press (or select delete on the context menu) and click ;
 - b) Click the left arrows on the file *representation.aird*, the subitem *RoboSim* and the subitem *RCPackage*. The names of the diagrams associated with the deleted file will be in a lighter font. Select them, press (or right-click and click) and confirm the deletion by clicking .

²If you wish, you can change the name.

³We must delete the diagram associated with the RoboSim package, not the whole *aird* file.

Model Checking with FDR

RoboTool provides facilities for running the CSP model checker – FDR – on user specified assertions directly via the tool. Currently, both RoboChart and RoboSim models can be analysed using FDR. Provided there are no errors, RoboTool automatically calculates and stores under the `csp-gen` folder the definition of the semantics using CSP_M , the machine-readable version of CSP accepted by FDR. This consists of files with the extension `csp` organised according to the tree structure outlined in Figure 5.1a.

RoboChart and RoboSim models can span one or more packages, defined in model files with extensions `rct` and `rst`, respectively. For each such file, a corresponding `.csp` file is generated by RoboTool under the directory `csp-gen`: if a file defines a **package** named `P`, then the generated file is named `P.csp`, and otherwise it takes the name of the `.rct`, or `.rst`, file concatenated after a prefix `file_`. At the root of `csp-gen` we have an untimed CSP semantics for RoboChart models, with auxiliary `.csp` files contained within the subfolder `defs`. A timed semantics for RoboChart models, defined using *tock*-CSP is contained within the subfolder `csp-gen▶timed`, that is structured similarly to `csp-gen`. Likewise, the *tock*-CSP semantics of RoboSim models is contained within the subfolder `csp-gen▶sim`.

Within RoboTool, the actual filesystem structure can be seen, for example, using the Project Explorer view. In Figure 5.1b we reproduce this view for an example, where a subset of the generated `.csp` files is visible.

The remaining sections of this chapter are structured as follows. Section 5.1 discusses how to use and adapt the generated semantics for verification. Section 5.2 discusses how to open FDR from within Eclipse. Section 5.3 describes the language used to specify assertions, and Section 5.5 describes the steps to run FDR from inside RoboTool.

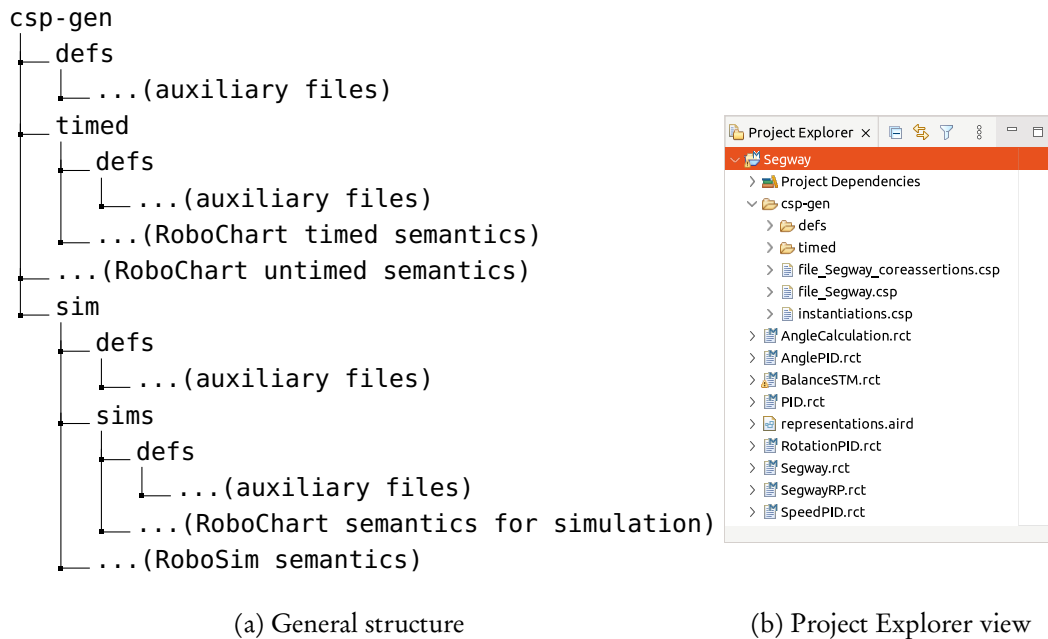


Figure 5.1: Filesystem structure of CSP_M semantics as generated by RoboTool

5.1 Adapting the generated semantics

RoboTool automatically generates the CSP_M semantics of each construct of a model, provided there are no errors concerning that construct, namely that all relevant well-formedness conditions are satisfied. The well-formedness conditions for RoboChart models are listed in [2, Chapter 3], while those for RoboSim models are listed in [1, Chapter 3]. Additionally, it generates a `instantiations.csp` file and `_coreassertions.csp` files for each model file.

These files use a special annotation to indicate which parts of the file can be re-generated. They have annotations of the form `-- generate ID`, where `ID` is any identifier, followed by excerpts of CSP. The specification under the comment will be regenerated every time the model is saved. If you do not want the specification under the comment to be re-generated, add a `not` after the identifier (`-- generate ID not`). This will cause the generator to ignore that part of the specification.

The `instantiations` file provides default values for constants, functions and bounds, and the `assertions` file provide default assertion checks where relevant. Use these files and the annotations above to add your own assertions and values without losing them on re-generation.

Obs.: If you delete these files, they will be regenerated but the changes will be lost.

5.1.1 RoboChart semantics: CSP_M conventions

The CSP_M semantics of RoboChart components is structured using modules¹ named after model elements, and nested according to the hierarchy of a RoboChart model. For example, the semantics of a Module M is defined by a parametrised process $D_{--}(id, c1, \dots, ci)$ in a CSP_M module named M , that can be referred to using the syntax $M::D_{--}(id, c1, \dots, ci)$, where $::$ is a delimiter and:

- id is an identifier (used in a semantics for collections []). It can be set to a default of \emptyset when instantiating the process for analysis;
- $c1$ to ci are constants used by M .

Effectively, in CSP_M $D_{--}(id, c1, \dots, ci)$ is a function that yields a complete CSP process. Within the CSP_M module M , there are declarations for all channels used in the semantics of that module. For example, the RoboChart event obstacle in a module M is modelled as a channel `obstacle` within M and can be referred using the notation $M::obstacle$. Events may be typed, so a corresponding CSP_M channel is typed accordingly as `event.InOut.type`, where the possible values of `InOut` are `in` to refer to an input event, or `out` for output events, and `type` is the actual event type modelled in CSP_M .

Similarly, a Controller C defined in a module M is defined in a nested CSP_M module of M . Its full name is $M::C$, and its semantics is given by a $M::C::D_{--}(\dots)$ process. Likewise, the same convention applies for state machines within Controllers.

In addition, for each D_{--} process there is a similar O_{--} process with the same parameters, that employs compression functions to reduce the state space for model-checking with FDR. A special process `AS_O_{--}` also exists in the semantics of Modules that allows writes to platform variables to be observed for the purpose of specialised analysis. We observe that these conventions are subject to change, and that specialised processes may be withdrawn in the future.

¹See <https://cocotec.io/fdr/manual/cspm/definitions.html#modules>

5.2 Opening FDR directly from Eclipse

The semantics of the RoboChart and RoboSim models is generated automatically in the `csp-gen` folder as a collection of `.csp` files, as previously described. Each file at the root folder `csp-gen` (or `csp-gen > timed` in the case of the *tock*-CSP semantics, and `csp-gen > sim` for the semantics of RoboSim) is self-contained and can be opened in FDR to perform analysis and simulation. The actual definitions of the elements of the model can be found in the files in the folder `defs`.

It is possible to open the `.csp` files directly from the `Model Explorer`.

If you installed FDR on Ubuntu, it is possible that FDR is the default program for `.csp` files. In this case, it may be possible to open the file in FDR by double clicking the file. If, instead of FDR, the file is open in Eclipse, right-click the `.csp` file, select `Open with >> System Editor`.

If FDR is not the default program for `.csp` files, you can add it to the list of editors in Eclipse as follows:

1. Right-click a `.csp` file and select `Open with >> Other`
2. Click `Browse...`, find the `fdr4` executable and click `OK`
3. Select `External programs`, and check both boxes below the `Browse...` button
4. Click `OK`

Once you follow these steps, every time you double click a `.csp` file in Eclipse, FDR should open. If you want to open the `.csp` file in a text editor, right-click the file, and select `Open with >> Text Editor`. To open the file in FDR again, right-click the file, and select `Open with >> fdr4`.

5.3 RoboChart assertions language

RoboTool provides a textual editor to support the specification of assertions, regarding RoboChart models, in files with extension `.assertions`.

5.3.1 Assertions

An assertion is specified as follows:

assertion = (**untimed** | **timed**)? **assertion** *NAME* : *specification* (**in the CSP Model**)?
 (**with parameters**)? (.)?

specification =
ELEMENT is (not)?
 (**deadlock-free** | **divergence-free** | **deterministic** | **timelock-free**) |
ELEMENT (does not terminate | **terminates)** |
STATE is (not)? reachable in ELEMENT |
ELEMENT (refines | **equals** | **does not refine** | **is not equal to**) **ELEMENT** |
clock CLOCK is (not)? initialised

CSP Model = **failures divergences model** | **traces model** | **failures model** |
tick-tock model

parameters = *AssertionParameter* (, *AssertionParameter*)* (, **and** *AssertionParameter*)?

AssertionParameter = *constant* | *parameter*

constant = **constant** *NAME of NAME* (**assigned** | **set to** | **with value**) *Expression*

parameter = **parameter** *NAME of NAME* (**assigned** | **set to** | **with value**) *Expression*

In the description above, *NAME* is any valid identifier, *ELEMENT* is the name of any state machine, controller, module or collection, *STATE* is any state in a state machine, and *CLOCK* is any clock name. Below is an example of an assertion:

```

1 assertion A1: LMachine is deadlock-free with constant
2   SET_SPACE_TIME of LMachine set to 1,
3   SET_SPACE_BOUND of LMachine set to 1,
4   CHANGE_LANE_TIME of LMachine set to 1,
5   CHANGE_LANE_BOUND of LMachine set to 1,
6   JOIN_DISTANCE_TIME of LMachine set to 1,
7   JOIN_DISTANCE_BOUND of LMachine set to 1,
8   and maxPL of LMachine with value 3

```

While it is not necessary to specify all required or defined constants in an assertion, the tool will give a warning indicating which constants are missing.

The value of unspecified constants is either a default value for the type of the constant (for example, 0 is the default value of the type `nat`), the value stored in the `instantiations.csp` file, or the initial value specified in the model.

Values specified in an assertion override the usual definition.

While the assertion language supports comparison of constructs (refinement and equality)², this may not be enough for the verification of more general properties. In order to address this limitation, we allow the specification of general CSP processes in an `.assertions` file, which can then be used in an assertion as a member of the class *ELEMENT*. The syntax that allows the specification of CSP processes in an `.assertions` file is as follows:

```
CSP specification =  
  (untimed | timed) csp PROCESS_NAME (associated to ELEMENT)?  
  csp-begin CSP-M csp-end
```

A *CSP specification* declares an *ELEMENT* with name n (*PROCESS_NAME*), that can be declared to be **associated to** a RoboChart *ELEMENT*. This information should be provided when the specification uses the semantics of a RoboChart component, so that the CSP_M generator can automatically bring into context the semantics of *ELEMENT* and the definition of data types used. The content *CSP-M* is a CSP specification in the syntax accepted by FDR. This specification must include one process called n .

Assertions involving custom *CSP specifications* using the **tick-tock model** require the declaration of a CSP_M constant named `PROCESS_NAME__sem__events`, where `PROCESS_NAME` is the name of the specification, within *CSP-M* that records the alphabet of the process. An example of its use is reproduced below, where `SpecStm0__sem__events` is defined on line 6 as a channel set, that includes the channels `stm1::in1`, `stm1::in2`, `stm1::in3` and `stm1::terminate`.

```
1 timed csp SpecStm0 associated to stm0 csp-begin  
2 SpecStm0 = stm0::0__ (0) [[   stm0::in1 <- stm1::in1,  
3                             stm0::in2 <- stm1::in2,  
4                             stm0::in3 <- stm1::in3,  
5                             stm0::terminate <- stm1::terminate ]]  
6 SpecStm0__sem__events = { |stm1::in1,stm1::in2,stm1::in3,stm1::terminate| }  
7 csp-end  
8  
9 timed assertion TT0 : SpecStm0 refines stm1 in the tick-tock model
```

²As of a recent version of the RoboChart CSP Generator, it is now possible to write refinement assertions between two RoboChart components, whereby events in the semantics are renamed automatically so that the comparison is meaningful. This mechanism does not support data refinement. Such cases should be handled via custom CSP specifications.

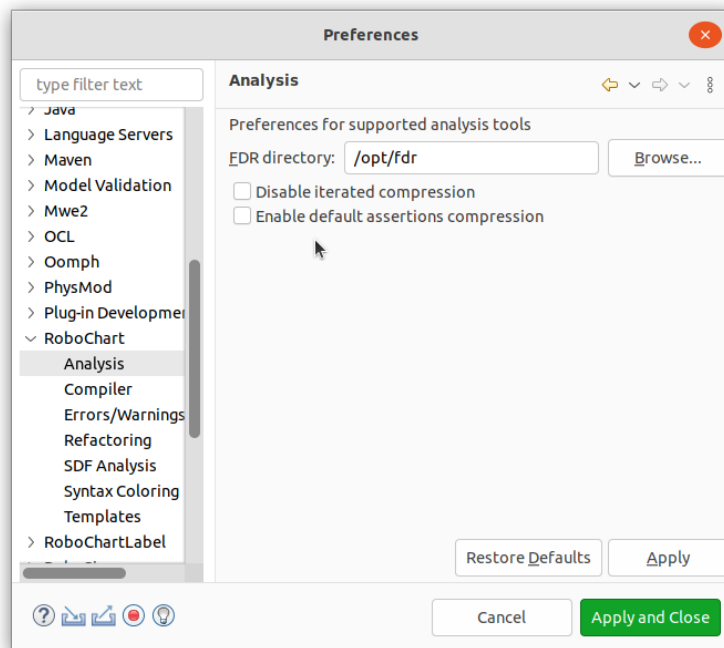


Figure 5.2: Optimisation options for generation of RoboChart semantics.

5.3.2 Optimisations

When checking models using FDR, performance may vary depending on the complexity of RoboChart models and the instantiation of data types. By default, the CSP_M generator attempts to generate an efficient representation for model-checking, including the use of compression functions offered by FDR. Some of these options can be toggled within RoboTool if analysis performs poorly by clicking on `Window >> Preferences` and selecting `RoboChart >> Analysis`. The current list of options, as shown in Figure 5.2, is described below:

- **Disable iterated compression** (default: **off**): this option controls how the semantics of the behaviour of state machines is composed in parallel with that of their memory. The default option generates a sequence of parallel compositions that are subsequently compressed using FDR's `dbisim` and `sbisim` functions. This works well when assignments to local variables constrain behaviours of a state machine or when some variables are not actually used in the model.
- **Enable default assertions compression** (default: **off**): this option controls whether the verification of assertions written using the RoboChart assertions language uses op-

timised semantics, that is, 0_{\dots} processes, instead of D_{\dots} processes. Because refinements are usually checked against much less complex specifications, compressing the semantics of a RoboChart component may not be efficient overall for property verification. On the other hand, if the RoboChart component is used as a specification in a refinement assertion, then this compression may be useful.

5.4 RoboSim assertions language

Similarly to the case with RoboChart assertions, RoboTool provides a textual editor to support the specification of assertions with respect to RoboSim models, in files with extension `.rsa`.

5.4.1 Assertions

Here, we focus on the syntax of RoboSim assertions. Syntax rules common to RoboChart, specified in Section 5.3.1, are generally omitted.

assertion = **assertion** *NAME* : *specification* (**in the CSP Model**)? (**with parameters**)? (.)?

The specification of assertions is similar to that for RoboChart, but includes the possibility to check whether a RoboSim component, namely a specification of all valid simulations defined over a RoboChart component, is **schedulable**, as discussed in [].

specification =
ELEMENT **is** (**not**)? (**deadlock-free** | **divergence-free** | **deterministic** |
ELEMENT (**does not terminate** | **terminates**) |
STATE **is** (**not**)? **reachable in** *ELEMENT* |
ELEMENT (**refines** | **equals** | **does not refine** | **is not equal to**) *ELEMENT* |
ELEMENT **is** (**not**)? **schedulable**

In addition to setting the value of **constants** and **parameters**, RoboSim assertions allow setting the **cycle** of SimModules, SimControllers and SimMachineDefs using the syntax below.

AssertionParameter = *constant* | *parameter* | *cycle*

cycle = **cycleDef** of *ELEMENT* (**assigned** | **set to** | **with value**) *Expression*

Similarly to RoboChart assertions, arbitrary CSP processes may also be written, as specified by the syntax rule *CSPPProcess*. In addition, RoboSim extends this mechanism to allow the specification of *SimulationInstantiations*, that define all valid (RoboSim) simulations of a RoboChart component, suitable for verification.

CSP specification = *CSPPProcess* | *SimulationInstantiation*

SimulationInstantiation =

simulation *NAME* **of** *ELEMENT* { **cycleDef** *Expression* } |
simulation *NAME* **of** *ELEMENT* { **cycleDef** *Expression* **InputContext** **OutputContext** }

A simulation instantiation must be *NAMED* and defines an *ELEMENT* of the same name, so that it can be used in an assertion. After the **of** keyword a RoboChart *ELEMENT* must be provided. A **cycleDef** must also be defined, and optionally an **input context** or **output context** can be defined like in RoboSim, as specified by the following syntax rules.

InputContext = **input context** { (**uses** *Interface* | **requires** *Interface* | *Event*)* }

OutputContext = **output context** { (**uses** *Interface* | **requires** *Interface* | *Event*)* }

Although RoboChart Modules and Controllers contain sufficient information to automatically determine its inputs and outputs from a RoboSim point-of-view, state machines may require further input from the user. For example, a required operation of a RoboChart state machine may be defined as a software operation, or required as robotic platform operation, and so this needs to be explicitly captured in an **output context** when defining a simulation. An example showing its use is reproduced below.

```

1 simulation SimSTM of M::STM {
2   cycleDef cycle==2
3   input context { event In:nat }
4   output context { event Out:nat }
5 }
6
7 assertion RS2 : STM equals SimSTM in the failures divergences model
8   with cycleDef of STM set to 1,
9   and cycleDef of SimSTM set to 1

```

Here, we have the definition of a **simulation** named *SimSTM*, obtained as all valid simulations of a RoboChart state machine *M::STM*, where: the **cycle** is set by default to 2, and the **input context** considers the event *In*, and the **output context** considers the event *Out*. Then, we have the definition of an **assertion** *RS2*, that checks whether all valid simulations of *M::STM*,

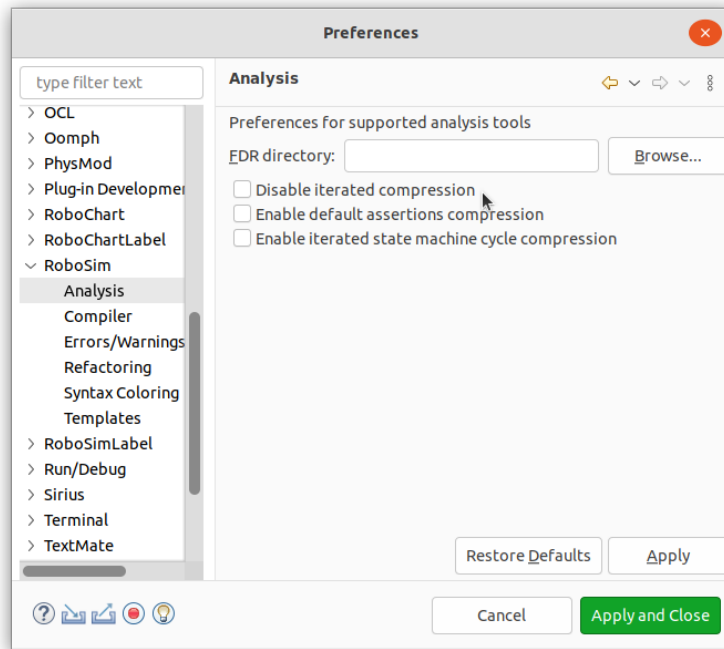


Figure 5.3: Optimisation options for generation of RoboSim semantics.

defined as `SimSTM` behave exactly as `STM`, a RoboSim state machine, where the `cycle` has been set to `1`.

5.4.2 Optimisations

Similarly to the verification of RoboChart assertions, when model checking with FDR, performance varies depending on the complexity of RoboSim models and the instantiation of data types. By default, the CSP_M generator attempts to generate an efficient representation for model-checking, including the use of compression functions offered by FDR. Some of these options can be toggled within RoboTool by clicking on `Window` `Preferences` and selecting `RoboSim` `Analysis`. The current list of options, as shown in Figure 5.3, is described below:

- **Disable iterated compression** (default: `off`): this option controls how the semantics of the behaviour of state machines is composed in parallel with that of their memory. The default option generates a sequence of parallel compositions that are subsequently compressed using FDR's `dbisim` and `sbisim` functions. This works well when assignments to local variables constrain behaviours of a state machine, or, when certain variables are not actually used.

- **Enable default assertions compression** (default: **off**): this option controls whether the verification of assertions written using the RoboSim assertions language uses optimised semantics, that is, **O__** processes, instead of **D__** processes. Because refinements are usually checked against much less complex specifications, compressing the semantics of a RoboSim component may not be efficient overall for verification.
- **Enable iterated state machine cycle compression** (default: **off**): this option controls how the semantics of the behaviour of a RoboSim state machine is composed with the semantics of its cycle. When enabled, the cycle is composed in parallel with the state machine, and then the semantics of inputs (modelling state) is composed in parallel, but iteratively using FDR's **dbisim** and **sbisim** compression functions. This option can work well when the value of inputs constrains the behaviour of a state machine, or, when certain inputs are not actually used by a state machine.

5.5 Running FDR

In order to run an `.assertions` file in FDR via RoboTool, the user must select the file in the `model explorer` or `project explorer` window, and either:

1. Click the `Run...` button on the FDR tool bar; or
2. Right-click the file, and click `RoboTool >> CSP >> Run...`.

In both cases, FDR will verify the assertions in the background, providing some progress information in the `Progress` window, and upon completion a HTML report is generated and displayed in the tool. This facility is currently only available for RoboChart assertions.

5.6 Forbidden trace generation via FDR

RoboTool provides a facility to calculate a set of forbidden traces from RoboChart and RoboSim models via its integration with FDR. To make use of this facility, FDR must be installed and activated. Then, open a RoboChart or RoboSim model, and using the `Model Explorer` navigate to the desired component by expanding the tree, as shown in the following example:

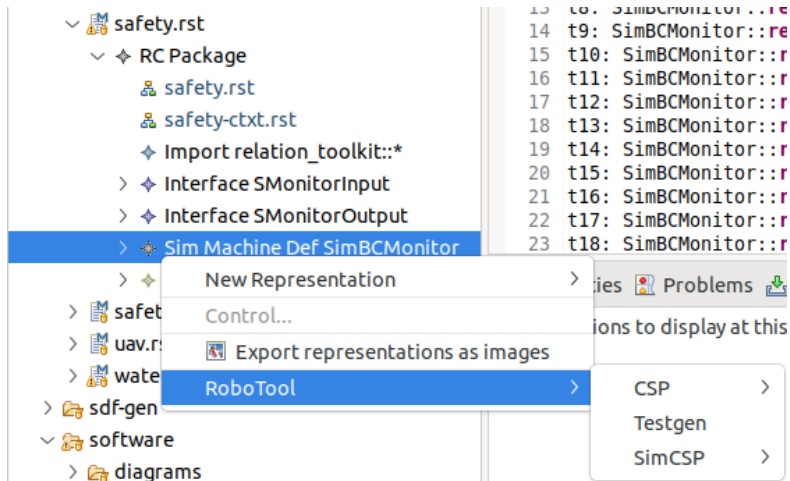


Figure 5.4: Selecting a model element for generating forbidden traces.

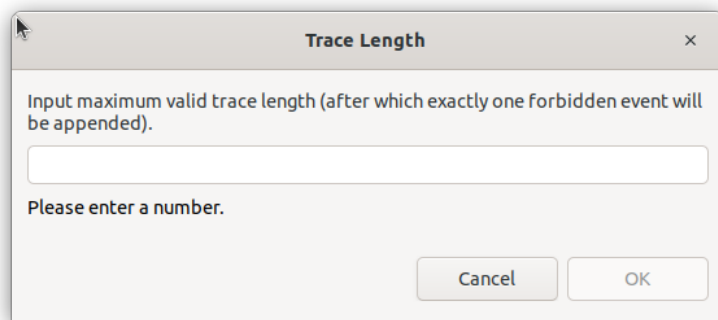


Figure 5.5: Forbidden trace dialog.

Then, right-click the component and select `RoboTool`. The first time it is selected the drop-down menu to the right is empty, so select `RoboTool` again so that the option `Testgen` is shown. A sequence of dialogs will appear asking for a maximum valid trace length to be considered when generating forbidden traces. In the case of RoboChart only, a sequence of further dialogs will appear asking whether a JSON representation is desired and whether traces should be timed.

The progress for generating forbidden traces can be tracked by examining the Progress tab as shown below.

Forbidden traces will be generated under a folder `test-gen` with a file of with extension `.rtspec` of the same name as that containing the model element. Please note that if the file

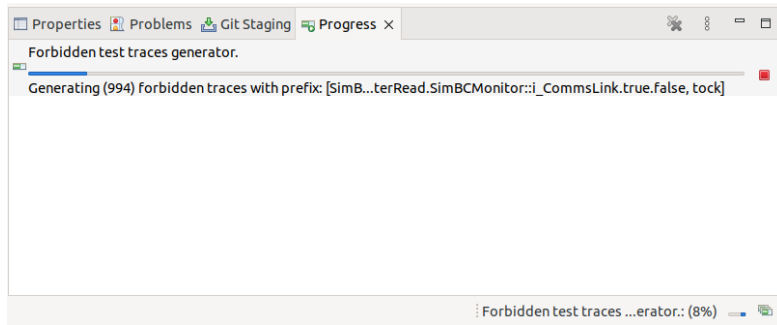


Figure 5.6: Forbidden trace generation progress.

already exists it will be overwritten without confirmation. A file with extension `.rtspec` can be opened within RoboTool and will be parsed, so that it can, for example, be used as input to generate component tests targetting ROS as part of ongoing research work.

Model Checking with PRISM

Additionally, RoboChart is capable of modelling probabilistic behaviour through probabilistic junctions and concrete probability values on transitions. This probabilistic behaviour is not captured in the automatically generated CSP semantics. Instead, we provide another manually invoked probabilistic behaviour generator or translator called `translate2prism`. It allows us to generate the probabilistic behaviour of a RoboChart model in a PRISM specification. Then we could use PRISM, a probabilistic model checker, to analyse the generated PRISM model.

6.1 Invoking the translator

1. Open a RoboChart project;
2. In *Model Explorer*, select the `rct` file that contains the top level RoboChart module; and
3. Right click the selected `rct` file and click `RoboTool >> PRISM >> Compile`;
4. The first time you run the translator, a `prism-gen` folder will be created.
5. Upon a successful translation, a popup dialog with the “Translate to PRISM Successfully” message will appear and then click `OK`. There will be several files generated in the `prism-gen` folder.
 - a) `rc2prism.prefs`: the configuration file for the translator;
 - b) `<rct-file-name>.prism`: the generated PRISM specification that we would like to analyse;
 - c) `<rct-file-name>.pm` and `<rct-file-name>.psc`: the intermediate files during the translation and could be deleted safely (but if the translation fails, they could provide additional information for debugging);

6. However, if an error occurs during the translation, a popup dialog with an error message (shown in Figure 6.1) will appear. This is mainly because of following reasons:
- a) Ex01: this feature is not yet supported in current release of the translator and will be supported in the future.
 - b) Ex02: this feature cannot be supported in the translator.
 - c) Ex03: the well-formedness condition is violated.
 - d) Ex09: miscellaneous causes as indicated in the error messages.

Note: these errors occur in the translation from RoboChart to PRISM and prevent the RoboChart model from being analysed by PRISM for its probabilistic behaviour, but they will not prevent it from being analysed by FDR. Therefore, you still can use FDR to verify other aspects of properties of the model.

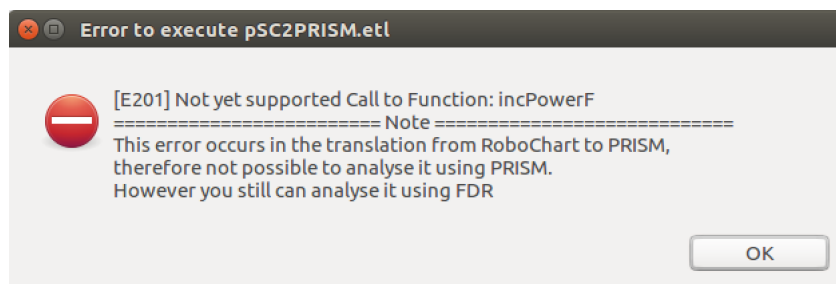


Figure 6.1: Translation Error Dialog

6.2 Configuration File

The translator has a configuration file (named `rc2prism.prefs`) that allows users to provide customised inputs to the translator. One example is illustrated below.

```
#Configuration file for the translator from RoboChart to PRISM
#Fri Mar 08 14:34:29 GMT 2019
NAT_MIN=0
INT_MIN=-10
```

```
NAT_MAX=10
USE_CONF_FROM_INSTANTIATION_CSP=true
PRISM_MODELTYPE=dtmc
DBG_OPTION=true
ERROR_LOG_FILE=rc2prism.log
OUTPUT_LOG_FILE=rc2prism.log
INT_MAX=10
TYPE_OVERRIDE=[STM]left_right.steps \: [0..MAX],\
    [STM]left_right.x \: [1..M]
MAX_SEQ_INS=30
```

Detailed description of each configuration item is given below.

1. INT_MIN and INT_MAX: minimum and maximum integer numbers;
2. NAT_MIN and NAT_MAX: minimum and maximum natural numbers;
3. USE_CONF_FROM_INSTANTIATION_CSP: if it is set to true, minimum and maximum integer and natural numbers will come from the instantiations file (`instantiations.csp`) that is in the `csp-gen` folder. Otherwise, minimum and maximum integer and natural numbers will use above configuration items given in this configuration file;
4. PRISM_MODELTYPE: either `dtmc` or `mdp` for the discrete-time Markov chains or Markov decision processes respectively;
5. ERROR_LOG_FILE and OUTPUT_LOG_FILE: debug and error log file names. If their values are empty, by default debug and error messages will be displayed in standard output (`stdout`) and error (`stderr`).
6. DBG_OPTION: if it is set to true, debug information will be printed in the console or the specified log file for diagnostic purpose.
7. TYPE_OVERRIDE: this allows users to provide customised PRISM type for individual variables in the RoboChart model due to the fact that subtypes (such as `[0..3]`) are not supported in RoboChart now.
 - `[STM]left_right.steps: [0..MAX]`: for the variable `steps` in the state machine `left_right`, its type is set to the range from 0 to MAX in PRISM.

- Apart from [STM], the translator also supports additional two tags: [RP] and [CTRL] for robotic platforms and controllers respectively.
8. MAX_SEQ_INS: the maximum size of sequences when they are treated as fixed size arrays.

6.3 Opening PRISM directly from Eclipse

In order to load the generated PRISM specification `<rct-file-name>.prism` in PRISM, we can either open the executable PRISM model checker `xprism` then load the specification file, or open the PRISM model checker directly from Eclipse. Opening PRISM directly from Eclipse is very similar to opening FDR directly from Eclipse. Right-click the `.prism` file, select `Open with` `System Editor`. If `xprism` is not the default program for `.prism` files, you can add it to the list of editors in Eclipse as follows:

1. Right-click a `.prism` file and select `Open with` `Other`
2. Select `External programs`, click `Browse...`, find the `xprism` executable and click `OK`
3. Click `OK`

Once you follow these steps, every time you double click a `.prism` file in Eclipse, PRISM should open. If you want to open the `.prism` file in a text editor, right-click the file, and select `Open with` `Text Editor`. To open the file in PRISM again, right-click the file, and select `Open with` `xprism`.

6.4 Probabilistic assertion language

The syntax of the probabilistic assertion language is documented in RoboChart reference manual. In order to invoke verification with PRISM through the probabilistic assertion language, please follow the steps below:

1. Open a RoboChart project;
2. Create an assertion file and its extension is `assertions`;
3. Write the properties to be verified in the assertion file according to its syntax;

4. (Optional) If the RoboChart model has not generated its PRISM model manually, right-click one `rct` file, and select the `RoboTool >> PRISM >> Compile` item to generate the PRISM model.
5. Right click the selected `assertions` file and click `RoboTool >> PRISM >> Run` to verify the properties;
6. Provided there are no errors in the assertions or models, PRISM checks the assertions in the background, and RoboTool summarises the result in the form of a report, which is automatically opened upon completion of the checks.

Mutation Testing with Wodel

The RoboTool plugins can be used to generate traces that a RoboChart model cannot perform; they can be used as the basis for defining tests. An overview of the workflow for generating these traces is shown in Figure 7.1. This approach has been developed using the mutation tool Wodel, which should be installed following the instructions on its website (<http://gomezabajo.github.io/Wodel/>). It may be possible to use this approach with other mutation testing tools. We refer to the paper *Testing Robots Using CSP*¹ for more details of our approach.

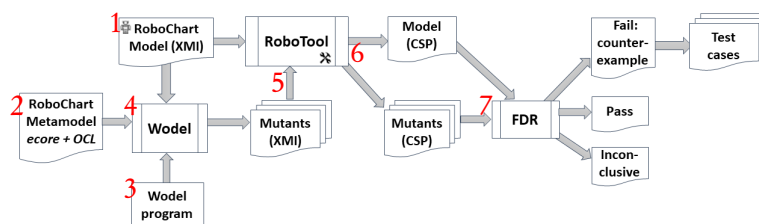


Figure 7.1: Overview of the mutation testing approach with Wodel

The RoboTool features used in this procedure for generating traces are RoboChart Model Importer, RoboChart Model Exporter, RoboChart CSP Generator, and RoboChart CSP Semantics Comparison (along with their dependencies), which should be installed from the RoboTool update site (<https://www.cs.york.ac.uk/robostar/robotool/update/>). This procedure also requires FDR to be installed.

The steps in Figure 7.1 are numbered, and correspond to the sections below. The first three steps (Sections 7.1 to 7.3) are concerned with preparing the inputs required by Wodel, the fourth step (Section 7.4) is the generation of mutants with Wodel, the fifth and sixth steps (Sections 7.5 and 7.6) concern the import of mutants into RoboTool and generating CSP from

¹Cavalcanti A., Baxter J., Hierons R.M., Lefticaru R. (2019) Testing Robots Using CSP. In: Beyer D., Keller C. (eds) Tests and Proofs. TAP 2019. Lecture Notes in Computer Science, vol 11823. Springer, Cham. https://doi.org/10.1007/978-3-030-31157-5_2

them, and the seventh step (Section 7.7) is the comparison of the mutants with the original model to generate traces.

7.1 Exporting a RoboChart Model to XMI

Wodel takes the model to mutate as an Eclipse Modelling Framework (EMF) model in the XML Metadata Interchange (XMI) format. RoboChart models can be exported in this format by performing the following steps in Eclipse:

1. Select the `File > Export...` menu item, or right-click on the RoboChart file to export and select `Export...`;
2. Select RoboChart and then RoboChart Project as EMF Model;
3. Click `Next`;
4. Select the RoboChart (.rct) files to export in the upper panes;
5. Use the `Browse...` button to select the desired destination folder for the exported XMI files, or type the path of the destination folder into the text field;
6. If the model being exported is composed of multiple .rct files, select the Merge input files into a single file named option and enter the name of the desired output file (which should have the .model extension), since Wodel operates only on a single XMI file at a time;
7. Click `Finish`.

The exported .model files are then placed in the provided destination folder.

7.2 Obtaining the RoboChart Metamodel

Wodel requires an Ecore metamodel describing the layout of the model to be mutated. The current RoboChart metamodel used by RoboTool is included in the files of the RoboChart Metamodel RoboTool feature and can be downloaded directly from:

<https://github.com/robo-star/circus.robocalc.robochart.parent/blob/master/circus.robocalc.robochart/model/robochart.ecore>

7.3 Writing Mutation Operators

A new Wodel project should be created as described at <https://github.com/gomezabajo/Wodel/wiki/Get-Started>:

1. Select the menu item;
2. Select Wodel then New Wodel Project;
3. Click ;
4. Enter the desired project name and mutation file name;
5. Click ;
6. Use the button to select the RoboChart metamodel downloaded in Section 7.2;
7. Click ;
8. In the popup message, click ;
9. Copy the `.model` file exported in Section 7.1 to the `data/model/` folder of the newly created Wodel project.

Wodel generates a `.mutator` file in the `src/` folder of the generated project. This should be modified to include suitable mutation operators for the desired mutations. We provide a standard set of mutation operators at <https://robo-star.cs.york.ac.uk/robotool/testing/RoboChartMutationOperators.mutator>, developed by Raluca Lefticaru for the Solar Panel Vacuum Cleaner case study, which cover a range of different RoboChart elements and should be applicable to most RoboChart models. Further details on how to write mutation operators can be found in <http://www.miso.es/pubs/Wodel.pdf>. Mutation operators should not change the robotic platform, since that changes the interface of the module used for comparison. The mutation operators that are most likely to generate well-formed mutants are usually those that change transitions, states and actions within a state machine.

Examples of mutation operators on RoboChart models:

- Converting an entry action to a during action:

```
1  retype one EntryAction as DuringAction
2
```

- Moving the target of a transition to another state:

```
1  modify target ^target from one Transition to other State
2
```

- Replacing a transition trigger with another event selected from an interface:

```
1  interf = select one Interface where {events <> null}
2  ev = select one Event in interf->events
3  tg = create Trigger with {event = ev}
4  modify one Transition with {trigger = tg}
5
```

- Deleting a state and the transitions from and to it:

```
1  st = select one State
2  remove all Transition where {^source = st}
3  remove all Transition where {^target = st}
4  remove st
5
```

- Removing the action of a transition:

```
1  tr = select one Transition where {action <> null}
2  remove one Call from tr->action
3  remove one SendEvent from tr->action
4  remove one Action from tr
5
```

7.4 Running Wodel

Once a suitable set of mutation operators has been created, they can be run on the model to produce mutants, by right-clicking on the `.mutator` file and selecting `Wodel... >> Execute Mutations`. This creates generated mutants in the mutant output folder (usually `data/out/`).

7.5 Importing Mutants into RoboTool

The mutants generated by Wodel can be imported back into RoboTool by performing the following steps in Eclipse:

1. Select the `File >> Import...` menu item;
2. Select RoboChart, then `RoboChart models into project`;
3. Click `Next`;
4. Use the `Browse...` to select the folder in which Wodel output the mutants, or enter the path to the folder in the text field;
5. Select the mutants you would like to import in the panes below the text field; if they are not visible then adjust or deactivate the `Limit search depth below root to` option;
6. Use the `Browse...` button below the panes to select a project folder to import the mutants into (this is usually the project the original file was exported from);
7. Select the `Preserve directory structure below root` option if you wish to keep the mutants in separate folders below the folder selected in step 4 (this is usually advisable to keep the project organised);
8. Select the `Prefix model element names with directory structure` to ensure unique names and `by default aPrefix filenames with directory structure` options (this is recommended in order to avoid conflicts in generating CSP code and comparing the mutants to the original model);
9. Click `Finish`.

The mutants (within their folders if the option in step 7 was selected) are then imported into the specified place in the project.

7.6 Generating RoboChart CSP Semantics

The CSP semantics for the original model should have been automatically constructed as the model was created. The CSP semantics for the mutants should have been automatically con-

structured when it was imported. If the CSP semantics has not been constructed, construction can be forced by selecting the `RoboTool >> CSP >> Compile` menu option.

If the RoboChart model has any constants, ensure that they have the correct values set for them in `csp-gen/defs/instantiations.csp` and `csp-gen/timed/defs/instantiations.csp`. Only the constants of the original model need to be set, as they are used for all mutants during the mutant comparison.

7.7 Generating Traces from Comparison with Mutants

The comparison can then be carried out by performing the following steps in Eclipse:

1. Right-click on the file containing the RoboChart module of the original model and select `RoboTool >> Compare to mutants...`;
2. Select the mutants to compare to in the upper panes;
3. Select `Group test trace by directories in report` if you wish to have the mutants from different directories listed separately in the report of generated traces (which corresponds to separating different mutation operators in the directory structure generated by Wodel);
4. Select `Remove traces that are duplicates of existing traces` if you wish duplicate traces to be removed from the report of generated traces;
5. Select `List mutants that are refinements of the original in the report` if you wish a list of valid mutants that did not produce counterexample traces to be included in the report of generated traces;
6. Select `List files that are invalid in the report` if you wish a list of files that were not checked due to being invalid or blank to be included in the report of generated traces;
7. Click `Ok`.

Note that checking the mutants against the original model may take some time, especially if the model is large or there are many mutants. If the checking of the mutants completes successfully, an HTML report of the generated traces is produced and opened in Eclipse. Invalid mutants are skipped during the mutant comparison, so traces are not generated for such mutants.

If the checking of mutants does not complete successfully, ensure that FDR is properly licensed by first opening the FDR graphical interface. It should also be ensured that all constants are properly set and within range for their types (type ranges in `instantiations.csp` may need to be checked). The mutation operators in the file provided in Section 7.3 assume that the ranges of numerical types contain the integers 0, 1 and 2, so the ranges in `instantiations.csp` must be defined to contain those ranges. This is already the case for the default `instantiations.csp` file generated by the tool.

If manual execution or customisation of the comparison is desired, the FDR script to perform the comparison is generated as a file named after the file containing the original model (selected in step 1 above) followed by `_compare.csp`, within the `csp-gen/` and `_compare.csp csp-gen/timed/` directories. An FDR script to perform with the comparison for each individual mutant is also generated, named after the file containing the original model joined to the name of the mutant with an underscore and appended with `_compare.csp`. These separate scripts are used during checking to improve memory use and report more detailed results. Note that only the timed version is used during the generation of mutant traces.

APPENDIX A

Deviations from RoboChart reference manual

RoboTool is developed and maintained to be consistent with the RoboChart language definition. Nevertheless, certain aspects of the language definition have not yet been implemented, or have been implemented in a slightly different way, due to limitations in technologies. Here we discuss such deviations.

A.1 Well-formedness condition

Two well-formedness conditions have not been implemented in RoboTool as it requires the use of automated or semi-automated theorem provers. They are:

- J2 The guards of the transitions out of a junction must form a cover; and
- V1 If the initial value of a required variable or constant of a state machine or controller is defined, it must be consistent with the value of any (complementing) variable provided by the contexts (controllers or modules) where the state machine or controller is used.

A.2 Semantics

While the code generation faithfully implements the semantics of RoboChart, it targets the flavour of CSP called CSP-M, which is the input language for FDR. In particular, we make extensive use, in the implementation, of process declarations and FDR compression functions to improve the usability of our semantic models.

Other aspects, such as naming conventions, channel declarations, and so on, which are underspecified in the semantics, are made concrete in the implementation to support the use of FDR with our models. Finally, the concrete syntax of CSP-M varies slightly from the syntax standard CSP, which is used in the reference manual.

Bibliography

- [1] A. Cavalcanti et al. *RoboSim Reference Manual*. Technical report. University of York, Mar. 2019. URL: <https://robostar.cs.york.ac.uk/publications/techreports/reports/robosim-reference.pdf> (cited on page 25).
- [2] A. Miyazawa et al. *RoboChart: a State-Machine Notation for Modelling and Verification of Mobile and Autonomous Robots*. Technical report. York, UK: University of York, Department of Computer Science, 2016. URL: <https://www.cs.york.ac.uk/circus/publications/techreports/reports/robochart-reference.pdf> (cited on page 25).