LTL Checker plug-in 6.1.4

Fabrizio Maria Maggi

Last updated on Wednesday, July 27, 2011.

Introduction

The LTL Checker plug-in allows the user to check a log w .r. t. a given LTL model specifying a number of settings. It is also possible to check a log w. r. t. a Declare model by preliminary translating the Declare model (either generated through the Declare Miner or imported from the file system) into an LTL model using the Declare2LTL plug-in. There are two versions of the plug-in. The first one, the LTL Checker, requires an LTL model as input. The second one, the LTL Checker Default, uses a predefined LTL model and does not require any LTL model as input. The functionality of the plug-in can best be described by an example.

Example use of the plug-in

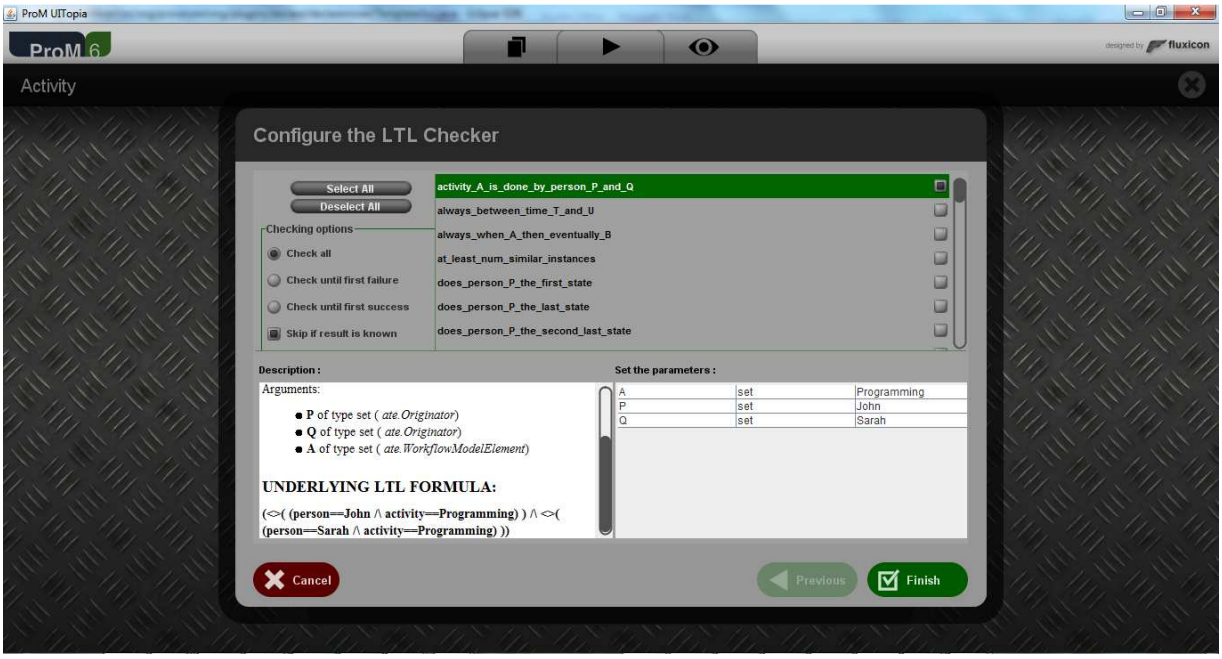
First, you need to start the checker. As the plug-in checks the conformance of a log w. r. t. a given LTL reference model, you first need to load a log.

To use the LTL Checker a reference LTL model is also necessary. It is possible to import an LTL model from the file system or generate it starting from a Declare model using the Declare2LTL plug-in.

The structure of an LTL model must be compliant with the *standard.ltl* file which can be downloaded on http://www.win.tue.nl/~fmaggi/downloads/standard.ltl

The LTL Checker Default does not require the user to specify an LTL model as input. It uses *standard.ltl* as reference LTL model.

After the LTL Checker plug-in has been started, it asks the user for the necessary settings. In particular, the following dialog appears:



In this dialog, you can specify which formulas of the LTL model you want to check.

In particular, selecting a formula in the list on the top-right hand side of the dialog, at the bottom a description of the selected formula and a panel to set the formula parameters (if any) are shown.

Note that, to improve the readability, the LTL formulas are listed using meaningful names (as specified in the corresponding LTL model). The underlying LTL expressions are specified at the end of the description of each formula (see picture above). If a formula is parametric the description changes dynamically according to the values specified for each parameter.

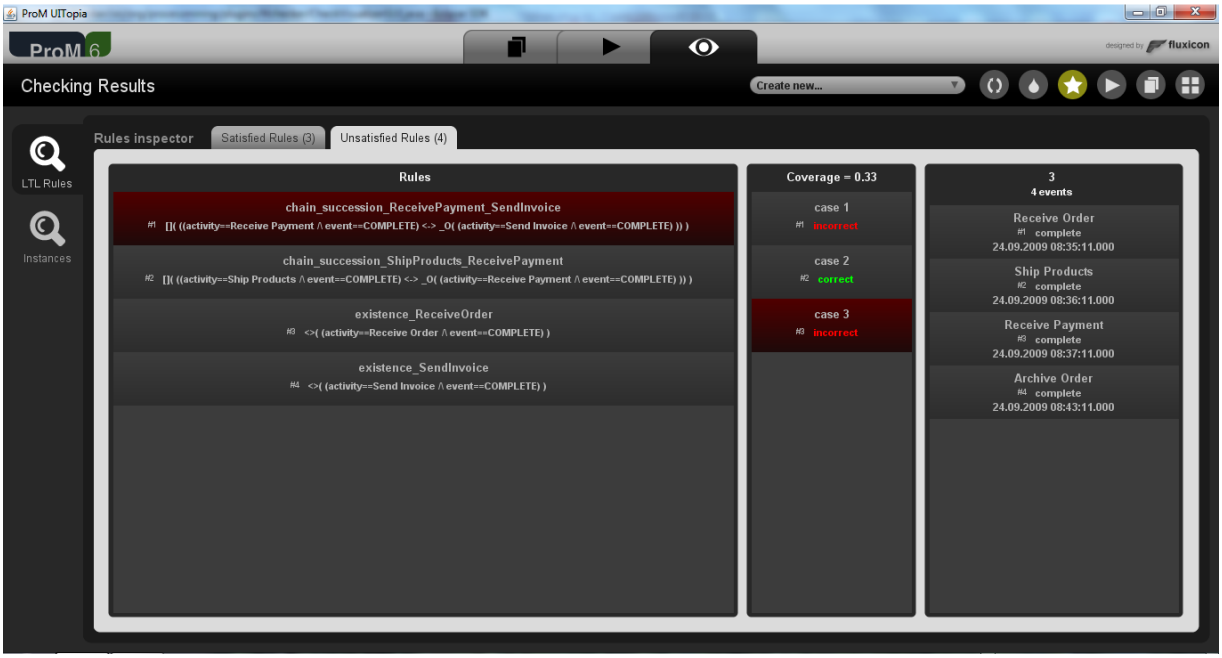
On the top-left hand side of the dialog you can select or deselect all the formulas in the list and also specify further checking options. In particular, you can choose to check all the cases in the log w. r. t. the selected formulas, or check until the first failing pair instance-formula is found, or check until the first successful pair instance-formula is found.

When a case in the log is checked w. r. t. an LTL formula the checking result is stored as an attribute of the case itself. To improve the performance you can check the option *skip if result is known*. The checker will not execute the checking again for the formulas whose checking result is already known.

When you have selected the formulas to be checked, specified the parameters and opportunely tuned the checking options, you can press **Finish**. After that the LTL Checker will start checking the log.

After the checking has finished, a **Checking Results** object will be visualized. The checking results visualization is bidimensional. In fact, you can analyze the results from two different perspectives represented by the *Rules inspector* and the *Instances inspector*.

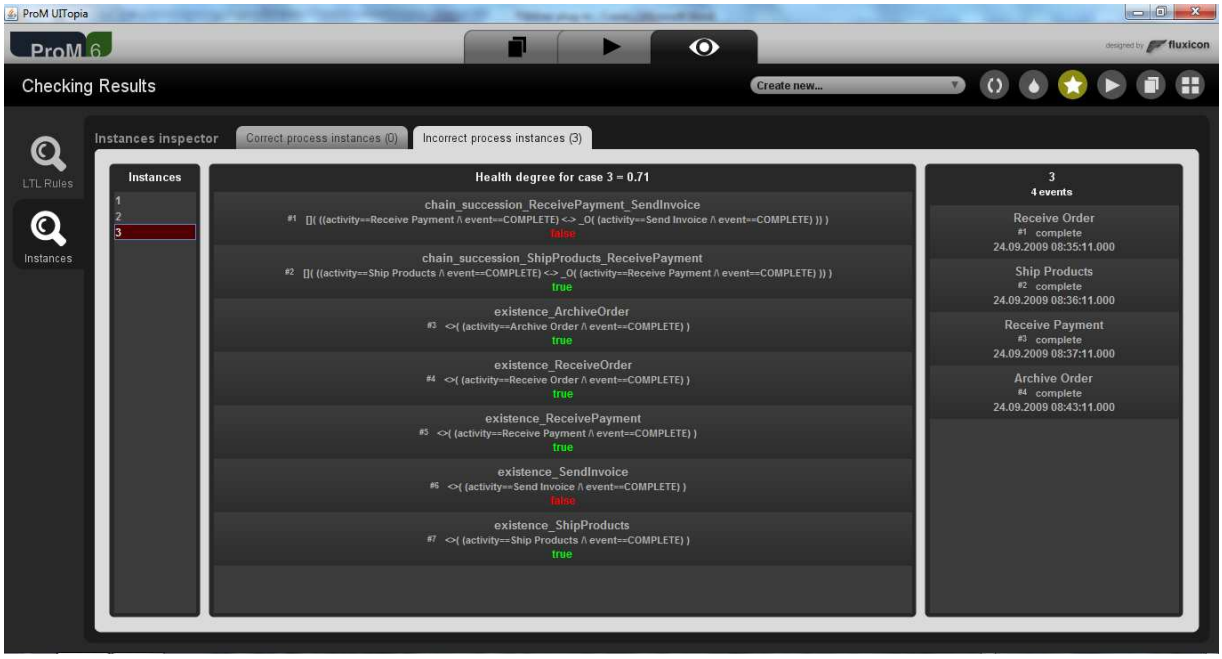
In the following screenshot the **Rules inspector** is shown:



It contains two different tabs. The first tab lists the *satisfied rules*, i.e., the rules satisfied for each case in the log. The second tab lists the *unsatisfied rules*, i.e., the rules for which there exists at least one incorrect case (where the rule is not satisfied).

Each tab of the Rules inspector is composed of three different frames. In the first frame all the checked rules are listed, specifying both the name and the underlying LTL expression. When a rule is selected, it is possible to read in the second frame a list of all the cases classified in correct cases (where the selected rule is satisfied) and incorrect cases (where the selected rule is not satisfied). In this frame the *coverage* of the selected rule is also specified, representing the percentage of correct cases for that specific rule. When a case is selected, in the third frame the sequence of the events of that case is shown. For instance, the above picture shows that the selected chain succession constraint is not satisfied for case 3 because after the Receive Payment activity there is no Send Invoice.

In the following screenshot the **Instances inspector** is shown:



It contains two different tabs. The first tab lists the *correct process instances*, i.e., the cases satisfying each rule of the reference model. The second tab lists the *incorrect process instances*, i.e., the cases for which there exists at least one rule of the model which is not satisfied.

Each tab of the Instances inspector is composed of three different frames. In the first frame the instances are listed represented by their ID. When a case is selected, it is possible to read in the second frame a list of all the checked rules where it is specified whether the selected case is correct w. r. t. each rule or not. In this frame it is also specified the *health degree for the case* representing the percentage of satisfied rules for that specific case. In the third frame the sequence of the events of the selected case is shown.

Plug-in

Classes

The LTL Checker plug-in contains 119 Java classes, which all reside in the **org.processmining.plugins.ltlchecker** package or in sub-packages of it. In the following the role of each sub-package and of the main Java classes is explained:

1. **LTLChecker**.java contains the checking method.
2. **CheckVisualizerGUI**.java holds the GUI visualized to specify the checker settings.

3. **CheckResultsInstanceBrowser**.java, **CheckResultsRuleBrowser**.java, **LTLVerificationResult**.java and **SlickerOpenCheckResults**.java are used to build the GUI to show the checking results.
4. **CheckResultObject**.java holds the results of the checker i.e. the matrix containing the correct/incorrect cases for each rule and the satisfied/unsatisfied rules for each case.
5. **model/LTLModel**.java is the ProM object structured as an LTL model.
6. **formulatree**.* contains the Java classes to map each LTL formula into a tree structure.
7. **importing/LTLModelImportPlugin**.java holds the method to import an LTL model from the file system.
8. **parser**.* contains the Java classes to parse an LTL model.
9. **util/ParamData**.java, **util/ParamTable**.java, **util/Substitutes**.java are used to set the parameters (if any) in an LTL formula.
10. **util/TreeBuilder**.java is the Java class to create a tree structure starting from a parsed LTL model.