Eshmun: Model Repair via Sat Solving

User Manual

Paul C. Attie and Mouhammad Issam Sakr

March 1, 2015

Department of Computer Science
American University of Beirut
Beirut, Lebanon

Contents

1 Introduction

In this manual we explain in detail the user interface of the Eshmun tool, which implements the model repair method of Attie and Sakr [1]. This manual, the Eshmun tool, and the technical report [1] describing the repair method in detail, are all available at `http://eshmuntool.blogspot.com/`.

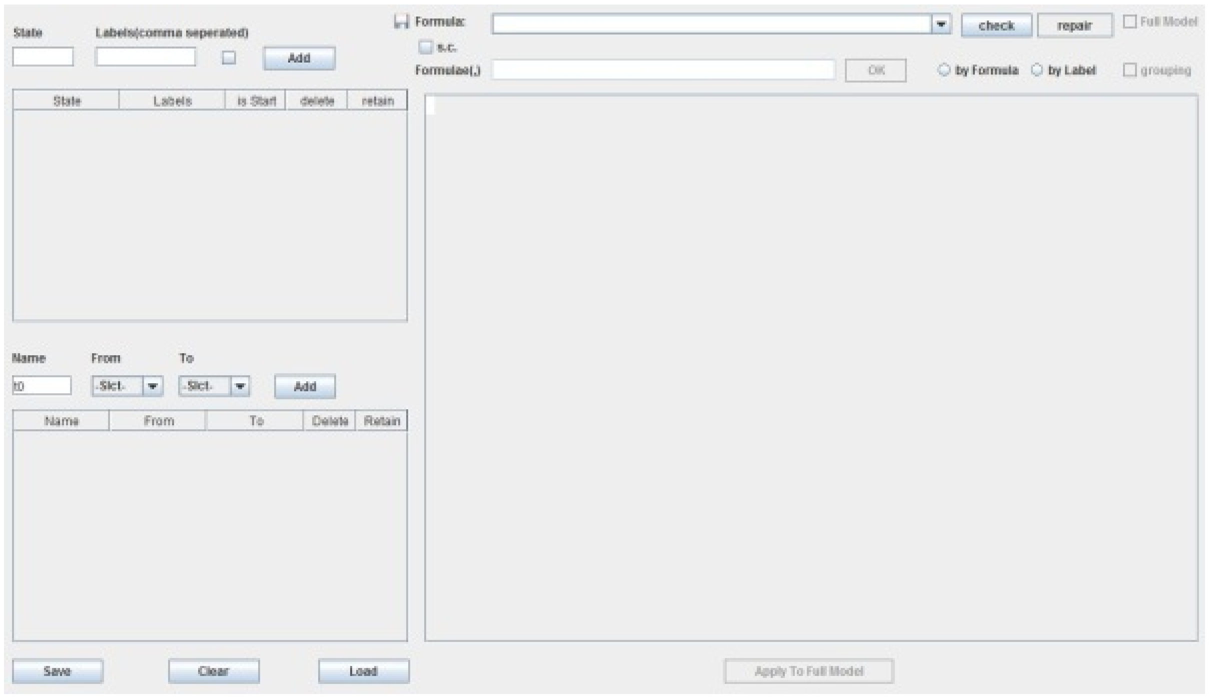


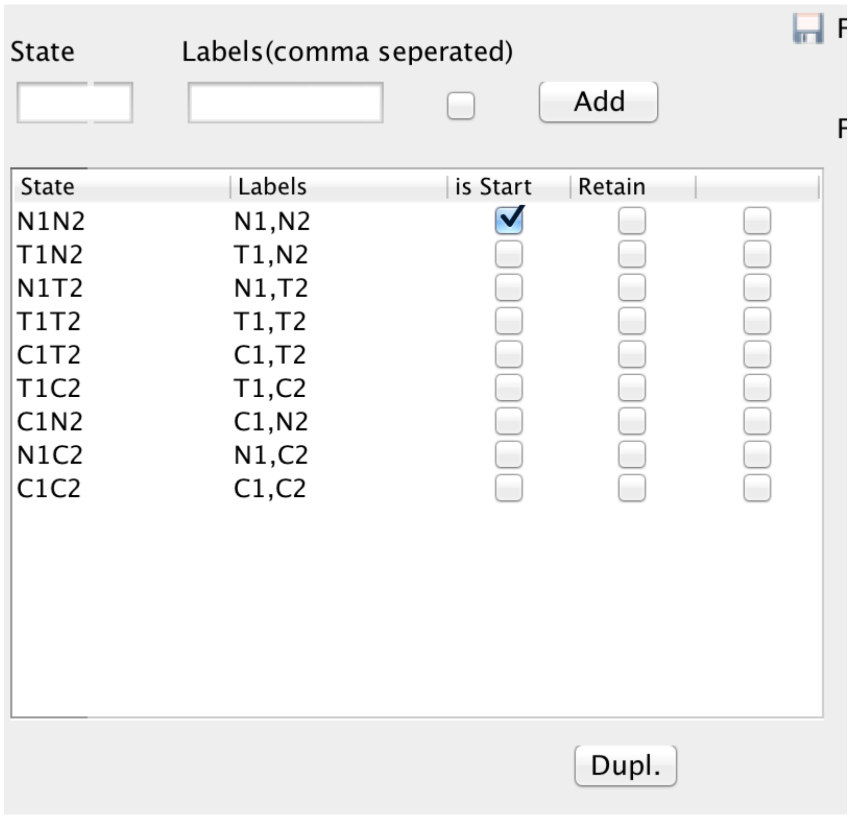
Figure 1: Eshmun main screen

We describe Eshmun's user interface and how to use all its features. This tool allow users to create and manage Kripke structures easily and then model check, and repair them if necessary. The main screen, shown in Fig. 1, is divided into seven sections:

1. State creation
2. Transition creation
3. Kripke structure visualization
4. Kripke structure loading, saving, and clearing
5. Model checking and model repair, and CTL formula syntax
6. Model abstraction, repair of abstract models, and concretization of abstract repairs
7. Repair of hierarchical Kripke structures

2 State management

The section at the top left of the main screen allows users to add, edit, and delete states as well as to manage their label (atomic propositions).



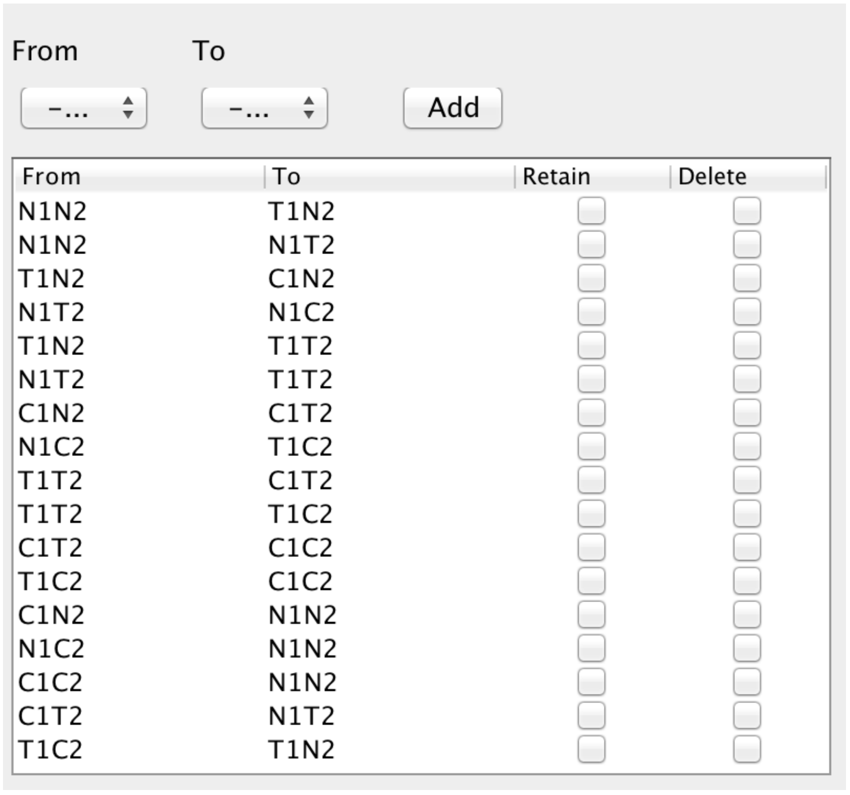
State	Labels	is Start	Retain	
N1N2	N1,N2	✓	☐	☐
T1N2	T1,N2	☐	☐	☐
N1T2	N1,T2	☐	☐	☐
T1T2	T1,T2	☐	☐	☐
C1T2	C1,T2	☐	☐	☐
T1C2	T1,C2	☐	☐	☐
C1N2	C1,N2	☐	☐	☐
N1C2	N1,C2	☐	☐	☐
C1C2	C1,C2	☐	☐	☐

Figure 2: State management for mutex problem

To create a state, the user first enters the state name, the associated labels (propositions that are true in the state) in a comma separated fashion (i.e., $p_1, p_2, p_3, ...p_n$), and specifies whether this state is initial or not. When the user clicks add the state will be displayed in the *states* table as in Fig. 2. All the states in the table are editable, the user can edit states' name, labels, intial flag, and can delete the state by checking the corresponding delete checkbox. The *retain* flag is used in the repair algorithm in order to deny the deletion of the state and to mark it as necessary for the correctness of the repaired model. The name of a state name must be unique, therefore if the user enters a name that already exists a warning message will be displayed.

3 Transition management

The second section in our interface on the bottom left of the main screen is the *transition management* section that allows the management of transitions between states.



From	To	Retain	Delete
N1N2	T1N2	☐	☐
N1N2	N1T2	☐	☐
T1N2	C1N2	☐	☐
N1T2	N1C2	☐	☐
T1N2	T1T2	☐	☐
N1T2	T1T2	☐	☐
C1N2	C1T2	☐	☐
N1C2	T1C2	☐	☐
T1T2	C1T2	☐	☐
T1T2	T1C2	☐	☐
C1T2	C1C2	☐	☐
T1C2	C1C2	☐	☐
C1N2	N1N2	☐	☐
N1C2	N1N2	☐	☐
C1C2	N1N2	☐	☐
C1T2	N1T2	☐	☐
T1C2	T1N2	☐	☐

Figure 3: Transitions screen for mutex problem

To create a transition, the user should select a start state and an end state. Start states and end states are selected from dropdown lists that are bound to the already created states in the previous section. When the user clicks the Add button the newly created transition will be displayed in the transitions' table as in Fig. 3. These Transitions are not editable but the user can delete them by checking the corresponding delete checkbox. The *retain* flag is used in the repair algorithm in order to deny the deletion of the transition and to mark it as necessary for the correctness of the repaired model.

4 Kripke structure visualization

Created Kripke structures are displayed in the bottom right section of the tool's main screen. For example, Fig. 4 shows the Kripke structure for two-process mutual exclusion (see [1] for detailed discussion of this example).

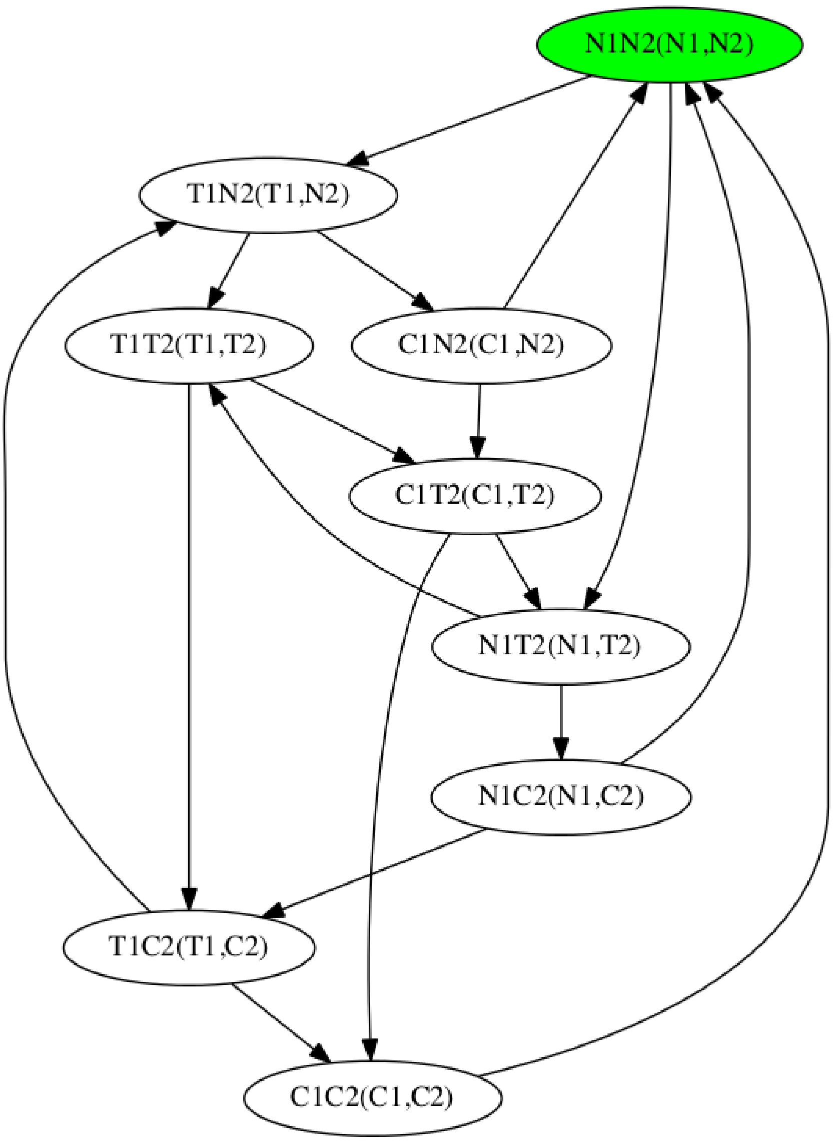


Figure 4: Kripke structure for mutex problem

Whenever the user makes changes to the state table or the transition table these changes will be immediately reflected in the displayed Kripke structure. For instance if the user creates two states `S1` and `S2`, Fig. 5 will be displayed. Green states are initial states. After adding a transition from `S1` to `S2`, Fig. 6 will be displayed.

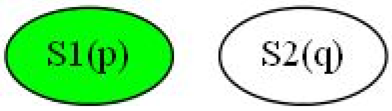


Figure 5: Two created states, `S1` and `S2`, with `S1` initial

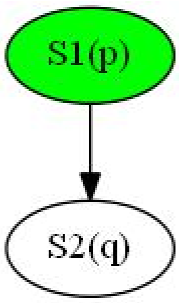


Figure 6: After adding a transition from `S1` to `S2`

5 Kripke structure loading, saving, and clearing

Eshmun allows users to save their work in a text file and then reload it from that file at any time. Users might need to save their work to complete later or to keep several versions of the targeted model. Under the transition table at the bottom left corner of the main screen there are three buttons (see Fig. 7) that allow users to save their model, load a previous model or clear the current model.

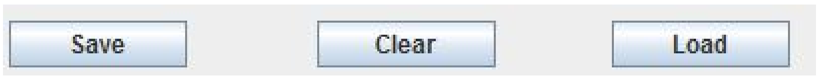
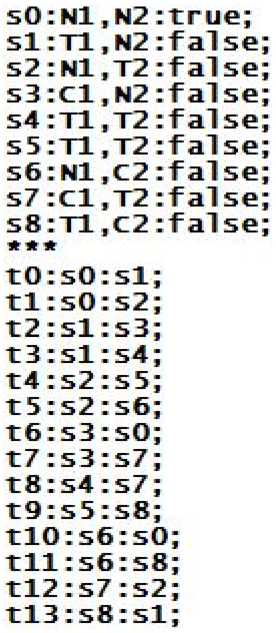


Figure 7: Loading, saving, and clearing Kripke structures

When a user clicks on the Save button a pop up will be displayed to select a file location and a name for this file. If the user clicks on the Load button a pop up will appear to allow user to browse saved files and load the saved models. Clear button will clear all tables in order to create a new model.



```
s0:N1,N2:true;
s1:T1,N2:false;
s2:N1,T2:false;
s3:C1,N2:false;
s4:T1,T2:false;
s5:T1,T2:false;
s6:N1,C2:false;
s7:C1,T2:false;
s8:T1,C2:false;
***
t0:s0:s1;
t1:s0:s2;
t2:s1:s3;
t3:s1:s4;
t4:s2:s5;
t5:s2:s6;
t6:s3:s0;
t7:s3:s7;
t8:s4:s7;
t9:s5:s8;
t10:s6:s0;
t11:s6:s8;
t12:s7:s2;
t13:s8:s1;
```

Figure 8: Saved file for mutex problem

Figure 8 shows the saved file structure. the file has to parts separated by ***. The first part is for states where each line describe a single state as follows: State Name : Labels comma separated : is initial state; . The second part is for transitions where each line describe a single transition as follows: Transition Name : Start State : End State; . (Note that linebreaks are added for clarity of the figure).

6 Model checking and model repair

Once the user completes his Kripke structure M, he can enter a specification η, which is CTL formula, written according to the syntax defined in Sect. 6.1 below. M can be model-checked against η, and repaired if necessary. This is done using the field and buttons show in Fig. 9, which are located at the top of the main screen.

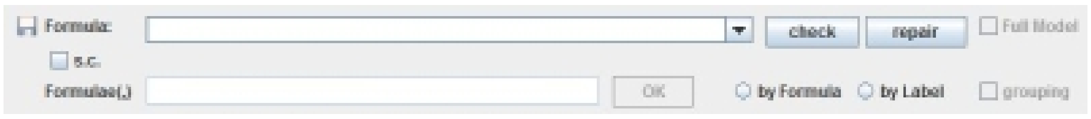


Figure 9: Model check and repair

To model check the designed Kripke structure, the user clicks the check button, which implements the CTL model-checking algorithm of Clarke, Emerson, and Sistla [2]. The result of model checking will be displayed as a text message informing the user whether his model is correct or not. If the result was negative, the user can repair it by clicking on repair button which implements our repair algorithm. If the model can be repaired, the displayed Kripke structure image will be changed by marking transitions to be deleted as dashed edges.

6.1 CTL formula syntax

The CTL formula η is entered into the field at the top of the screen (Fig. 9) with the label Formula. Below is the syntax of CTL formulae that is accepted by Eshmun, where α, β indicate sub-formulae. While some operators can be defined in terms of others (e.g., U in terms of V, & in terms of |), they are provided directly by the Eshmun parser for user convenience, and to make specifications shorter and more readable.

Important: To use this syntax, please click on the s.c. button to the left of the formula entry field at the top. this is a temporary button, for debugging purposes, and will be removed in future releasers.

- ***Not Operator***: $!(\alpha)$
- ***AND Operator***: $\alpha \,\&\, \beta$
- ***OR Operator***: $\alpha \mid \beta$
- ***Implies Operator***: $\alpha \;=>\; \beta$
- ***Equivalence Operator***: $\alpha \;<=>\; \beta$
- ***AU Operator***: $A[(\alpha)U(\beta)]$
- ***EU Operator***: $E[(\alpha)U(\beta)]$

- ***AV Operator***: $A[(\alpha)V(\beta)]$
- ***EV Operator***: $E[(\alpha)V(\beta)]$
- ***AG Operator***: $AG(\alpha)$
- ***EG Operator***: $EG(\alpha)$
- ***AF Operator***: $AF(\alpha)$
- ***EF Operator***: $EF(\alpha)$

The user can choose to save his formulae by clicking the Save button on the top left corner of the repair section. Saved formulas can later on be selected from the CTL dropdownlist to avoid reentering complicated formulas.

7 Model abstraction, repair of abstract models, and concretization of abstract repairs

Eshmun implements two abstraction methods, which can be used to abstract a Kripke structure, which typically results in a smaller abstract model. The abstract model can be repaired with respct to the CTL specification, and the repair then concretized back to the original model. Often this results in a better repair, since the abstract has a "focusing" effect, becasue parts of the structure irrelevant to the specification are typically collapsed by the abstraction.

The by label checkbox implements an equivalence relation with repect to atomic propositions. Once checked a reduced Kripke structure will be displayed and the user can now model check or repair the new Kripke structure. After repairing the reduced model, the result can be projected on the initial model by clicking on the Apply to full Model button.

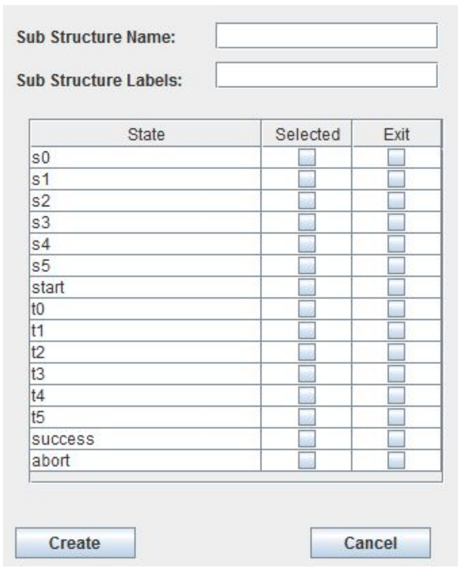
The checkbox by formula can be used in the same way by label checkbox is used but they differ in the reduction algorithm they implement where by formula checkbox implements an equivalence relation with respect to subformulae in the CTL formula. However it needs as input a set of subformulae that are either extracted automatically from the CTL formula or extracted from the textbox where user can enter them in a comma seperated fashion.

The checkbox grouping can be checked once by label or by formulae checkbox are checked. Grouping checkbox tells the tool not to take into consideration states' adjacency when implementing the two previous equivalence relations.

The checkbox Full Model will reload the full model that will replace all reductions. Another important and useful feature in our tool is that whenever the user chooses to reduce his model by using either the by label or by formulae checkbox, in addition to the ability to reduce or repair the reduced model, the states and transitions table on the left of the main screen will display states and transitions of the reduced model allowing users to edit these models, but then there will be no possibility to apply these changes to the initial model as the reduced model has been changed per user choice.

8 Repair of hierarchical Kripke structures

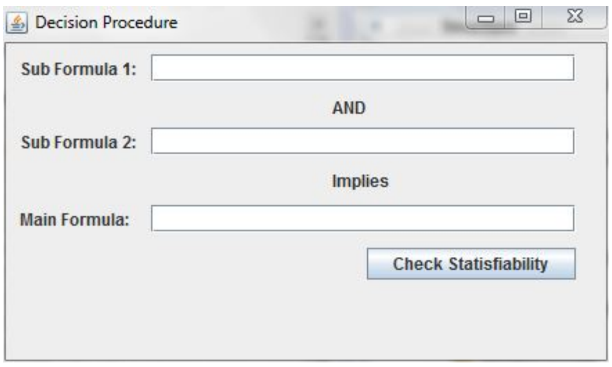
Eshmun allows the user to create substructures either starting from scratch or starting from existing initial model. If the user opt to create these substructures starting from an existing initial model, the tool provides a screen (Fig. 10) that displays all the initial model states and allow the user to select those states that form the substructure, once the states selection is done, the tool manages on its own the transitions (taken from initial model), and generates two new models: the substructure (based on user selection) and another model that is the same as the initial models but all the selected states are now represented as a single box(machine). Moreover, the "Exit" flag is used to allow the user to keep some states in the initial model as there are important transitions going out of them. If the user opts to create the substructures starting from scratch he will need to create them as any other model. To make use of the CTL decision procedure, the tool provides a screen (Fig. 11) that allows the user to enter the three fromulae η_B, φ, and η and displays a message if $\eta_B \wedge \varphi \Rightarrow \eta$ is valid or not. See [1] for a detailed discussion of hierarchical repair.


Sub Structure Name:

Sub Structure Labels:

State	Selected	Exit
s0	☐	☐
s1	☐	☐
s2	☐	☐
s3	☐	☐
s4	☐	☐
s5	☐	☐
start	☐	☐
t0	☐	☐
t1	☐	☐
t2	☐	☐
t3	☐	☐
t4	☐	☐
t5	☐	☐
success	☐	☐
abort	☐	☐

Figure 10: Kripke sub-structure screen


Decision Procedure

Sub Formula 1:

AND

Sub Formula 2:

Implies

Main Formula:

Figure 11: CTL decision procedure screen

References

[1] P. Attie and M. Sakr. Model repair via SAT solving. Technical report, Department of Computer Science, American University of Beirut, Beirut, Lebanon, 2015. available at https://www.dropbox.com/s/32wco7hy585wo9i/repairFull.pdf?dl=0.

[2] E. M. Clarke, E. A. Emerson, and P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *TOPLAS*, 1986.