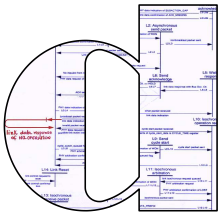
Saarland University
Faculty MI
Computer Science

Verification – WS 2018/2019

Prof. Dr.-Ing. Holger Hermanns
Daniel Stan, PhD
https://dcms.cs.uni-saarland.de/v_1819/



Project Description 1

1 Introduction

As was stated in the first lecture, one task of this course is for you to implement a small model checker on your own. This model checker will use a subset of the Modest modelling language to describe the problems that have to be checked. In the first part, we will implement the CTL-fragment of Modest.

A project template is already supplied for you, it can be found on the materials side of dCMS. This template includes functions to read, parse and represent modest files, and a launcher that stitches all parts of the project together to get an executable program in the end. All the following implementations have to be provided as part of the class `Part1` of the same source file (except Exercise 1.1). You are allowed (but not obliged) to provide additional source files, but cannot modify already existing files of the project (except for `Part1`). Our grading script will instantiate class `Part1` and run each method corresponding to a question with several models and specification and check for the correct answer.

You have to form groups of two to three people. Use the forums if you are still without a partner!

If questions should arise during the project, always feel free to take them to the forums and discus with others. Discussion like this is highly encouraged. However, you must not copy solutions of other teams!

If you think a question does not belong in the forum (for instance if it gives parts of your implementation away), we also like to remind you that we hold office hours each Thursday where we would be happy to help you out.

1.1 Flip Flop

Before you can start implementing the actual project, it is first necessary for you to familiarize yourselves with the input that you will be dealing with during the project. We write the description of our model checking problems in the input language of the Modest model checker. A documentation of the input language can be found in the dCMS materials.

In Figure 1 you can find a circuit representing a flip flop. A flip flop behaviour is defined as follows: "When the input x is turned on (after being off in the previous cycle), in the next step the output y changes its value." Your first task is to get familiar with Modest and build a modest file that has the same behaviour like the circuit.

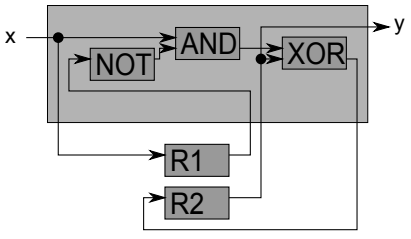


Figure 1: This figure represent a flip flop

We also want to check a CTL property for this circuit: "When x is low, y must not change". Model this property as a CTL property. Add everything to a file `FlipFlop` in the samples directory.

1.2 Bounded Representation

Now that you have a good understanding of modest files, we will start with the actual project. Since the model checking algorithms we use work on transition systems, not on modest directly, we will start of by implementing a function that checks if there is a finite transition system that represents the given modest file. Checking this property is however not possible directly as this problem is generally undecidable. Instead, we will check for a transition system that is size-bounded by some given parameter n. The size of a transition system in our setting is given by the number of states it includes. To this end, implement the `checkBounded(LTS, int)`-method.

2 CTL

Now that you have understood the input format and the data structures that you are presented with in the project, we can finally start do do some model checking! This part of the project will focus on the first model checking algorithm you have seen in the lecture: CTL model checking. By the end of this part, you will have a model checker that will be capable of checking general CTL formulas for finitely representable modest files.

Your main task will be the implementation of branching logic. The implementation for simple propositional logic is already given in the template.

2.1 Detecting CTL

At first we have to check if the given temporal formula is actually a CTL-formula. Remember: In a CTL formula, a temporal quantifier can only occur in the context of a branching quantifier. Since our model checking algorithm cannot deal with more general specifications, we have to reject those that are not supported at this point in the project. Therefore you have to implement the `isCTL(TFormula)` method that returns true or false.

2.2 Model checking – The easy case

The next task is to implement the `satSat(LTS, TFormula)` method that takes as an input a labelled transition system and a temporal formula and returns the satisfaction set[1] of this formula. In this first part, the formula can be assumed to be in ENF. Consequently, your task is to implement the cases for `TFormula` $= \exists\Box\,\Phi$, `TFormula` $= \exists\bigcirc\,\Phi$, and `TFormula` $= \exists\Phi\,\mathsf{U}\,\Psi$.

2.3 Model checking – The general case

Now that your model checker can deal with ENF formulae, you can finally implement general CTL model checking. There are a number of possibilities that can be used to do this, it is up to you to choose what route want to take. However, you have to be aware that some possible solutions might perform better than others. For instance, you should try to keep re-computation of the same satisfaction sets to a minimum.

2.4 Summary

The last task is to implement the `solve(·)` method that brings everything together. It has to call the previous methods and return the solution of the model checking algorithm if possible. Otherwise, an error is returned.

3 Submission

You have to submit you project till 10^{th} of December. Submit your project in teams of two to three students. It is your responsibility to form such groups.

Project submission is handled through the dCMS. Submission will be taken in the form of a zip-file that should contain all source files that are part of your implementation. Only one person from your group should submit the project. The project must include a `.txt` file that indicates the participants in that particular group!

Passing the projects is mandatory to be admitted to the exam! Be sure to test your project thoroughly before submitting your code.

[1] the concrete class implementing |Set<State>| is left to your decision