Trace Analysis Approach/Procedure

This project was developed by creating a set of UPPAAL test models, from which we could extract traces in order to further analyze them. Given the traces from an UPPAAL model, each set of values from them were inspected, so that they could be expressed as a mathematical equation. Following such procedure, the original set of traces were then transformed into a set of equations that resemble their behavior. Once all sets of equations were gathered, we proceeded and then created UPPAL models that represent the behavior of the traces from another UPPAAL model, which we can call for convenience a “recreated model”. Having the recreated models, we then executed the same simulation query in order to extract the same number of traces as the original model and under the same conditions. With the aim to compare the traces against each other and conclude whether the recreated model has the same behavior as the original model.

Objective

The purpose of this project is to accomplish the recreation of UPPAAL models, by only analyzing their traces. With the aim of assuring whether a recreated model can actually represent the same behavior as its original model.

Scope

As discussed in the background query language section. It is possible to retrieve traces from any component of an UPPAAL model. But we only focus on obtaining traces from declared variables of a template. The types of variables used in this project were mostly integers and double values. And each variable’s data or trace information was evaluated in order to represent its behavior as either a polynomial, exponential or harmonic oscillation function (cosine function to be precise). The details of the previous mentioned analysis will be discussed on the next chapters.

Methodology

The way that the traces from a model are analyzed is by the following methodology. We first filter the information of the obtained traces and only consider the numerical variables data for further analysis. We then implement a structure, which we will be calling “buffer” in order to store and analyze the desired information from the traces. After the buffer is initialized, three types of analysis are performed: polynomial analysis, exponential analysis and harmonic oscillation analysis. Each of the previous analyses consist of observing each data point value of the trace’s variables and then fit the information’s behavior to a mathematical expression. The supported functions to be fitted are: polynomial functions of degree n, exponential functions or cosine functions. Only the best suiting functions will be stored in a list of final analysis, which will help in order to construct the final recreated model. A set of algorithms were created in order to accomplish the previously discussed methodology, which, in terms of convenience will be explained in 3 different phases: trace phase, buffer phase and data point phase.

Add that the outcome is a probabilistic model.

Trace Phase algorithm

The initial phase begins by actually having a set of traces from a model that we want to recreate. The set of traces are given by the model generator (explained on the background section), which first needs to be correctly filtered in order to proceed with our methodology. The extraction of the information from the traces proceeds as the following:

----------- Trace Analyzer algorithm ------------

The set of traces are stored in an array of Traces, where we only analyze the information of the traces that come from numerical variables as we can see in line 6. This is how we ensure to only process the information of traces that can only be either Integer or Double variables (at least from what we can analyze from the model generator). We then proceed and initialize our buffer and analyze the trace.

Buffer

The necessity of a structure that holds information from a trace arises from the desire of having control of the amount of information to be analyzed and of how frequently it should be analyzed. We will be referring to the term buffer in this project as the structure that holds the desired fragment of information of a trace to be analyzed.

Let us consider a trace as the following: {0=0, 41=1, 42=2, 43=3, 44=4, 45=5, 46=6}

We can clearly see that the trace has a size of *n* number of data points with their respective *x* and *y* values (as discussed on the background). A buffer is created by first choosing a size <= *n* which will represent the quantity of information to be analyzed. Then a time step value is set, which represents how the information of the original trace will be traversed and analyzed based on the size of the pre-established buffer. This last parameter allows us to treat the buffer as a sliding window that will be shifting the data points of a trace from the beginning until the end.

----------- Buffer initialization algorithm ------------

Explain how the first value of the buffer is the exact first value of the trace, proceeded by a flooring calculation of the next data point. And how the buffer internally “slides”. Also say that the structure of the buffer represents the structure of the trace itself.

Data point Phase algorithm

At this point, we already have a proper buffer that will help to traverse the information of our trace. The information stored on the buffer will be the one to be fitted to one of the three types of functions. Having the data points to be stored on the buffer, allows us to perform several analyzes with the same information in order to choose which analyses suits best. Which is why once every analyses were completed, we compare the range of error that each and one of them had, in order to keep the best suitable one. We will discuss the details of the analyses starting from the next chapter. This phase is only focused on controlling how the information will be analyzed and how the best analysis is chosen for each fragment of the trace.

The organization of the buffer’s information and orchestration of the methods can be well described by the following pseudo-algorithm:

----------- Fitter algorithm ------------

At the beginning of the code from line 3 to 5, the initial errors for the analyses are set to infinity, as none of them were executed at that moment. Then the *x* and *y* data points from the buffer are extracted and stored on an observation list called: “observedData” (line 9). Which will eventually be passed to the 3 types of fitting methods from line 11 – 13. Once each fitting algorithm is finished, the error is then stored on their respective variables for the purpose of identifying the one with the least error and then eventually store the best suitable function that describes the behavior of the observed data. After the analyzes were executed, we proceed and then rearrange the information of the buffer by sliding the buffer window 1 timeStep removing the last data Point and inserting the next value from our trace to observe ((line 16 - 19). Afterwards, we ensure that the sliding window doesn’t surpass the amount of information that actually exists in the original trace (line 19-20) and then repeat the whole procedure again.