Network Analysis Engine:

Introduction:

The Network Analysis Engine (NAE) is intended as a tool for the analysis and synthesis of biological systems, to be used as a replacement for the existing tool, RE:IN. Its long-term goals are to be able to speed up the verification process, perhaps enabling the verification of larger networks, and to offer additional functionality.

Running NAE:

NOTE: NAE is currently in Beta, not yet recommended for end users.

NAE is a java program that can be run using the java command. It takes three mandatory command line arguments, the mode to run in, the name of the model file, and the name of the observation file, in that order. The mode corresponds to various encodings which will be described later. The model and observation files must be in the same format as they should be for RE:IN, with 2 differences. Identifiers cannot begin with a number or be one of the NuSMV keywords/ For example, ‘X’ would not be a valid name for an identifier, as it is the LTL next operator. The program with then output a chart of possible solutions, similarly to RE:IN’s run analysis.

Modes:

In an ongoing effort to speed up the verification process, many different methods of encoding and verifying the problem were researched. NAE supports many different modes, with the hope that as research goes on, two or three modes with be identified as superior and the rest will no longer be supported.

Before enumerating the different modes, it is useful to discuss what they (mostly) have in common.

The Abstract Boolean network was modeled as follows:

1. Each node had a NuSMV module named for it. This model had a Boolean variable called value, which represents the activation of the node. For each node that provides an input to this node, whether it be activation or repression, there is a module parameter named for that node whose input is assumed to be an instantiation of that node’s module. Value variables can be in the VAR clause or FROZENVAR clause, depending on if they can take on a value after the initial state in the mode defined.
2. The activation function is represented as an enum variable called transition, whose possible values are the numbers of the transition functions this node as allowed to take, as defined in the RE:IN user manual. These are FROZENVARs.
3. For each optional connection, there is a Boolean variable named <node name>\_isConnected. These are FROZENVARs.
4. In the ASSIGN block, the next value is assigned based on a case statement based on the value of transition, as well as the values of what nodes are connected.
5. The main module instantiates a module for all nodes, making sure their formal parameters are passed the correct values.
6. The main module in the DEFINE block creates a macro for all observation macros found in the observation file.
7. A specification is provided to find which values satisfy all experiments.

The individual modes mostly differ in four key aspects:

1. Whether they are intended to be used with a BDD or Sat based model checking, and as a result, whether they can use CTL or are restricted to LTL.
2. How they ensure that the solutions are a single set of transitions and connections that verify multiple experiments often with contradictory initial values.
3. How subsequent states are calculated.
4. How the different possible transitions are encoded, whether with a case statement, or pure Boolean logic.

The modes are:

1. CTL: This is perhaps the most straightforward, intuitive method. Since CTL is able to quantify different branches, we are able to in the initial state pick a set of connections and transitions, and in the second state pick a set of values. Then, in CTL, we can find an initial state that for which there exists for each experiment a next state that satisfies it. This mode, since it is in CTL, must use the BDD. Since value variables are assigned a garbage value in the initial state, then their real value in the second state, they must be in the VAR clause.
2. LTLR: Since we cannot look at branches in LTL, if we want to use BMC we must find an alternative encoding. One approach was to in the initial state pick a set of connections and transitions, and then in the next state pick a set of values. Then, after the duration of the first experiment, the model would allow a new set of values to be picked, etc. Essentially, the experiments are encoded as taking place on after the other, where the values of the network are allowed to reset in between experiments. Accordingly, the value variables cannot be FROZENVARs. Thus, a single set of connections and transitions could be asserted to solve all experiments. BDD or BMC can be used, although BMC was much faster and in large cases BDD didn’t went hours without terminating.
3. LTLP: In the vein of the previous mode, this was an attempt to quantify multiple experiments with a single set of transitions and connections using LTL. For each experiment, a redundant value variable was created. Thus, each experiment could be verified in parallel. BDD or BMC can be used, although BMC was much faster and in large cases BDD didn’t went hours without terminating. Value’s can be frozen.
4. LTLRF\LTLPF: These are slight variations on LTLR\LTLP, where as an attempted speedup, subsequent states are calculated differently. Based on the observation that in synchronous models, after defining an initial state, the behavior is deterministic, instead of calculating the next value by using next(value) in the ASSIGN clause, each module puts in its DEFINE clause a macro defining what the next value should be. For example, in a model where experiments last up until five timepoints, each value variable would have four corresponding macros, for what the value of that variable will be at time steps 1-4 (assuming we start at 0). When quantifying the solution, instead of using the LTL X operator, these macros can be used, reducing the bounds of BMC to 0 for LTLP and to the number of experiments for LTLR. This mode also has a few variants, which are currently not available from the command line but must be selected in the source code as slight variation. They are:
   1. One-hot encoding. This represents the transition variable in one-hot format, and instead of using a case block to specify different transitions, does so using pure Boolean logic. This was intended to avoid branching.
   2. DEFINE/VAR. In some versions, the values of subsequent times are macros in the define clause, but these macros get exceedingly large, so as an alternative they can be variables instead, enabling NuSMV to perform substitutions more easily.