A verification tool to check Transducer properties using Z3 SMT Solver encoded in C++ (as a Qt project)

With emphasis on Security related Properties

To Compile and Run CONCERT:

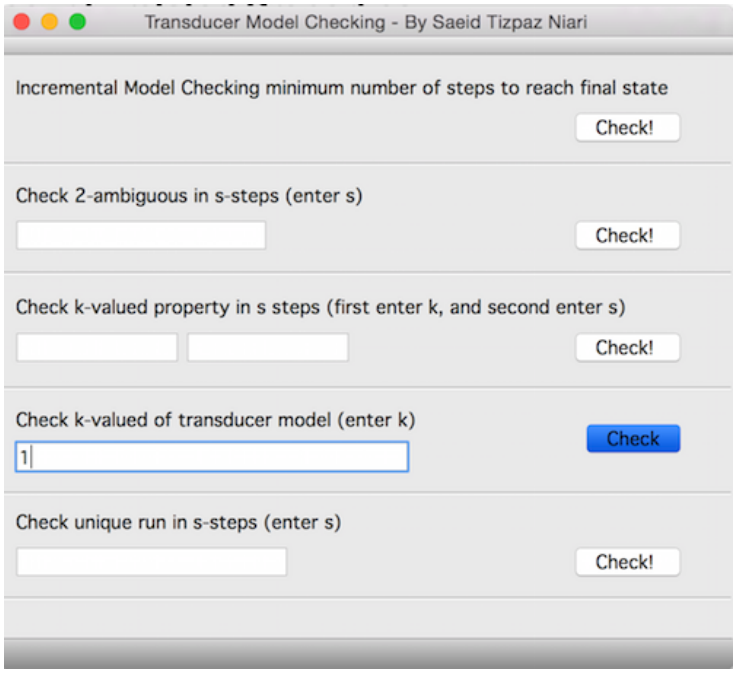
First of all, you need Qt compiler and QtCreator to build this project. Because the project is combined with Z3 library, you need to add libz3.dylib. It has already attached in the project, and you need to set environment variables in your build process: go to Project, then in Build & Run section, choose the run window and find Run Environment in the bottom of the window. Click on Add and set the following variable and value:

DYLD_LIBRARY_PATH = “Your current path in which the project is building”

Then, you should be able to run project without any problem (this is for Mac os and Linux. For windows application, you need to introduce Z3 Library of C++ into the project).

We have one specified transducer inside **Other files** directory of project named **FST.xml** file. This is corresponding to abstract model obtained from a case study. If you want to verify other transducer models, just change **FST.xml** file. We also provide other transducer models inside "Other files" started with **backup-fst...**.

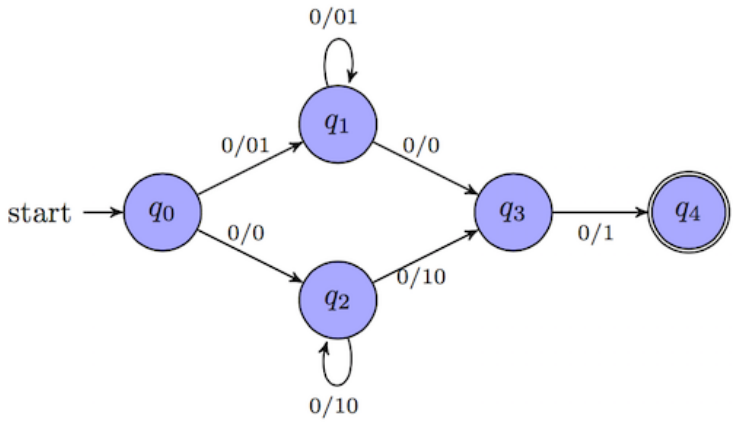
After running program, you will see the window which has Check button. After clicking on Check, the output of the program will be shown in Application Output. This Output includes SMT formulae (definition of constant, variables, functions, and assertions) and the result of evaluation of SMT solver. If the SMT formula is satisfied, it will return the model. In addition, note that the SMT solver may run many times, and it will show its results each time.



Please refer to following reports to understand transducer and its important properties (not required but recommended):
1- Transducer-Confidentiality
2- Transducer-Confidentiality-Slides

Supported Properties

In this part, we want to overview properties supported by CONCERT. Please note that the default transducer model specified in **FST.xml** is as follow:



In this model, the input and output of each transition are separated by **/**. For example, **0000** is an input which is a successful run and produces **010101** either it takes **q1** or **q2** to reach final state. Note that the state is specified using **s0** ... **sn** inside **.xml** file. In the following, we are going to give a high-level explanation of implemented properties.

1. Minimum number of steps to reach final state:

This property shows the minimum number of steps to reach the final state. For example, in case of **FST.xml**, **3** is minimum number of steps to reach final state.

2. Check 2-ambiguity:

2-Ambiguity is satisfied if and only if there are two distinct sequences of states such that both of the sequences are successful runs, and they have the same input. Considering the example of **FST.xml**, **q0-q1-q3-q4** and **q0-q2-q3-q4** are two distinct sequences of states such that both of them are successful runs and **000** is the same input for both of them. As a result, the transducer of **FST.xml** is 2-ambiguity. This property is checked under the bounded model of the

transducer, and the user should specify the number of steps (the minimum steps can be useful here, but the user can specify an arbitrary number).

3. k-valued in s-steps:

Informally, a transducer is k-valued if and only if there are k different successful runs such that all of them have the same inputs, but they produce k different outputs. The necessary conditions of transduers to be k-valued is that the transducer should be k-ambiguity where k >= 1. 1-valued transducer is functional: if a transducer does not satisfy 2-ambiguity conditions, we can conclude that the transducer is 1-valued (functional). Otherwise, we need to check this property to figure out the exact k for a given transducer. Considering the example of **FST.xml**, the transducer is 2-ambiguity, but it is functional (not 2-valued): Consider two successful runs with the same input **0000**, the transucer produces the same outputs: **010101**. If the transition of q0 to q2 would be **0/1**, then the two runs will produce different outputs namely **010101** and **110101**, and the transducer would be 2-valued. This is also bounded version of checking k-valued property. The user needs to specify k and s for k-valued in s-steps respectivelly.

4. k-valued

This is unbounded k-valued model checking. The user does not need to specify any limitation for the number of steps. For formal and technical explanation, please refer to following reports:
1- CONfidentiality CERTifier
2- Verification of Transducer Functionality using Presburger Arthmetic
Please note that SMT Solver may execute many times. If a set of SMT formulae satisfies, it will return model. The model is a counter-example of k-valued property. If SMT formulae are not satisfied, we say that the transducer model is k-valued. In addition, the implementation is checking 1-valued (functionality) now, and we will extend implementations for k > 2.

5. Unique Run

A program satisfies unique-run property if and only if for every run, there is no any other run which can be equivalent run for it. Equivalence of a run is a set of runs which have the same outputs with the run, but they differ at least in one transition. This property can be considered as a way to enforce noninterference in programs. If a program satisfies the unique-run, it does not hold noninterference, and it has some confidentiality leakage. By abstraction of a program in Transducer, we can check unique-run. Please refer to following report for more information:
1- Program Confidentiality Analysis using CEGAR

Please alos note that the SMT solver may execute many times. In the case of **unique_run** property, if it finds a run and its equivalent, it will build new SMT formulae and repeat the same process. Finally, you will see either “there is no any unique run”, or “there is a unique run”.