Circular assume-guarantee reasoning of synchronous systems using Kind 2 and Z3Py

2110436 Ngo Tien Duc

**Background.** SMT-based model checking (MC) is a technique for verifying the correctness of a system using satisfiability modulo theories (SMT) solvers. SMT solvers are a tool for checking the satisfiability of first-order predicate logic formulas. In SMT-based MC, a system model (described as a state transition system) and a temporal property are encoded into a set of logic formulas; then, an SMT solver is applied to verify whether the system satisfies the property. As a target system becomes large and complex, so do the encoded logic formulas, leading to an increased execution time of the verification.

Kind 2 is an SMT-based MC tool that is designed for Lustre programs. Lustre is a synchronous data-flow programming language that is often used in safety-critical system development. Kind 2 deals with safety properties on a program $P$. The input is an annotated program $\{A\}P\{G\}$ (called triple) where $A$ and $G$ are sets of properties (an assume-guarantee contract). Kind 2 verifies the correctness of the triple; namely, for any input signal always satisfying $A$, the output signal always satisfies $G$. Kind 2 implements standard SMT-based MC methods such as the BMC and k-induction methods. Kind 2 supports compositional verification to improve the efficiency of the MC process based on the node-wise component structure of Lustre programs. In a compositional MC process, Kind 2 considers a set of annotated nodes $\{\{A_1\}N_1\{G_1\}, \ldots, \{A_n\}N_n\{G_n\}\}$ and verifies that each of them is correct. In the MC of parent nodes, the descriptions of child nodes $N_i$ are abstracted with their contracts $A_i$ and $G_i$ to make the process efficient.

Kind 2 has limitations in handling practical Lustre programs when a given property depends on lengthy behaviors or when a node description contains circular referencing of signals. Lengthy behaviors result in large logic formulas in the MC process and the compositional process can be effective to analyze them. Circular referencing of signals occurs when two nodes $N$ and $N'$ depend on each other. For instance, the output of $N$ provides the input for $N'$ and *vice versa*. When verifying a program containing such a circular pair of nodes, the compositional MC function of Kind 2 does not work, resulting in a spurious counterexample.

**Objectives.** This thesis proposes a new compositional MC method for Lustre programs that can handle circular programs. The research consists of the following activities: 1) We study the relationship between Lustre programs and the theory of synchronous modules. Lustre nodes can be translated to

synchronous modules so that the existing compositional reasoning methods can be applied. For instance, this will enable to apply a reasoning rule for circular composite modules. 2) We study the automation of deductive reasoning based on the composition rules. Using the Z3 SMT solver with compositional reasoning rules encoded as universally quantified formulas, it is possible to automate the proposed method. We implement a tool that translates Lustre nodes to synchronous modules and then performs compositional reasoning. 3) We conduct experiments based on examples to evaluate the validity and performance of the proposed method.

**Originality.** Circular cases are only supported partially in Kind 2, so we study how to perform compositional verification more completely. Our method is original because it is different from the compositional verification function of Kind 2, as we regard Lustre node instances as synchronous modules and exploit the dedicated deduction rules for circular reasoning.

**Significance.** Verifying the safety of practical synchronous systems is important to prevent accidents and ensure correct operation, especially in safety-critical systems. For example, Kind 2 and our method can be used to verify a control system for a transport-class aircraft.

Practical synchronous systems are described as a large and complex set of Lustre nodes, possibly involving circular referencing. Our compositional MC method is significant because the compositional approach is an effective way to improve the scalability of verification tools.

**Methodology.** This research contains a survey on SMT solvers, synchronous modules, model checking methods, and the Kind 2 tool.

We design a method whose input is an annotated Lustre program and output is the correctness of the input. The basic steps of the proposed method are: 1) *Translation from Lustre nodes into synchronous modules.* We convert Lustre node instances $NI_i$ and properties $AI_i$ and $GI_i$ into synchronous modules $M(NI_i)$, $M(AI_i)$ and $M(GI_i)$, respectively. The we interpret triples $\{AI_i\}\ NI_i\ \{GI_i\}$ into implementation relations on the modules $M(NI_i)\ ||\ M(AI_i) \models M(GI_i)$. 2) *Automatic construction of a proof tree based on the deduction rules.* We encode the definition of the compositional reasoning rules, the declarations of modules, and the implementation relations among modules into a set of logic formulas. Then, we run an SMT solving process to search for a proof tree. Kind 2 is used to check the correctness of leaf nodes. We discuss the soundness of the method. It is followed from the correctness of the translation between Lustre nodes and synchronous modules and the soundness of the deduction rules.

We have implemented the method as a Python script using Kind 2 and Z3Py. The tool consists of: 1) *A translator module.* Given node instances and annotated properties, the module generates an intermediate representation

of synchronous modules. 2) *A printer module* that prints intermediate data, such as a monolithic form of the input program and separated Lustre node instances. 3) *A validator module* that performs the proof tree search using Z3Py.

**Evaluation.** We conducted an experiment to compare the execution time of a monolithic verification process of Kind 2 and a compositional verification process of the proposed method.

We prepared an example of an integrator delayed by two nested counters. The execution time increased because the number of execution steps required to be analyzed in the verification increased as the values of the two parameters increased. In the experiment, we observed correlations between the parameter values, the number of analyzed steps, and the execution time. Along with parameter values, execution time increased rapidly (up to 16 times for one increase) for the monolithic process, while the compositional process took much less time (up to a 150-fold improvement). The reason is that the number of analyzed steps in the monolithic verification was affected by the delay of the two counters, while compositional verification could verify each node separately without taking into account the delay.

We conducted another experiment to evaluate whether our implementation verifies the Lustre programs correctly and efficiently. We prepared two circular examples that cannot be handled by the compositional verification function of Kind 2: 1) A simple parallel composition of synchronous modules. 2) A more complex example containing two digital filters. We confirmed that our implementation verified the two examples correctly. Besides, the process for the second example was inefficient due to the large search space among many possible deductions in the compositional reasoning.

**Conclusion.** In the research, we studied how to apply deduction rules for circular reasoning so that we can perform compositional verification on Lustre programs with a different approach from the Kind 2 tool.

**Keywords**: SMT-based model checking, Lustre programming language, synchronous systems, assume-guarantee reasoning, compositional verification