Project 22703: SYRF

Deliverable 3.2.3

Use of the Stålmarck Method

---

1 Introduction

This deliverable presents the results achieved within task 3.2.3 during SYRF's third year of operation. At a previous review meeting, it was agreed that this task should be changed to reflect Prover Technology's strong interest in continuing to work in a synchronous setting, rather than extending to cover asynchronous systems as well. Reflecting its strong interest in the verification of synchronous systems, Prover has continued to develop LUCIFER, a tool for verifying LUSTRE programs. Work on the LUCIFER tool has been partially funded by SYRF.

2 LUCIFER: a tool for verifying LUSTRE programs

To use Stålmarck's method to verify LUSTRE programs, one needs a translator from Lustre to propositional logic with arithmetic. The macro structure of the model produced by LUCIFER mirrors exactly the node structure of the LUSTRE source program, even down to the variable names. The model is a formula in propositional logic with integer arithmetic expressing the initialisation and the transition function of the LUSTRE program, which is assumed to include a synchronous observer expressing a safety property [2].

2.1 Proofs by induction

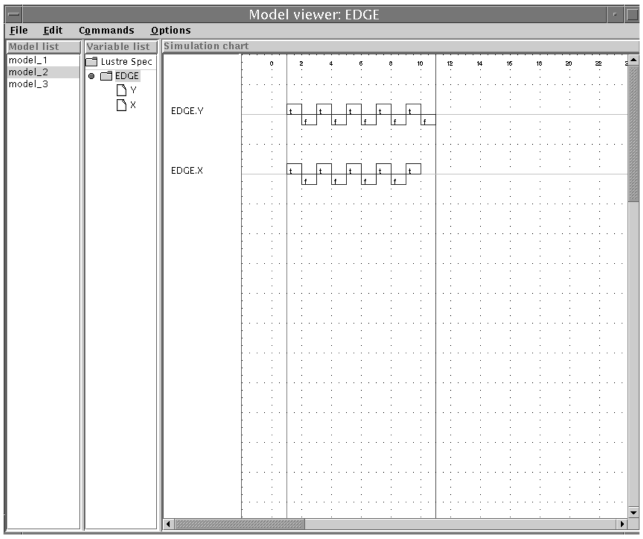
In the simplest case, proofs check that a single output signal (called OK) is always true, and they are by induction. Normally, NP-Tools is used to prove the base case, that the OK signal is true at time zero, and that if it is true at some arbitrary time n, then it is also true at time $n + 1$. That is, we analyse a single time unit in the base case, and a single copy of the transition function in the step. Sometimes though, depending on the shape of the program to be analysed (in terms of its configuration of `pre` operators), it is necessary to consider instead k initial steps and k transitions from an arbitrary state. This proof method we call induction with depth. These kinds of inductive verification proofs are used routinely by Prover Technology in industrial projects. Last year's version of LUCIFER took a positive k as a parameter. However, the resulting verification method is not a complete one. In cases where the transition relation contains reachable as well as unreachable states, the method can be sabotaged by troublesome patterns of states in the unreachable state space. For example, an unreachable state satisfying the property that has a transition both to itself and to a state not satisfying the property will always cause the inductive step to fail, no matter how we turn up the number of transitions considered at a time.

Stålmarck has proposed a simple fix to this problem. We simply demand that all the states considered in the composition of several copies of the transition relation (in the base case and the step) be distinct. Now, a method in which we start with induction depth 1, and increase that depth should the inductive step fail gives a complete method for checking safety properties. Should the base case fail, on the other hand, then we have a countermodel to the desired property; in addition, we know that this countermodel is a shortest possible such. This is of great use during debugging. The fact that the hierarchy of the original Lustre program is preserved in the tranlsation is also helpful when the user views countermodels. This induction-based form of model checking for safety properties has been implemented in LUCIFER. A tutorial on the use of Stålmarck's method written for the Esprit project PROSPER [5] explains the basic ideas behind the verification of safety properties by induction, making use of concrete examples [6]. Some optimisations involving additional satisfiability checks have also been implemented in LUCIFER. They lead to quicker termination in some cases.

Once one begins to consider the use of stronger invariants to strengthen the proof method, then many such invariants spring to mind. We have experimented with several such invariants. For example, one can restrict onself to **locally shortest paths** by demanding that no sequence of composed transitions should appear in an unrolling of the transition relation if it could be replaced by a shorter such sequence linking the same pair of states. This constraint has the advantage that it is naturally expressed in propositional logic, by using (many copies of) the negation of the transition relation. We had high hopes of this constraint, but found that in practice it does no better than our original "all states different" constraint, at least for the examples that we have tried so far. However, it is very important to try to limit the induction depth needed. We cannot afford in large-scale verification to increase the induction depth beyond the diameter of the state machine in question, so we are compelled to find ways to limit the number of iterations necessary. Recently, we believe that we have found the necessary Quantified Boolean Formulas (QBF) to ensure that we never iterate beyond the diameter. In this case, we are really limiting ourselves to **shortest paths**. The resulting formulas are closely related to those that we use in our experiments with more standard fixpoint-based model checking using Stålmarck's method instead of BDDs. A draft paper documents induction-based methods for checking of safety properties [8] To be able to make use of these insights about stronger invariants, we need to translate the QBF formulas to propositional logic, using a method recently developed by Stålmarck [10]. A preliminary version of that method has been implemented as an API (funded in part by the Esprit project PROSPER, # 26241) and a more sophisticated version is now under development.

2.2 Viewing traces of Lustre programs

A new backend has been added to LUCIFER. It generates a transition function in Many Sorted First Order Logic (MSFOL) with finite domains, instead of in the macro language used to represent circuits in NP-Tools. Thus, the translation of a variable x becomes a predicate $x(i)$, where i denotes the discrete time. The translation currently copes only



with a restricted subset of Lustre. In particular, it does not include the `current` and `when` operators.

The MSFOL transition function is passed on to the FAST kernel (a system developed during the Esprit project FAST, # 25581). FAST is a separate project that has developed a method of viewing models (or traces) of specifications in the temporal specification language TRIO. In SYRF, we have demonstrated that the same technology can be used with Lustre, which is a much simpler language. In the FAST kernel the MSFOL formula is passed to `ms2prop`, which translates it to a propositional formula. At the same time one can add a "test objective", a constraint in MSFOL, $x(4) > x(2)$, say, and Prover's satisfiability test is then used for model generation. The models generated are viewed in the FAST Graphical User Interrace (see screen-shot). In the current state of the FAST GUI, one cannot add partial models, like the above test objective, in the GUI directly; that only works if TRIO is used. Instead, one has to add the objective in the MSFOL formula. Clearly, more work needs to be done in this area, but we consider these first experiments to have been successful. We see applications of this technology in test pattern generation.

2.3 Case studies in the use of LUCIFER

The LUCIFER tool has been used in some industrial verification projects at Prover Technology. The tool is currently being evaluated by some potential customers.

At Xilinx Inc., a large US manufacturer of Field Programmable Gate Arrays (FPGAs), Lillieroth and Singh have used Lucifer to verify FPGA cores [3]. Their verification method compares the proposed implementation against a "golden" circuit synthesised (using standard CAD tools) from a specification. Thus, the circuits to be compared are both in the low level format EDIF. This application has driven both the development of LUCIFER and our research in equivalence checking for sequential systems.

3 Journal version of the tutorial on Stålmarck's method

The tutorial about Stålmarck's method that was written during year 2 of the project was selected to appear in a special issue of the journal Formal Methods in System Design, to appear in January 2000. It is very much revised and extended, and is an important documentation of the method, which had not previously been sufficiently exposed in the literature [7]. The tutorial appears as an annex to Deliverable 0.2.3.

4 Reducing hardness of proofs in Stålmarck's method

Work has continued on the addition of extra definitions of fresh variables to system formulas in order to reduce proof hardness. The extra definitions, which we call helper variables increase the set of sub-formulas available to the prover and can reduce branching and so proof complexity. However, this has not been a high-priority activity. The work has been superseded by the development of new proof strategies for propositional logic theorem proving.

References

[1] FAST – Integrating formal approaches to specification, test case generation and automatic design verification: Esprit project number 25581. http://www.prover.com/fast/

[2] N. Halbwachs, F. Lagnier and P. Raymond: Synchronous observers and the verification of reactive systems. In Third Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93, Springer Workshops in Computeting, 1993.

[3] C.J. Lillieroth and S. Singh: Formal Verification of FPGA Cores. Nordic Journal of Computing 6, 1999.

[4] Formal modelling and automatic verification of Lustre programs using NP-Tools, Master's Thesis, Prover Technology and Dept. of Teleinformatics, Royal Institute of Technology (KTH), Sweden, 1999.

[5] PROSPER – Proof and Specification Assisted Design Environments. Esprit project 26241. http://www.dcs.gla.ac.uk/prosper/

[6] M. Sheeran: A tutorial on the use of Stålmarck's proof procedure for propositional logic. Part of Deliverable 4.1, EU Project Prosper [5].

[7] M. Sheeran and G. Stålmarck: A tutorial on Stålmarck's proof procedure for propositional logic. To appear in Formal Methods in System Design, 16:1, January 1999.

[8] Mary Sheeran and Gunnar Stålmarck: Checking safety properties of finite state systems using induction and boolean satisfiability. Draft paper, 1999.

[9] G. Stålmarck: A system for determining propositional logic theorems by applying values and rules to triplets that are generated from a formula, 1989. Swedish Patent No. 467 076 (approved 1992), U.S. Patent No. 5 276 897 (approved 1994), European Patent No. 0403 454 (approved 1995).

[10] Gunnar Stålmarck. Stålmarck's method with extensions to Quantified Boolean Formulas (short abstract of invited talk). In Proceedings of the Eleventh International Conference on Computer Aided Verification, LNCS, Springer Verlag, 1999.