Project 22703: SYRF

Deliverable 3.4.3

Deductive Proofs of Data-flow Programs

---

1 Introduction

This deliverable is intended to present the current year achievements with respect to task 3.4. This task aimed at exploring both the need of theorem proving for synchronous data-flow programs and ways to achieve it. As an example, the chosen data-flow language was LUSTRE and the theorem proving tool was PVS. According to the proposal, we had in mind the following schedule:

- deliverable 3.4.1 (12 months) : Comparative study of data-flow to PVS translations.
- deliverable 3.4.2 (24 months) : Case studies on checking data-flow programs with PVS.
- deliverable 3.4.3 (36 months) : Front-end prototype.

Yet, as explained in deliverable 3.4.2, this schedule was reordered: some quite convincing progress on case studies were already performed in the first year [3], while a new translation method was studied and experimented during the second year. Then this year was devoted to implementing it. This implementation is at section 2.

By the time, the IRISA team working on SIGNAL experimented another theorem proving tool: Coq [7, 2] and proposed to join this task. This implementation and experimentation is described at section 3.

In addition, as a result of developments in work package 3 in year 2 (see deliverable 3.2.2), we could perform some experiments on verifying Lustre programs with the theorem prover NP-Tools during the third year.

2 A PVS Proof Obligation Generator for LUSTRE Programs

As explained in deliverable 3.4.2, the problem of dealing with clocks leads to consider either "absent" elements and a relational semantic as proposed at section 3 or both finite and infinite sequences. As we aimed at staying as near as possible from the source language, we decided to choose the second solution and, fortunately, Kahn semantic [8] combined with linear mu-calculus [10] allowed us to achieve this goal.

Yet, defining PVS strategies to automatically handle the LUSTRE to PVS translation appeared very inefficient. This is why we chose to run these strategies off-line, in the translation tool itself (written in Caml), and to only generate PVS proof obligations, that is to say, what remains to prove, once the unfolding of inference rules has been performed. The tool is further described in the attached paper [5].

Then, this tool appears very efficient: it allows us to handle multiple clocks in a functional style, it gives us the same proof obligations that resulted from the old translation tool, but much faster and it sometimes perform even better, in the sense that it often finds the right number of unfoldings which allows a property to be proved.

Furthermore, it offers the possibility to deal with asynchronous and possibly unbounded Kahn networks, and thus the possibility to model and prove mixed synchronous-asynchronous designs.

3 Co-Inductive Axiomatization of Signal in Coq

A Signal process is a set of equations on signals. Its semantics is the set of streams of values verifying the equations. The domains of signals are extended with a special value $\perp$ which denotes the absence of the signal. We define co-inductively [11] these streams of values and $\perp$. Signal operators are denoted by co-inductive functions on signals. A Signal equation is denoted as a co-inductive relation on signals. The parallel composition is denoted by the logical *and* of the underlying logic and the restriction, which enables to declare local signals, is denoted by an existential quantifier. From these definitions we can derive the clock properties of the Signals operators.

An important hypothesis of the synchronous programming model is that a signal is assumed to have the property of being present (with a value) within a finite deadline (a set of instants). In Signal, this property is translated into the assumption that there only exists a finite number of $\perp$ between two values of a signal. This property about an infinite object is necessarily formalized by a co-inductive predicate but to ensure a *finite* number of $\perp$ we have to mix co-induction with induction.

We also define a bisimulation on signals: two signals are bisimilar if they provide the same values in the same order, that is to say without their $\perp$ they are equal.

This approach is more precisely described in the annexed paper [13].

The above theory has been implemented with the Coq [2, 7] proof assistant using co-inductive types. It has been put into practice by means of a case study, consisting of the implementation of a steam-boiler in Signal and Coq [9].

4 NP-Tools proofs based on LUSTRE

The work in work package 7 identified some abstractions of the plant model which makes verifying the model of a closed loop system a possibility. In the third year of the project we have used the tool Lucifer which translates Lustre programs to the format acceptable by NP-Tools. Using this tool one can perform inductive proofs, with a base ≥ 1 in a convenient manner [12].

As well as translating a Lustre program, Lucifer can "unfold" the transition function for the system over n steps by automatically generating and connecting n copies of the translation of the original model. Thus, a property such as P will hold after step N can be expressed and proved using an induction on the invariance property

$OK = (\mathit{step\_counter} > N) \to P.$

Thus, the system model automatically unfolded in N steps can be used for proving this induction. This approach has been tried on some toy examples.

The work in work package 7 has led to formalisation of the physical environment for the climatic chamber case study, so that a model in Lustre could be developed. Thus, a closed loop model was translated to NP-Tools via Lucifer. For some safety properties directly expressible in terms of the controller model the proofs in NP-Tools were easy (within seconds). Next, we attempted some timeliness properties based on the unfolded model of the closed loop system.

While this approach seems feasible, the actual proof for bounded response property of the climatic chamber was not completed by the time of writing the deliverables. Some of the base steps were performed real fast, whereas the proof never reached the induction step before it was interrupted.

5 Conclusion

At the end of the project, we come up with two approaches: the LUSTRE-PVS and the SIGNAL-Coq one, which seem to operate quite satisfactorily. Unfortunately, we have not been able to make comparisons between them and this remains to be done.

Several other points remain to be studied:

- The SIGNAL-Coq approach experimented the proof of liveness properties; this has also to be done in theLUSTRE-PVS one; this should be easy since nearly the same linear Mu-calculus logic applies to both of them.
- The experiments have dealt with flat programs. This is likely to only apply to small programs and raises the question of addressing large, structured ones.
- One way of answering this last question could be to study some refinement-based design method *à la* B [1], adapted to the synchronous programming context. In this sense, a proof obligation generator like the one described at section 2 could be a good starting point.

As far as inductive proofs in the climatic chamber case: it would seem that some tuning of the prover heuristics or some reordering of the subgoals in the stated properties have to be tried to further continue with this work.

References

[1] J.-R. Abrial. *The B-Book*. Cambridge University Press, 1995.

[2] Bruno Barras and al. The Coq Proof Assistant Reference Manual. Technical report, INRIA, 1996.

[3] S. Bensalem, P. Caspi, C. Dumas, and C. Parent-Vigouroux. A methodology for proving control programs with Lustre and PVS. In *Dependable Computing for Critical Applications, DCCA-7, San Jose*. IEEE Computer Society, January 1999.

[4] A. Benveniste, P. Le Guernic, and Ch. Jacquemot. Synchronous programming with events and relations : the Signal language and its semantics. *Science of Computer Programming*, 16:103–149, 1991.

[5] C. Dumas. A PVS proof obligation generator for Lustre programs. *Unpublished draft*, 1999.

[6] T. Gautier, P. Le Guernic, and F. Dupont. SIGNAL v4 : manuel de référence (version préliminaire). Publication interne 832, IRISA, June 1994.

[7] E. Giménez. *Un Calcul de Constructions Infinies et son Application à la Vérification des Systèmes Communicants*. PhD thesis, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, December 1996.

[8] G. Kahn. The semantics of a simple language for parallel programming. In *IFIP 74 Congress*. North Holland, Amsterdam, 1974.

[9] M. Kerboeuf, D. Nowak, J.-P. Talpin. The steam-boiler problem in SIGNAL-COQ. Research Report No. 3773. INRIA, October 1999.

[10] D. Kozen. Results on the propositional mu-calculus. *Theor.Comp.Sci*, 27:333–354, 1983.

[11] R. Milner and M. Tofte. Co-induction in relational semantics. *Theoretical Computer Science*, 87(1):209–220, 16 September 1991.

[12] S. Nadjm-Tehrani, O. Åkerlund. Combining theorem proving and continuous models in synchronous design. In *Proceedings of the World Congress on Formal Methods*, Toulouse, France, September 1999, Volume II, LNCS 1709, pages 1384-1399, (C)

[13] D. Nowak, J.-R. Beauvais, J.-P. Talpin. Co-Inductive Axiomatization of a Synchronous Language. In *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics*, number 1479 in LNCS, pages 387–399 Springer Verlag, October 1998. http://www.irisa.fr/prive/nowak/publis/tphols98.ps.gz.