Towards a Generic Tool for Reasoning about Labeled Transition Systems

Pierre Castéran and Davy Rouillard

LaBRI, Bordeaux I University, 33405 TALENCE Cedex, FRANCE,
{casteran,rouillar}@labri.u-bordeaux.fr,
WWW home page :
http://dept-info.labri.u-bordeaux.fr/~casteran/CClair/CClair.html
This work is partially supported by the French RNRT project Calife.

Abstract. The *CClair* project is designed to be a general framework for working with various kinds of transition systems, allowing both verification and testing activities. It is structured as a set of theories of *Isabelle/HOL*, the root being a theory of transition systems and their behaviour. Subtheories define particular families of systems, like constrained and timed automata. Besides the great expressivity of higher order logic, we show how important features like rewriting and existential variables are determinant in this kind of framework.

1 Introduction

Labeled transition systems [4, 15] form a widely used formalism for studying properties of reactive softwares : for instance, we can prove invariants, more or less strong equivalences between two systems, or build executions for simulation and test purposes. Besides the plain model — a set of labeled transitions linking some states —, there exists many variations in the litterature : constrained automata [3, 9], input-output automata [19], various kinds of timed automata [1, 18], without forgetting Petri nets, Turing Machines, Robin Milner's CCS [20].

In general, these concepts are used to modelize critical and/or complex systems, the behaviour of which is important and hard to understand. Computer-aided tools are thus necessary to do huge computations or reasoning, allowing to avoid some human but harmful errors.

Depending on which kind of system we want to work on, two main approaches can be considered :

Automatic tools can work on some well characterized families of transition systems which have nice decidability properties; they often cross huge finite automata for checking some invariant, doing simulations, etc. Among these tools, we can cite *MEC* [2], *Hytech* [14], *Kronos* [12]. When the size of some system is too big, or the system is parameterized, or its behaviour is described in terms of complex data structures, automatic tools based on enumeration techniques may fail, because of physical limitations of computers or undecidability results [8].

Proof assistants offer the possibility of working with a great variety of mathematical objects (including infinite ones) and obtaining results with a very good

level of confidence. Their real usability comes from their semi-automatic features : although they are human-driven — the user states all the main reasoning steps —, some tools like tactics, conversions, decision procedures, spare the user the bother of making directly lots of inferences. Some of them allow the user to call "oracles" to delegate some computations to other tools, with a possible loss of confidence. Among well-known proof assistants, let us cite *COQ* [24], *HOL* [21, 31], *Isabelle* [25], *PVS* [30], *Lego* [28], *NuPRL* [13].

As Pnueli points out in his `FM'99` conference [27], both kinds of tools are bound to cooperate within the framework of verification of reactive systems : for instance, we can use a proof assistant to prove that some property P holds for each reachable state of a system S, and use an automatic tool to computation of the set of reachable states of S.

The *CClair* project [10], written in *Isabelle/HOL*, aims to provide the user with an environment for working with transition systems of various kinds, allowing verification and testing in a single framework.

All the techniques presented herein are illustrated by examples we could not include in this communication but are available at the site of the project [11].

2 The theory structure

CClair's kernel is a main theory, called *MainTS*, of plain labeled transition systems, which allows to name states, action labels, transitions, traces and executions, and contains some tools for computing and reasoning. More specific classes of transition systems, like Büchi, constrained or timed automata are defined in terms of their translation into plain transition systems.

2.1 Labeled transition systems

Plain labeled transition systems are defined in a polymorphic way, as in Müller and Nipkow's work on I/O automata [23]. Let α and σ be type variables for action labels and states. The polymorphic type (α, σ) `transition` is used for representing transitions as triples (s, a, s'), and the type (α, σ) `ts` of labeled transition systems is equivalent to (α, σ) `transition set`. Please notice that, since in *HOL*, sets are represented as predicates, no finiteness condition is required transition systems.

Let us denote by "$s \xrightarrow[S]{a} s'$" the relationship "there is a transition in S from s to s' labeled with a". Some derived concepts are defined, mainly by induction or coinduction : the sets of finitely reachable, vivacious states, and executions and traces. Potentially infinite executions and traces are represented using Paulson's lazy lists [26] : traces are lists of action labels and executions are lists of transitions.

Let us denote by "$s \xrightarrow[S]{\xi, w} s'$" the (inductively defined) predicate "ξ is an execution in S from state s to state s' and w is the corresponding trace". Some variants are available, by forgetting the component ξ or w; in the same way,

the coinductively defined predicate "$s \xrightarrow[S]{\xi,w}$" allows us to consider potentially infinite executions and traces, but no state of arrival is considered.

This main theory also defines classical concepts common to all classes of transition systems : invariants, simulation of some system by another, synchronized product and so on.

The concept of abstraction [17, 19, 22] is also defined in this root theory. A system A is an abstraction of some system C if we can give two mappings f and g mapping the states (respectively the action labels) of C into those of A, and verifying the property : $s \xrightarrow[C]{a} s' \Longrightarrow f(s) \xrightarrow[A]{g(a)} f(s')$.

2.2 Specific kinds of transition systems

Plain transition systems are a too vague model for working on real instances. More accurate models can be described in subtheories of *MainTS*. In general, such a model is defined by a translation into plain transition systems or into already defined automata. At the present time, we mainly worked with two categories of automata, described below. Some experiments were made with constrained and Büchi automata too.

Enumerated transition systems The theory *SmallTS* does not introduce any new definition, but is composed of tactics and conversions which work on transition systems presented as finite enumerations of transitions. At present, we assume that action labels and states are defined on finite datatypes.

This representation allows us to program in *ML* various tactics and conversions based on finite graph algorithmic. For instance, we can get automatically the set of reachable states, or all cycle-free executions from one state to another. Please notice that, unlike using external programs, all the results returned by these tools have the status of 100% proven theorems, since the *ML* functions are only used to direct *Isabelle* proofs.

Parameterized timed automata *Parameterized* timed automata ("p-automata" in short) are constrained automata provided with a single, non resettable clock.

A state in a p-automaton is composed of three informations :

- a *location*, also called *control state*,
- a *date*, which is the current value of the clock,
- a *valuation*, *i.e.* a value of any type useful for the modelization; this type is often a cartesian product, which represents tuples of updatable variables.

A p-automaton is composed of the following informations

- a set of locations,
- an initial location,
- an *invariant*, which associates to every location a predicate relating dates and valuations,

– a set of *discrete transitions*, composed of a start location, a destination location, an action label, and a non deterministic way of updating the valuation. For instance, if some automaton works on two variables x and y, the relationship $x' > x + y \wedge y' = y$ is such an updating, increasing only the value of x.

The operational semantics of p-automata is defined by a translation into the model of plain transition systems. To each p-automaton p is associated a transition system s having the two following properties :

– all states of s satisfy the invariant of p,
– the transitions of s are either the discrete transitions of p, or *temporal transitions* in which only the clock is allowed to change; the invariant associated with the current location must be maintained during this interval of time.

The *control graph* of a p-automaton is its abstraction obtained by ignoring all but its action labels and locations.

We must notice some facts about this model :

– it allows us to define infinite, parameterized automata, with any type for variables ; as a consequence, it is impossible to provide any fully automatic tool for computing or reasoning on these automata,
– traces, executions, reachable states of a given automaton need not to be defined again : it is enough to consider the traces, executions, ... , of the translation into the parent theory.

This model was designed for the formalization and proof of algorithms used in telecommunications and is at the heart of the RNRT scientific project *Calife*. It is fully described in [5] and used in [6], then in [29], to prove the correctness of an "Available Bit Rate" (*ABR*) algorithm which controls the cell rate in an ATM network. This proof uses a synchronized product of 3 basic p-automata, which corresponds to a system with 30 locations, 9 updatable real variables and 3 fixed parameters. The reference [29] describes the precise representation of p-automata within *CClair* as well as the correctness proof.

3 Verification and Testing

Tools for verification and testing are mainly defined in the root theory *MainTS*, and are therefore available for any model of transition system. Nevertheless, it is always possible to define some specific tool in the appropriate subtheory.

3.1 Verification techniques

Let us recall that the behaviour of any transition system is described by the relationships "$s \xrightarrow[S]{\xi,w} s'$" and "$s \xrightarrow[S]{\xi,w}$". Then a verification problem has one

of the two following forms, where P is any predicate relating states, executions and traces.

$$s \xrightarrow[S]{\xi,w} s' \Longrightarrow P(s,\xi,w,s') \tag{1}$$

$$s \xrightarrow[S]{\xi,w} \Longrightarrow P(s,\xi,w) \tag{2}$$

This general form allows us to express many instances of verification problems : temporal properties on traces or executions, invariants defined on reachable states ... , with no limitation in theory.

Two main kinds of tools were developed for that purpose :

- Elimination rules for executions, including those automatically generated by *Isabelle/HOL* and rules allowing to cut executions at any place,
- Libraries of rules for the most frequent properties to be checked, including rewriting systems on *LTL* operators.

For more complex verifications, we adapted to our general framework Müller's work on abstraction [22] : abstraction rules allow to transform verification goals on the concrete system C into verification goals on the abstract system A. In the case where the property to be checked is an *LTL* formula, these abstraction rules reduce proof on executions to trivial obligation proofs on transitions.

3.2 Simulation and Testing

Simulation and testing may be considered as complementary activities with respect to verification. Building executions and traces having some given properties can help to understand better the behaviour of often complex systems, and even to win or lose confidence in some conjectured properties. Moreover, the executions we build can be confronted with the behaviour of the industrial final product in order to detect implementation errors.

Simulation goals can be schematized by the two following statements expressed with the help of *Isabelle*'s existential variables (signaled by question marks) :

$$s \xrightarrow[S]{\xi^?,w^?} s'^? \wedge P(s,\xi^?,w^?,s'^?) \tag{3}$$

$$s \xrightarrow[S]{\xi^?,w^?} \wedge P(s,\xi^?,w^?) \tag{4}$$

Like with *Prolog*-like interaction, the theorems we expect to get are instances of these statements. As for verification, we present below some techniques we often used in simulation activity.

- Building some finite execution is basically done using introduction rules on executions. Once some existential variable is — even partially — instantiated, we can try to simplify the proposition $P(\dots)$. This simplification

sometimes helps to detect failures and leads to backtrack in order to find another execution.

- In the *SmallTS* theory (section 2.2), conversions and tactics based on graph crossing compute and prove automatically statements of the form $s \xrightarrow[S]{\xi} s'$; if the user provides only s, s' and S, w and ξ are determined by the tool. Moreover, this tactic is compatible with *Isabelle*'s backtracking, and a proof of a statement like (3) takes the form of a communication between the generation of executions and the simplification of the $P(\dots)$ part, which may result in partial failures. For instance, let a and b be actions symbols, s and s' be states of some finite system S; then a query like "$s \xrightarrow[S]{\xi^?, w^?} s' \wedge w^? \models \Diamond(a \wedge \bigcirc\Diamond b)$" which looks for some execution from s to s' such that its trace contains some occurrence of a before some occurrence of b, is solved in one step, thanks to *Isabelle*'s *Prolog*-like facilities.
- Building infinite executions is obviously more complex. If we do not want to do it manually, the best is to use operators like $\bullet^\Omega$ (infinite iteration on languages) and reduce the problems like (4) to building finite cycles (which reduces the problems to the form (3)). For example, looking for a trace in $\Box\Diamond L$ may be reduced to the search of a finite cycle in $\Diamond L$.
- In difficult cases, tactics based on abstraction can make the task easier; for instance, one may extract the *control graph* of some timed automaton; if this graph is finite, one can use some automatic tools of *SmallTS* to compute candidates for executions, which are confronted to invariants and guards of the original automaton. This tactic helps us to restrict the search space when looking for some execution. Experiments on the use of abstraction for testing are still restricted to the use of control graph; we hope to generalize this technique to more complex cases.

Examples involving tests have been done on enumerated and temporal systems; please refer to [11] for detailed developments.

4 Actual and future development

CClair has been built as an *experimental* framework on transition systems, mixing deduction and computation. On three classes of systems — finite, constrained, timed automata —, the examples we worked on seem to show that various techniques like higher-order resolution, coinduction, term rewriting and *Prolog*-like interaction, put together, provide a powerful framework for working on reactive systems.

One of the examples — the *ABR*-conformance algorithm — is large enough to provide a field of experimentation for the design of new tools. Other examples are available at the project home page.

Nevertheless, the set of rules and theorems we proved is still very incomplete; many of these results came as lemmas required by other developments and examples, and we have to build more systematically our libraries.

Future work may take several directions :

- In their work on "input/output" automata, Müller and Nipkow [23] used the domain theory of *HOLCF*. Our work is largely inpired by theirs, but we worked directly in "pure *HOL*"; it would be interesting to check which improvements some adaptation of *CClair* to *HOLCF* could bring.
- At the present time, the use of our framework needs some competence on theorem-proving. If we want our system to be used by specialists of reactive systems or engineers, we must provide tools to hide a theorem-proving technology which may seem to be restricted to specialists. In the same spirit, the use of the *Isar* style in our developments should make them more accessible.
- During simulation and testing, many unknown variables are generated, and it should be useful to instantiate them as soon as possible. Integration of our tools with some constraint solver should be of a great help.
- The *COQ* proof assistant, based on the Inductive Calculus of Constructions, will soon be enriched with new computation (rewriting) facilities. It would be interesting to adapt our work to this constructive framework, and for instance consider execution generation by *extraction*.
- The main part of our definitions use *Isabelle*'s inductive/coinductive package, which is itself based on the μ-calculus on powersets. Developing further this theory may — besides a scientific interest —, help to unify much of our tools.

5 Related work

Many works have been done about the application of theorem proving to the study of reactive systems. As far as we know, the original aspects of our work lie in its simulation/testing techniques and its aim to genericity.

As said above, the closest work to ours seems to be Müller and Nipkow's theory of input/output automata. Theories of traces and executions are developped, and automatism by rewriting works well.

The *Temporal Logic of Action* (*TLA* [16]) is another verification framework for reasoning about concurrent systems. A formalization of *TLA* is written by Stephan Merz and is available with the sources of *Isabelle/HOL*. Unlike *CClair*, both systems and properties are represented by logical formulas. *TLA* comes with a wide collection of deductive rules some of which have been translated into our formalism.

STeP [7] is a tool that supports both deductive and algorithmic approaches for the verification of reactive systems. Properties are expressed using linear temporal logic and are reduced into a set of first-order formulas by applying some predefined deductive rules. *STeP* integrates a model-checker for finite states systems and the automatic computation of invariant properties. Because *CClair* is based on *HOL*, it seems to be more flexible : systems can be defined in a natural way, possibly by supplying declarations of some higher order constants, and the specification of properties to be checked is not limited to the *LTL* formalism. Moreover, *Isabelle* offers to *CClair* a high level of upgradeability : any proven theorem may be considered as a potential deductive rule.

6 Conclusion

CClair presents a complete formalization of the theory of labeled transition systems, which serves as a starting point to define more complex models. These models inherit the results and tools developed in the core theory.

Moreover, it is possible to use at the same time different formalizations, compare them, and merge verification and testing activities into a single one : augment the knowledge of some automaton or family of automata.

The evolution of *CClair* will take into account the need for interfaces usable by non-specialists, by hiding the complexity of logic, and building bridges with already existing tools.

References

1. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.
2. A. Arnold, D. Bégay, and P. Crubillé. *Construction and analysis of transition systems with MEC*. World Scientific Publishers, 1994.
3. A. Arnold, A. Griffault, G. Point, and A. Rauzy. The altarica formalism for describing concurrent systems. *Fundamenta Informaticae*, 34:109–124, 2000.
4. André Arnold. Transition systems and concurrent processes. In *Mathematical problems in Computation theory*. Banach Center Publications,vol. 21, 1987.
5. B.Bérard, P.Castéran, E.Fleury, JF.Monin L.Fribourg, C.Paulin, A.Petit, and D.Rouillard. Automates temporisés calife, 2000. URL = http://www.loria.fr/projets/calife/WebCalifePublic/FOURNITURES.
6. B. Bérard, L. Fribourg, F. Klay, and J.-F. Monin. A compared study of two correctness proofs for the standardized algorithm of ABR conformance. Research Report LSV-99-7, Lab. Specification and Verification, ENS de Cachan, Cachan, France, August 1999. 27 pages.
7. Nikolaj Bjorner, Anca Browne, Michael Colon, Bernd Finkbeiner, Zohar Manna, Henny Sipma, and Tomas Uribe. Verifying temporal properties of reactive systems: A step tutorial. *to appear in Formal Methods in System Design*, 2000.
8. Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Are Timed Automata Updatable? In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer–Verlag, 2000.
9. S. Brlek and A. Rauzy. Synchronization of Constrained Transition Systems. In H. Hong, editor, *Proc. of the First International Symposium on Parallel Symbolic Computation — PASCO'94*, pages 54–62. World Scientific Publishing, 1994.
10. P. Castéran and Davy Rouillard. Reasoning about parametrized automata. In *Proceedings,* 8^{th} *International Conference on Real-Time System*, volume 8, pages 107–119, 2000.
11. Pierre Castéran and Davy Rouillard. The cclair project, 1999. `http://dept-info.labri.u-bordeaux.fr/~casteran/CClair/`.
12. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. *Lecture Notes in Computer Science*, 1066:208–219, 1996.
13. Robert L. Constable et al. *Implementing mathematics with the Nuprl Proof Development System*. Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

14. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *Lecture Notes in Computer Science*, 1254:460–463, 1997.
15. Robert M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, July 1976.
16. Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
17. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design: An International Journal*, 6(1):11–44, January 1995.
18. N. Lynch, F. Vaandrager, F. backward, s Part, and I. untimed. Information and computation, 1995.
19. N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989.
20. R. Milner. *Communication and concurrency*. Prentice-Hall, 1989.
21. M.J.C. Gordon. Introduction to the HOL system. In M. Archer, J.J. Joyce, K.N. Levitt, and P.J. Windley, editors, *International Workshop on Higher Order Logic Theorem Proving and its Applications*, pages 2–3, Davis, California, August 1991. IEEE Computer Society, ACM SIGDA, IEEE Computer Society Press.
22. Olaf Müller. I/O automata and beyond - temporal logic and abstraction in Isabelle. In *TPHOL'98, Proc. of the 11th International Workshop on Theorem Proving in Higher Order Logics*, pages 331–348. LNCS 1479, 1998.
23. Olaf Müller and Tobias Nipkow. Traces of I/O-automata in Isabelle/HOLCF. In M. Bidoit and M. Dauchet, editors, *Proceedings of the Seventh International Joint Conference on the Theory and Practice of Software Development (TAPSOFT'97)*, Lille, France, April 1997. Springer-Verlag LNCS 1214.
24. Christine Paulin-Mohring. The coq project, 1999. http://coq.inria.fr.
25. Lawrence C. Paulson. The Isabelle reference manual. Technical Report 283, University of Cambridge, Computer Laboratory, 1993.
26. Lawrence C. Paulson. Mechanizing coinduction and corecursion in higher-order logic. *J. Logic and Computation*, 7:175–204, 1997.
27. Amir Pnueli. Deduction is forever. World Conference on Formal Methods (FM'99), Toulouse, France, 1999.
28. Robert Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1994.
29. Davy Rouillard. Le modèle des p-automates dans CClair. rapport technique 1242-00, LaBRI, 2000. also available in "`http://dept-info.labri.u-bordeaux.fr/~casteran/CClair/`".
30. N. Shankar, S. Owre, and J. Rushby. The pvs proof checker: A reference manual. Technical report, Computer Science Laboratory, SRI International, Menlo Park CA, 1993.
31. Daryl Stewart. The hol system, 1998. http://www.cl.cam.ac.uk/Research/HVG/HOL.