Verifying Mobile Processes in the HAL Environment*

G. Ferrari[1], S. Gnesi[2], U. Montanari[1], M. Pistore[1], and G. Ristori[1,2]

[1] Dipartimento di Informatica, Università di Pisa
[2] Istituto di Elaborazione dell'Informazione - C.N.R., Pisa

Gioia Ristori has recently left this for a better world. She will however remain with us for ever.

1 Introduction

The *HD Automata Laboratory* (HAL) is an integrated tool set for the specification, verification and analysis of concurrent and distributed systems. A basic notion for the HAL environment is that of *history-dependent automata* (HD-automata) [11]. As ordinary automata, they are composed of states and of transitions between states. However, states and transitions of HD-automata are enriched with sets of local names. In particular, each transition can refer to the names associated to its source state but can also introduce new names, which can then appear in the destination state. Hence, names are not global and static entity but they are explicitly represented within states and transitions and can be dynamically created. HD-automata have shown to be appropriate to model systems whose behaviours are *history dependent*, i.e., systems where the observable behaviour of a step of a computation may depend on what has been done in the past steps of the same computation. An interesting example of history dependent behaviours is provided by mobile processes as specified in the π-calculus [6]. Its primitives are simple but expressive: channel names can be created, communicated (thus giving the possibility of dynamically reconfigurating process acquaintances) and they are subjected to sophisticated scoping rules. In the π-calculus history dependency manifests itself as the ability of referring names created at run-time by previous communications. In [7] a procedure is described which allows *finitary* π-calculus agents to be represented by finite-state HD-automata. Similar mappings have been defined for CCS with causality and for CCS with localities [10]. Moreover, finite HD-automata have been also obtained for history-preserving semantics of Petri nets [8, 9].

The HAL environment includes modules which implement decision procedures to calculate behavioural equivalences, and modules which support verification of behavioural properties expressed as formulae of suitable temporal logics. In

* Work partially founded by CNR Integrated Project *Metodi e Strumenti per la Progettazione e la Verifica di Sistemi Eterogenei Connessi mediante Reti di Comunicazione*, CNR Integrated Project *Modelli e Metodi per la Matematica e l'Ingegneria* and Esprit Working Group *CONFER2*.

this note we provide an overview of the current implementation of the HAL environment. The environment has been successfully applied in the specification and verification of mobile processes defined as π-calculus agents. Two major case studies of mobile agents (the handover protocol for mobile telephones, and a web browser specification) were carried out within the HAL environment. A fuller account of the case studies may be found in [4, 5].

2 System Overview

Figure 1 presents an overview of the HAL environment. The dashed boxes indicate work-in-progress, i.e., modules which are under development. The HAL environment allows π-calculus agents to be translated into ordinary automata, so that existing equivalence checkers can be used to calculate whether the π-calculus agents are bisimilar. The environment also supports verification of logical formulae expressing desired properties of the behaviour of π-calculus agents. To this purpose, we found convenient to exploit a logic with modalities indexed by π-calculus actions, and to implement a translation of this π-logic into a logic for ordinary automata. Hence, existing model checkers can be used to verify whether or not a formula holds for a given π-calculus agent.

In the current implementation the HAL environment consists essentially of five modules: three modules perform the translations from π-calculus agents to HD-automata, from HD-automata to ordinary automata, and from π-logic formulae to ordinary ACTL formulae. The fourth module provides routines that manipulate the HD-automata. The fifth module is basically the JACK system [1] which works at the level of ordinary automata and performs the standard operations on them like behavioural verification and model checking. The idea behind the JACK environment is to combine different specification and verification tools, independently developed, around a common format for representing ordinary automata: the FC2 file format [2]. FC2 makes it possible to exchange automata between JACK tools. Indeed, the fifth module in HALis simply a filter that calls the already existing functionalities of JACK. Hence, the JACK bisimulation checker MAUTO is used to verify (strong and weak) bisimilarity of π-calculus agents. Automata minimization, according to weak bisimulation is also possible, by using the functionalities offered in JACK by the HOGGAR tool. Moreover, the ACTL model checker AMC is used for verifying properties of mobile processes, after that the π-logic formulae expressing the properties have been translated into ACTL formulae.

The HAL environment supports a textual user interface to invoke the commands in the modules of the system. For instance the command "*hdaut* := `buildHD` *agent*" is used to to generate the HD-automaton *hdaut* associated to the π-calculus agent *agent*. The command "*aut* := `buildFC2` *hdaut*" generates the ordinary automaton *aut* from *hdaut*. Appropriate diagnostic information is returned to the user. We are currently working on a graphical user interface.

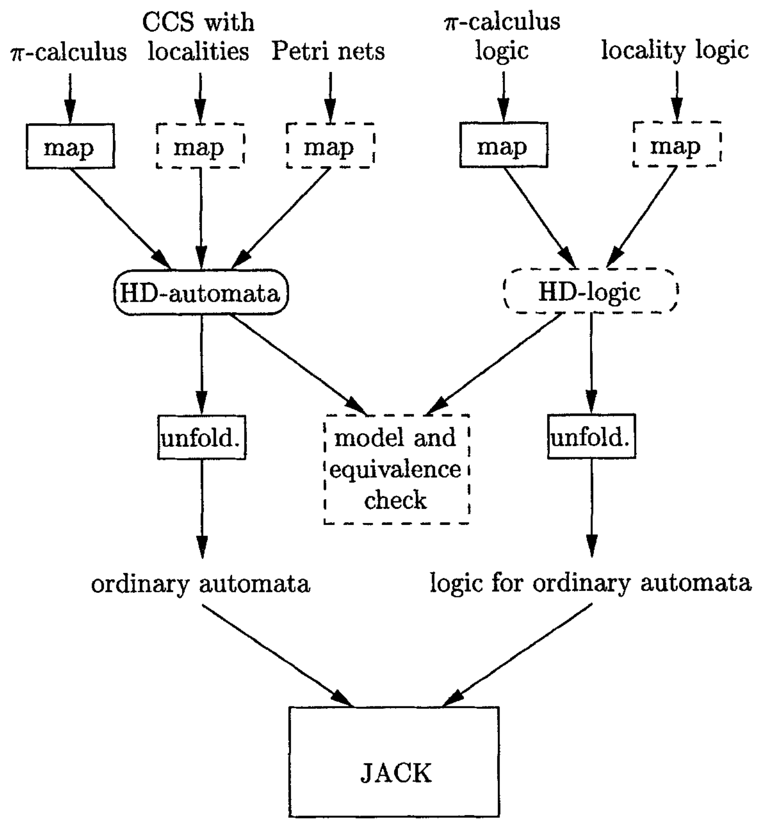

Fig. 1. The HAL environment: an overview.

The HAL environment is written in C++ and compiles with the GNU C++ compiler. It is currently running on SUN stations (under SUN-OS) and on PC stations (under Linux).

3 A Case Study: The Handover Protocol

As a case study we consider the specification of the core of the handover protocol for the GSM Public Land Mobile Network proposed by the European Telecommunication Standards Institute. The specification is borrowed from [12]; it consists of four modules:

- a *Mobile Station*, mounted in a car that moves through two different geographical areas (cells); it provides services to an end user;
- a *Mobile Switching Centre*, that is the controller of the radio communications within the whole area composed by the two cells;
- two *Base Station* modules, one for each cell, that are the interfaces between the Mobile Station and the Mobile Switching Centre.

Table 1. Performance issue

command	states	transitions	time
hdaut `:=` `buildHD` *handover.pi*	11015	21774	4473 sec.
aut `:=` `buildFC2` *hdaut*	32263	62990	442 sec.
min-aut `:=` `minimize` *aut*	49	91	10 sec.
`verify` *no-loss-of-messages* `on` *min-aut*	—	—	6 sec.

The observable actions performed by the Mobile Switching Centre are the input of the messages transmitted from the external environment through an *input* channel. The observable actions performed by the Mobile Station are the transmissions, via an *output* channel, of the messages to the end user. The communications between the Mobile Switching Centre and the Mobile Station happen via the base corresponding to the cell in which the car is located. When the car moves from one cell to the other, the Mobile Switching Centre starts a procedure to communicate to the Mobile Station the names of the new transmission channels, related to the base corresponding to the new cell. The communication of the new channel names to the Mobile Station is done via the base that is in use at the moment. All the communications of messages between the Mobile Switching Centre and the Mobile Station are suspended until the Mobile Station receives the names of the new transmission channels. Then the base corresponding to the new cell is activated, and the communications between the Mobile Switching Centre and the Mobile Station continue through the new base.

There are two kinds of correctness verification that can be done in the environment. One is the checking that the specification of the system is bisimilar to a more abstract service specification in which the system is simply seen as a particular buffer from the input channel to the output channel. The other one is the checking of properties expressed as π-logic formulae, that the specification must satisfy to meet the desired behaviour: for instance, no message can be lost from the input to the output channel and the order of the messages must be preserved. Both these kinds of verification have been successfully performed in our verification environment.

In Table 1 we report the time spent in the different steps of a typical session of verification (performed on a SUN Ultra workstation) for the handover protocol, as well as the number of states and transitions of the automata that are built in the different steps.

4 Concluding Remarks and Future Works

The current implementation of the HAL environment works just on π-calculus agents and on π-logic formulae. As future developments we plan to extend the environment in several directions. The implementation of translation modules from other history dependent calculi to HD-automata is under development.

Moreover, we plan to include in HAL a verification module which implements decision procedures for behavioural equivalences and model checkers directly on HD-automata; this is convenient since ordinary automata have often dramatically more states and transitions than the corresponding HD-automata.

The *Mobility Workbench* (MBW) [13] is another existing tool for verifying properties of π-calculus agent. In the MWB the verification of bisimulation equivalence made *on the fly*, that is the state spaces of the agents are built during the construction of the bisimulation relation. The model checking functionality offered by the MWB is based on the implementation of a tableau-based proof system [3] for the *propositional μ-calculus with name-passing*. The main difference between our approach and that adopted in the MWB is that in HALthe state space of a π-calculus agent is built once and for all. Hence, it can be minimized with respect to some minimization criteria and then used for behavioural verifications and for model checking of logical properties.

References

1. A. Bouali, S. Gnesi and S. Larosa. The integration project for the JACK environment. *Bullettin of the EATCS*, 54, 1994. Detailed information about JACK are also available at `http://repl.iei.pi.cnr.it/Projects/JACK`.
2. A. Bouali, A. Ressouche, V. Roy and R. de Simone. The FC2Tools set. In *Proc. CAV'96*, LNCS 1102. Springer Verlag, 1996.
3. M. Dam. Model checking mobile processes. In *Proc. CONCUR'93*, LNCS 715. Springer Verlag, 1993.
4. G. Ferrari, G. Ferro, S. Gnesi, U. Montanari, M. Pistore and G. Ristori. An automata-based verification environment for mobile processes. In *Proc. TACAS'97*, LNCS 1217. Springer Verlag, 1997.
5. S. Gnesi and G. Ristori. A model checking algorithm for π-calculus agents. In Proc. ICTL'97. Kluwer Academic Publishers, 1997.
6. R. Milner, J. Parrow and D. Walker. A calculus of mobile processes, Part I and II. *Information and Computation*, 100(1):1–77, 1992.
7. U. Montanari and M. Pistore. Checking bisimilarity for finitary π-calculus. In *Proc. CONCUR'95*, LNCS 962. Springer Verlag, 1995.
8. U. Montanari and M. Pistore. History dependent verification for partial order systems. In *Partial Order Methods in Verification*, DIMACS Series, Vol. 29. American Mathematical Society, 1997.
9. U. Montanari and M. Pistore. Minimal transition systems for history-preserving bisimulation. In *Proc. STACS'97*, LNCS 1200. Springer Verlag, 1997.
10. U. Montanari, M. Pistore and D. Yankelevich. Efficient minimization up to location equivalence. In *Proc. ESOP'96*, LNCS 1058. Springer Verlag, 1996.
11. U. Montanari and M. Pistore. History-Dependent Automata. To appear as Technical Report, Department of Computer Science, University of Pisa, 1998.
12. F. Orava and J. Parrow. An Algebraic Verification of a Mobile Network. *Formal Aspects of Computing*, 4:497–543, 1992.
13. B. Victor and F. Moller. The Mobility Workbench — A tool for the π-calculus. In *Proc. CAV'94*, LNCS 818. Springer Verlag, 1994.