Process Calculi, from Theory to Practice: Verification Tools

Gérard Boudol
Valérie Roy [*]
Robert de Simone
Didier Vergamini [†]
I.N.R.I.A.
Route des Lucioles
Sophia-Antipolis
06561 Valbonne CEDEX
France

Abstract

We present here two software tools, AUTO and AUTOGRAPH. Both originated directly from the basic theory of process calculi. Both were experimented on well-known problems to enhance their accordance to users expectations.

AUTO is a verification tool for process terms with finite automata representation. It computes minimal normal forms along a variety of user parameterized semantics, including some taking into account partial observation and abstraction. It checks for bisimulation equivalence (on the normal forms), and allows powerful diagnostics methods in case of failure.

AUTOGRAPH is a graphical, non syntactic system for manipulation of process algebraic terms as intuitively appealing drawings. It allows graphical editing by the user, but also visual support for display of information recovered from analysis with AUTO.

1 Introduction

The theory of process calculi as started with CCS [Mil 80] resulted in a number of verification tools designs, mostly in the case of terms with finitary representation (finite automata) [CPS 89,BoC 88,GLZ 89]. Part of these attempts was AUTO[Ver 87b,LMV 87a], which originated as a (strong- and weak-) bisimulation congruence checker on terms of the MEIJE algebra [Bou 85].

Such tools can easily build large transition systems and check two of them for bisimulation, on a scale unmanageable by a human operator [Ver 86,Ver 88]. In addition the complexity of the growth of these systems can be cut down to some extent by using the congruence properties in order to reduce subterms first, before setting them in parallel. This is especially true for the weak congruence. Specific algorithms were studied, which are now fairly established. Such algorithms proceed along the following line: first devise a normal form of some kind by reducing each term individually, then perform the so-called *partitioning algorithm* to equate both terms to be proven bisimilar.

[*] ENSMP-CMA Sophia-Antipolis

[†] CERICS Sophia-Antipolis

This approach was pushed further in AUTO [SV 89], under the teachings of practice. Reductions to quotients of automata under various semantical criteria showed to be a promising way of analysis. A syntactical formalism for defining those reductions was then in order. We shall present here the state of the art in AUTO in this domain.

Along with the original definition of the MEIJE algebra in [Bou 85] came the notion of *abstract actions* and abstraction criteria, which are a powerful mechanism for defining levels of atomicity with different granularity, and actually move away from low-level details of basic concrete actions. It is a quite natural generalization of the ideas behind weak bisimulation, giving the user the possibility to decide himself on what is to be considered a relevant "experiment" performed on the system. Similar ideas may be found in [HeMi 85,Par 79]. Although we shall further elaborate on this later on, we can just say here that an *abstract action* is a set (usually regular) of concrete action sequences, to be thought of as "having the same meaning", as long as this sort of experiment is considered on the system.

One point of success is that in general abstracted transition systems are reduced drastically in size. They can be considered as characteristic of a partial vision of the system. This is to be contrasted with a temporal logic approach where statements are already imposed before checking, so that one does not get much out of an answer "no". When defining relevant abstract actions, the user usually provides (sets of) sequences with particular meaning which should appear, as well as others which should not. The presence of undesired actions in the quotient abstracted automaton indicates at once in which conditions they may take place, which is unvaluable information while "debugging" a system. Experiments were conducted in [Lec 89].

Use of AUTO quickly showed that editing of process terms was error-prone, due to misspelling of signal names and other deceptive mistakes that could obscure the communication abilities. This was the price to pay for writing terms in such a low-level formalism.Then a graphical representation of terms was wanted both as more flexible and more immediate than a textual one. Communications could be traced with lines joining ports, instead of using the lengthy notations of renaming and restriction operators, which induced most mistakes. Parallel operators could also be easily generalized to more than two processes for instance. Representation followed the lines of flowgraphs [Mil 79].

The graphical system was named AUTOGRAPH [RS 89]. It was not fully integrated with AUTO so that both can be used to a large extent independently. In particular, AUTOGRAPH's output may easily be turned to any process calculus manipulation system.

In fact the future of AUTOGRAPH resides not so much in graphical edition, as full languages tend to be far more complex than simple process algebras, but rather in graphical support of programs skeletons, including only their process structures, on which to visualize results of manipulation analysis from verification systems. This is nowadays our main direction of effort.

2 A short description of AUTOGRAPH

AUTOGRAPH is a graphical system, fully endowed with multi-window facilities. Functions are applied through a mouse button after selection of a menu in a menu bar. We shall not detail AUTOGRAPH general functionalities here, but rather focus on the nature of edited objects as well as functions specifically dedicated to visualizations of interesting results. Examples of typical AUTOGRAPHic drawings are pictured in the sequel. Let us just mention here that pictures may be printed on paper (and in reports!). AUTOGRAPH generates then a specific *Postscript* translation which makes drawings look much nicer than on the screen (and which separates object types more distinctly too).

AUTOGRAPH knows two main types of editable objects:

Networks

They represent terms and subterms, and are drawn as rectangular boxes. They usually bear ports on their border, which are tied together with straight or broken lines to indicate communications. A communication is called internal if it does not pervade to the father box. Communications need not be named so that all matters of renamings and restrictions are left to the system. The only pertinent names that are required upon signals communications are the port names of innermost boxes, as well as communication names (eventually on the drawn lines) at the outermost level. These may not be guessed of course.

A box may contain a name in order for its content to be drawn in some other window (windows have titles giving names to their full content). Subterms may be shared, so that several boxes in the same window may bear the same name.

A box may also contain one automaton (at most), in which case the display of this automaton may not exceed the box boundaries.

In AUTOGRAPH one may retrieve information produced from AUTO: for example in an AUTOGRAPH **Net** one may highlight the set of states (distributed among all components) corresponding to a given state of a global system produced by AUTO. Then using this primary feature we could display either equivalence classes of such states, browsing back and forth through its scattered states; or behavior paths, by depicting the distributed state jumps, as well as the performed actions and synchronisations at ports at any level up the graphic process tree. This work is still under progress, but does not seem to make any problem.

Automata

They are represented by round-shaped vertices, which are joined by broken line edges. Both edges and vertices may be named, although it is mandatory for edges only.

An edge may actually be named several times, thereby representing several transitions at once. Identically named vertices refer to the same state, but at most one of them may have outgoing edges (it is then the state behavior "declaration", while the others are introduced to avoid loops in drawings). In fact there exist several such short-hand conventions in AUTOGRAPH allowing to simplify drawings. We shall not enter into details here.

Automata may be contained in boxes; alternatively there can be one residing directly inside the window.

Automata representing system components should be entered by the user, as the model of his problem. But one may also depict an automaton as resulting from analysis under AUTO. We call this "exploration". The automaton is not automatically positioned: instead, the initial state is given, and then one-step transitions of any explored state are progressively provided on demand. The reason for this choice was that automatic placement is often disappointing, while progressive unfolding of the states and transitions may lead to interesting considerations (much like simulations of systems).

3 A short description of AUTO

AUTO is a system consisting of a main toplevel loop, in which one may type *commands*. Commands may be of various sorts (including input/output to and from files). But most of them bind identifiers to results of *functions* applied to objects. Functions may be composed from a list of primary functions, which constitute the heart of AUTO. Other usual commands are those binding identifiers to syntactic objects. In this case one has to invoke the corresponding parser explicitly (*e.g.* `parse x = a:stop` is a command parsing a simple MEIJE term). In the former case one simply types `set y = function(...)`.

AUTO knows 6 main types: *(process) terms*, *signals lists* (for sorts of processes), *automata* (for internal representation of compiled systems), *partitions* (for internal representation of equivalence classes of states), *paths* (for sequences of behaviors), and finally *abstraction criteria*.

3.1 Reductions

Abstraction criteria, along with several other notions such as *contexts* [Lar 87], are the syntactical means for AUTO to characterize process behaviors so as to reduce them further. An abstract action is a set of sequences of actions, and in AUTO a regular such set. A criterion is a collection of specific abstract actions, and in AUTO a finite such set.

Abstract actions lead to state identifications, and thus to smaller quotient systems which may be analyzed more easily. This reduction only partially retains properties, but this is under full control of the user. In particular, when the union of all abstract actions does not add up to the full free monoid of possible concrete actions, then certain (sequences of) behaviors may go unnoticed. This amounts to a fairness assumption: such behaviors would not pertain to the abstract model. Think of infinite $\tau-$loops in the weak bisimulation case for instance.

Short-hand notations for functions are used when the criterion to be applied is simple and well-recognized. This is the case of course for weak bisimulation reduction, where we call *a-experiment* any sequence of (more concrete) actions in $\tau\star : a : \tau\star$. This criterion is generalized to the case where only some actions remain visible, while others are renamed to τ.

We can now present a first set of functions in AUTO, some based on the abstraction mechanisms and some on more classical reduction principles. They all share the property that they produce normal forms for automata, from terms, each along a given semantics. They use congruence properties wherever possible. Importantly, these functions may be composed. For details of application, see AUTO's Handbook [SV 89].

tta
constructs the full global automaton corresponding to a term.

mini
constructs a normal form automaton *w.r.t.* strong bisimulation.

obs
constructs a normal form automaton *w.r.t.* weak bisimulation.

tau-simpl
constructs a normal form automaton *w.r.t.* elimination of $\tau-$loops and single $\tau-$transitions.

trace
constructs a normal form automaton *w.r.t.* trace language equivalence.

dterm
constructs a normal form automaton *w.r.t.* determinisation.

exclusion
constructs a normal form automaton *w.r.t.* elimination of transitions whose labels, as compound actions, contain atomic signals declared as incompatible in a parameter used by the function. Thus it trims away branches in the underlying graph.

tau-sature
saturates an automaton using transitive closure of the transitions $\tau\star : a : \tau\star$ and $\tau\star$.

abstract
abstracts an automaton by a given criterion, given as parameter. Unlike the previous functions, this one does not take benefit of congruence properties.

Other similar functions should progressively add up to this list, endowing the user with a consistent range of well-identified functions to create his own reduction notions. A mechanism of user-defined functions is also envisaged, to give name to most popular reduction schemes. An example of desirable function is the *context-dependent* reduction, where one trims away behaviors of the process which are not part of the ones allowed by a given context. A context is a set of sequences of actions and thus amounts to an abstract action.

3.2 Comparisons

Of course resulting automata may be compared, through any of the two functions:

eq
: for checking strong bisimulation, and

obseq
: for checking weak bisimulation.

It was foreseen that the result of these functions should be a temporal logic formula in case of failure, but other recent efforts in this domain have proved it to be a difficult matter, especially due to the size of this synthesised formula. A progressive simultaneous exploration of the two terms seems a more promising method, even though it will be less automated.

Here again several further functions could be added, mainly the preorder comparisons, and a function providing the result of *testing* a process by a given observer (with *may/must* options).

3.3 Analysis

None of the preceding functions keeps unnecessary intermediate informations, for (space) efficiency reasons. For example $\tau-$behaviors do not remember which synchronizations produced them. Still, information is conveyed at two specific points, in the naming of states:

- The name of a state resulting from the expansion of a parallel system is the ordered list of states in components.
- The name of a state in a quotient automaton is picked from a representative of this class in the original automaton.

This information is enough for most cases, for it allows one to retrieve states and paths in original automata from reduced ones. So observations in our "partial view" systems may be uplifted to the most concrete automata. Now a further step would be to regain this information on the process itself. This amounts to retrieve which (sequences of) synchronisations led to $\tau-$behaviors, knowing each time the start and target states. It is under way.

Corresponding functions are:

structure
: provides the external naming of a state in a given automaton. Otherwise names are referred by integer internal row.

path
: provides a path in a given automaton leading from a state to another (or from the initial state). This function should be completed so as to allow an abstract action to indicate admissible behaviors for performed (concrete) actions along this path.

Of course the internal names of states as required by the **structure** function above should not be user-provided, but obtained by the system. To this end there are functions computing (sets of) states enjoying some properties:

dead
: returns the deadlock states of an automaton

diverge
: returns potentially diverging states of an automaton, those with *real* $\tau-$loops (or livelocks).

refusals
returns all states which may refuse to perform a signal outside a given list of signals.

A proper mixture of abstraction criteria and these functions may allow an analysis leading to a concrete result, as sketched in the example of section 4. We are not going to expand this type of functionality in AUTO, trying to spot every property of interest in the literature. Instead, collaboration with systems more directly dedicated to the definition and manipulation of such properties [Arn 89] seems more fruitful.

In order to realize this, while sticking to the main body of process calculi, we introduced a function performing the partitioning algorithm for (strong) bisimulation reduction from a given initial partition. It is called **refined-mini**. It may also help the user defining his own semantical reduction criteria at will.

Finally, it should be remarked that the original partition may itself be produced by another partitioning experiment, possibly with a specific abstraction formulation or otherwise. More generally, one may at any moment want to grasp and analyze which states are equivalent w.r.t. a given semantics. This is the purpose of the following AUTO functions:

strong-partition
returns (an internal representation of) the collection of equivalence classes in an automaton *w.r.t.* strong bisimulation.

weak-partition
same thing, with weak bisimulation.

crit-partition
same thing, with bisimulation parameterized by a given abstraction criterion.

row
provides the row of the class to which a given (concrete) state belongs.

class
provides the list of elements in a class, given its row.

As we mentioned before, both paths and equivalence classes of states can be displayed with AUTOGRAPH on a graphical version of process terms.

3.4 Managing the complexity

There is no miracle to what AUTO may do in this domain. Efficient data structures and algorithms may push the limit a little further, so that for the time being systems of 10^4 states and around 10^5 transitions may be dealt with in few minutes. For larger systems the problem actually comes from storage limits, more than time bounds. So the solutions advocated in AUTO consist in **never** building full global systems, but instead only reductions of them relying on congruence properties, further enhanced by the partial elimination of unvisible actions, or by abstraction. Another feature here is the division of usual functionalities into smaller-grain functions, allowing finer reduction strategies for the user. For example it was found that the usual weak reduction algorithm, which corresponds to **mini(tau-sature(tau-simpl(process)))** (assuming that **process** contains but one level of parallel nesting, so that we leave away congruence considerations), was in many "symmetrical" problems replaced with benefit by **mini(tau-sature(mini(tau-simpl(process))))**. This is because the transitive completion of transitions performed by **tau-sature** is actually in practice the most consuming of our algorithms, especially in space. So any reduction before this phase is welcome.

Still, observing the complexity growth is not easy. AUTO provides through a collection of flag options the tracing of various measures: time, sizes of subterms at parallel construction, maximal length of τ–sequences to name a few.

It is hoped that these ideas could make up for an analysis environment that makes AUTO a practical tool, while remaining faithful to the grounds of pure process calculi theory.

4 A Small Example

We chose a simple algorithmic solution to the mutual exclusion problem due to Hyman and extracted from [Ray 85], which has the important property of being erroneous, so that one can apply techniques for discovering the reason why.

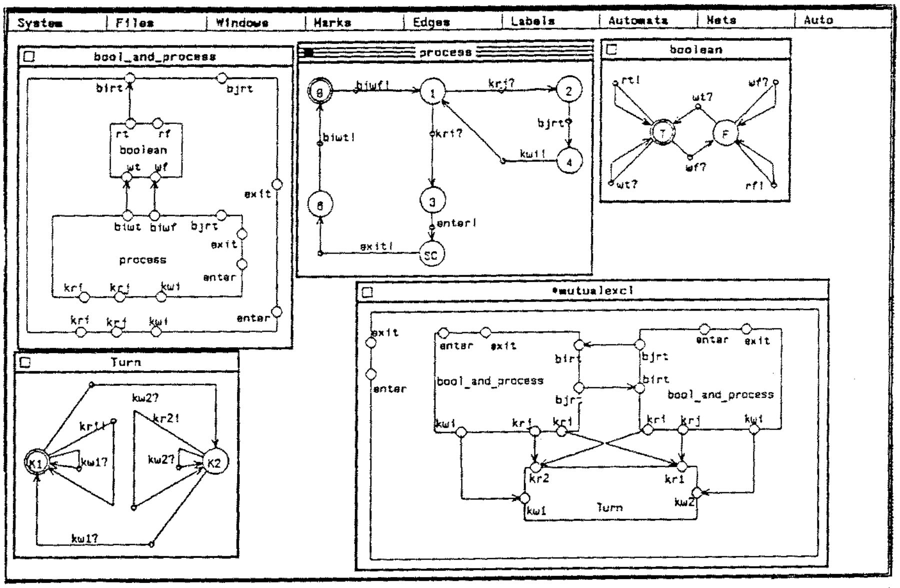

Figure 1: The full example

We represented the problem in process calculi graphical syntax as follows:

- the **mutualexcl** global system is made out of two similar **bool_and_process** halves, each representing a process along with the boolean flag through which it shows its intention to enter the critical section.

- in addition there is a **Turn** process for breaking ties, in case the processes proceed with exact symmetry. This last process is represented by a small automaton (implementing a memory variable with two possible values). Each state corresponds to the privilege granted to the corresponding process.

- the booleans are represented as processes exactly like the **Turn**.

- processes are expressed as automata also, with a straightforward translation from the small algorithmic imperative language they were expressed in originally. Going to (and out of) critical section is represented by the **enter!** (**exit!**) signal emission.

- Other internal signals are encoded with the following conventions on their constituting letters: *b* means "communicating with a boolean"; *k* means "communicating with the Turn"; *r* means "read"; *w* means "write"; *i* means "my own"; *j* means "the other process"; *t* means "true"; *f* means "false". So for example **bjrt?** as appears in the **process** term means: "on reading true as the value of the other process's boolean".

Figure 1 shows the AUTOGRAPHic screen after editing the example. AUTOGRAPH then produces textual files from this graphical representation. We shall now suppose them loaded into AUTO and comment a short AUTO session on it.

```
@ set res=obs mutualexcl;
res : Automaton time = 0.74s
@ display res short;
size = 9 states, 16 transitions, 3 actions.
```

The weak-bisimulation reduced form of the global system is computed (with intermediate reductions on subterms). Its size shows it does not correspond to the expected specification, a loop on enter!:exit! in sequence, which takes only 2 states. One should then devise a couple of actions, one asserting the normal iterative behavior, the other a feared one.

```
@ parse-criterion Verif = crit> good! = (tau*:enter!:tau*:exit!:tau*),
crit> bad! = (tau*:enter!:tau*:enter!:tau*);
Verif : Criterion
@ set Vres = abstract(res,Verif);
Vres : Automaton
@ set V2 = obs Vres;
V2 : Automaton
@ display V2 meije;
let rec st_0 = bad!:st_1 + good!:st_0 + good!:st_2
        and
        st_1 = stop
        and
        st_2 = good!:st_0 + good!:st_2
in st_0
```

The faulty behavior actually takes place. Notice how it leads to a deadlock, as we did not introduce any abstract action allowing two exit! in a row.

```
set Wrongpath = path(res,structure(V2, car(dead V2)));
Wrongpath : Path
time = 0.08s
```

In AUTO the car function stands for Lisp-like "first element in a list". Similarly there are cdr and append.

```
@ show Wrongpath;
2 T-st_6-T-st_6-K1
-- tau --> 0 F-st_2-F-st_3-K1
-- tau --> 1 F-st_3-F-st_3-K1
-- enter! --> 7 F-st_3-F-SC-K1
-- enter! --> 4 F-SC-F-SC-K2
  : Path
```

One has then got full information on a path of the early automaton leading to (a representative of) the erroneous state. In addition, one recollects names of states in a distributed fashion. For now it is not quite readable, but display under AUTOGRAPH should give a substantial help.

5 Conclusions

There are few basic ideas sustaining the relevance of AUTO and AUTOGRAPH to a practical applicability of the theoretic notions behind process calculi.

- The user should be given full freedom to generate his own reduction mechanisms for computing quotient automata for processes, *provided* these reductions are semantically sound and well identified.
- The activity of validation should be as much as possible divided into: reduction first (possibly of an abstract specification also), and then comparison.
- Graphical systems are imperatively needed for displaying the results of analysis at a manageable cost.

These are the future directions of development for AUTO and AUTOGRAPH.

References

[Arn 89] A. Arnold *"Mec: a System for Constructing and Analysing Transition Systems", this volume*

[BoC 88] T. Bolognesi, M. Caneve, *"Squiggles, a tool for the analysis of Lotos specifications", in Proceedings BCS-FACS Workshop on Specification and Verification of Concurrent Systems* (1988).

[Bou 85] G. Boudol *"Notes on algebraic calculi of processes", Logics and Models of Concurrent Systems, NATO ASI Series F13,K. Apt, Ed.(1985)*

[CPS 89] R. Cleaveland, J. Parrow, B. Streffen, *"The Concurrency Workbench", this volume*

[GLZ 89] J. Godskesen, K. Larsen, M. Zeeberg, *"Tav Users Manual", Aalborg University Report* (1989)

[HeMi 85] M. Hennessy, R. Milner, *"Algebraic laws for Non-determinism and Concurrency, JACM 32, (1985)*

[Lar 87] K. Larsen, *"Context-dependent Bisimulation between Processes, TCS* (1987)

[Lec 89] V. Lecompte, *"Vérification Automatique de Programmes Esterel", Thèse de l'Université Jussieu Paris 7,* (1989)

[Mil 80] R. Milner *"A Calculus of Communicating Systems", LNCS 92, Springer-Verlag (1980)*

[Mil 79] R. Milner *"Flowgraphs and Flow Algebras", University of Edinburgh. Edinburgh. Scotland JACM, Vol. 26, No. 4, October 1979, pp 794-818*

[Par 79] R. Parikh, *" Propositional Dynamic Logic: A Survey", LNCS 52* (1979)

[Ray 85] M. Raynal, *"Algorithmique du parallelisme: le probleme de l'Exclusion Mutuelle,* Dunod ed.

[RS 89] V. Roy, R. de Simone, *"An* AUTOGRAPH *Primer", I.N.R.I.A. Report, to be published (1989)*

[SV 89] R. de Simone, D. Vergamini *"Aboard* AUTO*", I.N.R.I.A. Report, to be published (1989)*

[Ver 86] D. Vergamini. *" Verification by means of observational equivalence on automata" Rapport de recherche INRIA No. 501, mars 1986*

[LMV 87a] V. Lecompte, E. Madelaine, D. Vergamini, *"Auto Un système de vérification de processus parallèles et communicants" Rapport Technique INRIA No. 83, mars 1987*

[Ver 87b] D. Vergamini. *"Vérification de réseaux d'automates finis par équivalence observationnelles: le système Auto", Thèse de doctorat 1987*

[Ver 88] D. Vergamini. *"Verification of Distributed Systems: an Experiment", INRIA Report 934, (1988)*