Auto/Autograph

Valérie Roy *
Robert de Simone †

Abstract

We describe the Auto and Autograph tools for verification and analysis of concurrent systems in their more recent developments. Auto is dedicated to a philosophy of "verification by reduction", based on automata morphisms and quotients. Autograph provides a graphical lay-out on which to display both terms and informations on terms, back and forth to Auto. We stress the openness aspects of both systems and their interface formats to the outside world. We see this as a contribution to the evergrowing collaborative trends in between similar tools, mostly under the pressure of national or european Esprit projects.

1 Introduction

Collaborative work in the field of verification tools for process calculi is steadily increasing, mostly under the pressure of european Esprit Projects. Interconnections in between such tools, which appear to be highly complementory, is now becoming a reality which should altogether avoid anarchical outburst and strict bureaucratic overplanning. While describing advances in the design of Auto and Autograph, we insist on their particular forms of interface formats, some of them already common with other tools[GLZ 89, Fer 87, BG 87, CPS 89, Arn 89, BFHP 89].

But the major part of this paper is aimed at describing new features of the Auto and Autograph systems, mostly concerned with the way they deal with recollection of results from analysis after reductions. These informations should be reported on the algebraic process terms themselves (or even better: on their graphical display), while experiments are conducted on their underlying transition systems.

2 Auto

2.1 Presentation

Auto [SV 89] is a support system for verification, and more generally analysis, of process calculi terms. It deals only with terms with finite model representations in the semantics

*ENSMP-CMA Sophia-Antipolis
†INRIA Sophia-Antipolis

of transition systems, and so bases itself on automata transformation algorithms, hereafter called **reductions**. The main activities in Auto may be split in four: Structural Automata Construction, Automata Reduction (with Abstraction), Formal Analysis: Equivalence and other Comparisons and Retrieval of Results and Diagnostics.

Of course those activities are to a large extend entwined, and separated only for the sake of description. Reductions may take place at intermediate levels of construction, due to congruence properties of the equivalence directing the reduction. Partial results lifted back from certain directions of analysis may be reinjected along other lines. By *retrieval of results* we mean the uplifting of informations from the reduced structures back to the early processes. We shall stress this aspect further in a following section.

The general syntaxic means of reduction in Auto is embodied in *abstraction*, using so-called *abstract actions*. Abstract actions represent an higher level class of behaviours. The idea is to consider certain (terminated) sequences of concrete behaviours (signal emissions/receptions) as altogether equivalent and atomic, and to call such a set simply an abstract action. A process which may succeed in completing any of these "runs" at the more concrete level will perform the abstract action then. This framework generalises *observational behaviours* defining weak bisimulation, where any sequence from $\tau^*.a.\tau^*$, with τ a private actions, are to be seen as equal to a simple a.

Abstract actions gather in a new alphabet, called an *Abtraction Criterion*. Abstracting a process results in a system which is conceptually far simpler, and where meaningful activities have been isolated. Pragmatical use of abstraction in "everyday" verification often follows these lines: abstract actions are split in between "positive" ones, leading to configurations of the system to be inquired; then some "negative" ones do represent faulty behaviours which should certainly not take place there. The abstraction is then a refutation attempt. But if the dreaded behaviours do nevertheless occur, they identify a number of (abstract) questionable states, and also questionable behaviours leading to them.

2.2 Data Objects and Types

Auto uses but a restricted number of objects classes, specially meaningful in the approach of *verification by reduction*. They are: Actions and Processes, Abstract Actions and Criteria, Automata, Partitions, Pathes.

Processes and *Abstract Actions* are user-provided and enjoy an external syntax for input to the system. Current algebraic *processes* syntax is adapted from Meije[Bou 85], a language that extends CCS[Mil 80] to cope with the possibility of unrelated signals taking place simultaneously. This leads to a full commutative group of actions in a single step duration. On the other hand efforts are made to allow *any* process calculus as legible syntax to Auto [MV 90]. Current *abstract actions* syntax is based on regular expressions notations, with as generators predicates on signal names. Words from the entailed rational languages should be seen as sequences of behaviours. Full details may be found in [BRSV 89]. These syntaxes may be output by the Autograph interface (described later), from simple graphical representations.

Pathes are just sequences of transitions, alternating states with performed actions, while starting and ending with a state.

Partitions are (most often disjoint) sets of sets of states, representing for instance

equivalence classes amongst all states of a given automaton. A partition is thus almost always linked to a related automaton. In case the partition does not recover all states, missing ones are supposed to share in the same additional class. A one-class partition thus represents a predicate (or property) selecting a subset of an automaton state.

The relation from data structures to conceptual Auto activities is as follows: *Processes* are expanded into *automata* by **structural automata construction**, as defined in the SOS rules formalism; *Abstract actions* help define the **reductions** to be applied on these automata. Fortunately many short-hand notations for specific useful abstractions are also implemented in Auto, so one rarely needs defining them from scratch. Still, dedicated abstractions add power to the expressiveness of reductions; One subsequently extract either *partitions* or *pathes* through **Analysis** and **Equivalence** checking, as informative structures. *Partitions* collect classes of equivalent states, or lists of states enjoying a given behavioural property. *Pathes* are generated in particular as indications of faulty behaviours by application of "abstraction as refutation"; *Partitions* and *Pathes* are reinterpreted and carried back to ther original process algebraic description by **Retrieval of Results**. In this sense both should have in the future a syntactic description to be handed back to graphical display. Another structure which is important to picture is the reduced automaton itself, so that one checks whether the abstract system conforms the expectation, or not.

2.3 Auto as a system

Auto is a small command language consisting of a main toplevel loop, storing results of parsings and computations in global identifiers. *Parsing* commands, are used for entering processes and abstract actions into Auto, possibly from Autograph; *set* commands are used for computing (composition) of functions on objects already stored in the environment, binding the result to new identifier names. All this was described more thoroughly in [BRSV 89].

2.4 Input/output

We stress here to a certain level of detail the formats Auto may use to exchange files with various companion systems. Some were devised in the framework of the french C^3 project, while others bridge and adapt to various european verification tools in the framework of Esprit projects.

Auto supports textual "easily" readable syntaxic formats for reading and writing terms and automata in Meije (or CCS) syntax. While pleasant to deal with, this format suffers two drawbacks: at reading, the parsing is time-consuming –specially on large automata–; at writing, pretty-printing procedures make it space-consuming instead. This format is the one currently used from Autograph to Auto.

Concerning automata, there exists a specific "non-human-readable" format containing exactly the lisp object, which is used for saving and restoring automata, or from the reactive real-time language Esterel to Auto [Res 90]. But future directions lay in the definition of a common format (called `fc` in french) with the tools Mec[Arn 89] and Aldebaran[Fer 87]. The format needs hardly parsing and proves fairly complete to carry

additional optional informations (and in the case of Auto to embed pathes and partitions in their related automaton structure).

Now the situation is far less clear in the case of networks of processes, specially since there is a broad variety of operators used there. First considerations on basic needs shall be expressed while describing Autograph.

2.5 Reductions

All reductions are performed by merging states (of the same automaton), constructing quotients from behavioural equivalence relations. These relations are possibly computed on transformed automata, after abstraction of behaviours has taken place.

2.5.1 Abstractions

Behaviours abstraction functions in Auto are:

1. `abstract`, which uses an abstract criterion as parameter.
2. `tau-sature`, which completes transitions according to prefix or postfix *tau* transitions (building τ^* and $\tau^*.a.\tau^*$).

`Abstract` results in an automaton on a new abstract alphabet, constructed while searching iteratively the concrete one for sequences matching the abstract actions. `tau-sature` is only a useful short-hand for computing weak bisimulation.

2.5.2 Quotients

Semantical equivalences are defined on these refined automata and their behaviours. They use fixpoints of relations, like bisimulations and other branching time semantics, or treat full sequences of behaviours as experiments, like trace languages and other linear time semantics [Gla 88]. All such semantics shall be congruences for all operators of the form of [GV 89]. Such functions in Auto are:

1. `mini` for strong bisimulation congruence.
2. `refined-mini`, starting from an initial partition.
3. `obs` for weak observational bisimulation congruence.
4. `tau-simpl`, which collapses τ-cycles behaviours.
5. `trace` for trace language congruence.
6. `dterm` for trace determinisation (without minimisation). congruence.

All these reductions take optional signal lists parameters, indicating which subset stays visible. Other signals are renamed down to τ. We could add a `refined-trace` functions, using as parameter a one-class partition indicating terminal states. `Obs` is not primitive and could be obtained as `mini(tau-sature(tau-simpl(....`

2.6 Comparisons and Equivalences

We already described some equivalences used for minimizing automata. The same functions may theoretically be used to compare two transition systems, depending whether both initial states lie equivalent.

But here we want to collect informations in case of non equivalence. Supposing both processes were first reduced to normal form, then the final partition retains only singleton or couples as equivalent classes: couples represent equivalent states (one from each automaton), and singletons represent states without match (from either automaton). This structure should be further queried, possibly interactively, or the subset of equivalent states could be displayed graphically, or longest strings of unmatched behaviours running only through singletons could be searched. More actual experiments are still due here.

The functions providing equivalence informations are few in Auto. One should remember that they may be combined with previous reduction functions to compose richer equivalence checking.

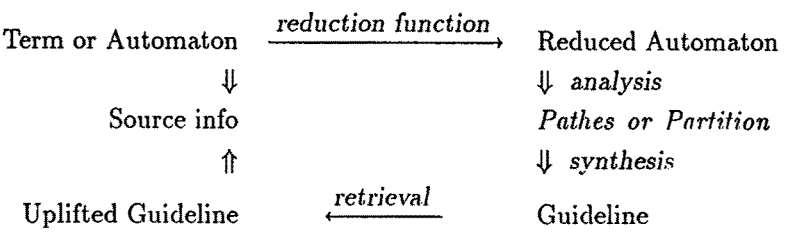
1. `eq` provides a "yes/no" answer on bisimulation equivalence.
2. `strong-partition` provides the list of equivalence classes.

Equivalence is only an instance of comparaison, observation and testing on an automata under analysis. Promising other directions are *graphical specifications* [BL 89].

2.7 Retrieval of Results and Diagnostics

Results put to light by reductions and further analysis should now be lifted back to the initial process terms. Pathes and state partitions will thus be provided "inverse images", so to speak. As a way to attain this, we build a side structure from the shape of the results and the actual combinations of reductions performed. We shall call *guideline* such a structure.

A guideline is a generalized automaton-like observer, safe that it includes additional info in its states, possibly drawn from the process itself. In contrast, an observer is usually required to be defined independantly from its observed system. Typical such info consists in states that are allowed as configurations of the observed process while the guideline reaches this state. Guidelines may also contain another sort of state information, namely places it allows as terminated endpoints of abstract behaviours.



As pictured above, the guideline shall be built going backward stucturally along the chain of reductions performed. Some reductions will change its state space, others shall

add or transform transitions. In the end there shall always be the possibility of **applying** it on the corresponding observed system of the same stage. Applying a guideline on a process (or an automata) will consist in "simulating" the observed process in a way that will satisfy the guideline, or equivalently in a way constrained by the guideline. This will identify which states, or behaviours, of the term under analysis, shall indeed project themselves by reduction down to the chosen pathes or partition classes.

With this reconstruction philosophy the only information which needs be kept through reductions is the relation from a state to the class of states that were mapped to it taking the quotient. These states in Auto are formally encoded in a vector description of states in the sequential automata components occurring in the early algebraic process description.

The operation induced by reduction functions on guidelines are actually quite simple:

quotients replace in each state the set of allowed configurations of the analysed system. It is replaced by its inverse image for the morphism associated with the quotient.

Abstraction replaces actual actions by their concrete actions regular sets counterparts. This induces new states in the guideline, and a need to indicate which sets may be terminal as representative behaviours of the analysed process.

3 Autograph

3.1 Presentation

Autograph [RS 89] is a plain, non syntaxic graphical editor for pictures that are later interpreted into process calculi terms. By this we mean that the drawings produced are not structural –safe in particular spots–, and that coherency and structure are checked only on demand, for translation into textual algebraic form. This allows the graphical description to depart from strict ties to the textual one, including elliptic representation conventions. This allow also semantics of drawings generated from few simple graphical object types to be endowed with scarcely variant semantics, depending on the target formalism and language. Autograph was started with the Meije algebra in mind, but could give graphical versions to (process algebraic simple kernels of) Esterel[BG 87] and Lotos[BB 89].

3.2 Graphical Data Objects and Meanings

3.2.1 Graphics

Autograph works from menus and mouse selection of functions to be applied on graphical objects. As in Auto, the object classes were kept to a bare minimum, highly common, and endowed with graphical links properties which are fairly *ad-hoc*. They are: Boxes, Ports, Vertices, Edges and Labels (not a graphical object)

Roughly, these few graphical elements represent all that was informally used in graphical description of parallel systems (communicating automata) even before efforts were made to embed them into an algebraic syntax formally, which was to lead to process calculi theory.

Boxes are rectangles and represent subprocesses. Boxes next to one another are supposedly set in parallel. Ports are circles, which may only be located on the boxes frames.

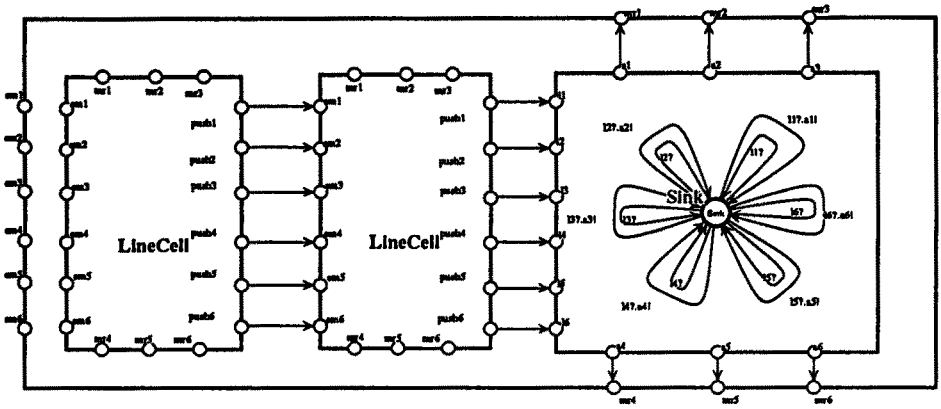


Figure 1: a loosing and duplicating line of the Stenning protocol

They figure the connexion points for communication. Possible communications are represented by edges drawned in between ports. So boxes, ports and edges build up *networks* of processes, and recollect the *structural operators* features from process calculi. Figure 1 shows a network and automaton example.

Vertices are round-shaped and figure automata states. Transitions are figured as edges (just like connections in between ports). So edges are broken lines, whose ends imperatively fall both into states or ports. Vertices and edges thus create *regular individual processes*, and recollect the *dynamic behavioural operators* features from process calculi.

3.2.2 Namings

All these elements may be named (or tagged) using labels. Labels appear on the screen but are not conceived as graphical objects, that is they do not share the same uniform internal treatment as other objects (in particular they may not be themselves named!). Naming is not always imperative. Names serve different purposes depending on the type of object they are applied on:

• on boxes, they provide a global or local naming facility so that a description may be shared in between several windows, and subterms instanciated in several locations. Recursive terms (using themselves as subterms) may also be generated this way;

• on ports, they indicate the local name of the signal to be communicated upon. In particular, all ports of named boxes referring to subsystems drawned elsewhere should be named, and no other need be. This implies that Autograph takes care of all renamings to avoid names clash and generate proper communications at translation into textual form. Naming other ports still may help control this translation;

• on vertices, they help control the translation by providing user-defined identifiers for the state names, which then become the variables occurring in the (guarded right linear) recursive definition of automata term; they may also occur several time in the same automaton (if only once with outgoing edges) and save drawing long backheaded arrows

through a multirepresentation of the same state.
• on edges, they provide on one hand labels for transitions, that are imperative (an edge may actually be labelled several times, thus representing more than one transition); on the other hand they allow naming of connections inter ports, to save drawings of line connections and secure user-provided names at translation altogether. All ports locally sharing connections with the same names are transitivally supposed to be connected. See [Roy 89] for details. All connections pervading to the outside need to be named at this level, to figure the way signals show externally and provide the sort of the description.

To recall, names are either: **imperative**, in the case of transition labels, innermost boxes ports and empty subterms boxes; **short-hand conventions** allowing recourse to textual pointers links rather than common physical line joining, in the case of automata states and communication connections; **naming controls** on the identifiers used at translation into process calculus syntax in the case of states, intermediate levels ports and connections.

3.2.3 Additional semantics

There is a default semantics (expressed as Meije or CCS terms) to the drawings, at least the correct ones. Notions of correct drawings, together with all abuse of representation allowed by Autograph, are out the scope of this paper and described in [Roy 89]. Briefly: Boxes next to one another represent parallel substerms, containment of boxes represent the subterm relation, edges without outside ports generate signal restrictions and edges in between ports with different names generate relabellings. Automata are even more simply interpreted.

But this semantics may be altered, using a system of semantical annotations by functions from a special menu, each time dedicated to a specific new calculus to represent. This holds mostly of networks.

Justification for these annotations is drawned from the Ecrins [MdS 87] system, where a syntaxic formalism is defined for creating new calculi through new operators. A new operator is introduced by syntaxic considerations (name, binding power, type) and SOS rules for semantics. The type is always of the form $(Signals^m \otimes Process^n) \longrightarrow Process$. So we may superpose a semantics (and even several if they may be disambiguated by binding powers) with semantical markings: on a box (the target process). its ports (the *Signals* parameters), its son boxes (the process arguments), their ports (these arguments behaviours).

This part of Autograph is in development and shall be developped hopefully in the final paper.

3.3 Autograph's Interface

3.3.1 Menus and Functions

Autograph contains a number of menus (in its current version). Half are classical screen and window management menus, including cut-and-paste of subdrawings and Postscript dump of improved representations of both networks and automata (with edges splined up for instance). The other half is more specific to the edition of objects. There are four such menus: `Nets` for boxes and ports, `Automata` for vertices, `Edges` for transitions

and connections, and `Labels` for namings. They all contain functions like `select`, `move`, `create`, `kill` and so on...

There is a `completion` function in the `Windows` menu of Autograph, which checks for coherency of drawings and construct the following records at nodes, whose fields may be variant according to the **Type** field:

1. **Type**, the name of the corresponding operator (*e.g.* PAR, SEQ, AUTOMATON, VAR, LOCALDECL)
2. **Name** (mostly in case of variables and local process declarations).
3. **Fathernode** for navigation.
4. **SigParameters** for the relevant arguments to the operator.
5. **SigRename** because signals renamings, which introduce process instantiation, have a specific representation in Autograph (by edges drawing).
6. **Internals** for signals local scope binding, as in CCS restriction.
7. **Sort** giving optionally the set of all visible signals at this subterm level
8. **SubProcessesGraph** identifying the subprocesses and optionally a relation in between them
9. **LocalDeclaredProcesses** for local process definitions that are used in the subprocesses graph (mostly in case of LOCALDECL operators).
10. **Automata** for sequential automata components. These automata will correspond after actual translation to entries in a "automata fc" table.
11. **Pragmas** for uninterpreted info (like graphical positions remainders).

3.3.2 Input/output

There is an additional variable menu in Autograph, or more exactly there are distinct versions of Autograph, depending on a menu providing the semantic annotating and translation functions to a given calculus format, or a companion verification systems (*e.g.* Auto). In the future translation functions should all produce instances of a "common format", prealgebraic. This is largely started for automata with the format described in annex, far less for networks. But globally the format should closely follow the records structure produced by the previous `completion` function.

There is also a specific `explore` function, whose mode is in particular started while inputting a description file which lacks geometrical features, for instance as provided by Auto. Exploration of automata is an established functionality of Autograph. Exploration of networks is being completed, and should be described in the full paper, as well as exploration of *pathes* and *partitions* on already displayed process descriptions. These functions are immediate, provided an "fc" input format for these objects is agreed on.

References

[Arn 89] A. Arnold *"Mec: a System for Constructing and Analysing Transition Systems", in Acts of the Workshop on Automatic Verification Methods for Finite State Systems, LNCS (1989)*

[BFHP 89] B. Backlund, P. Forslund, O. Hagsand, B. Pehrson, "Generation of Graphic Language-oriented Design Environments", 10th IFIP Protocol Specification, Testing and Verification", Twente, North-Hooland (1989)

[BG 87] G. Berry, G. Gonthier, *"The Esterel Synchronous Programming Language: Design, Semantics, Implementation", to appear in Comp. Sci. Prog. (1989)*

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

[BRSV 89] G. Boudol, V. Roy, R. de Simone, D. Vergamini, *"Process Calculi, from Theory to Practice: Verification Tools", in Acts of the Workshop on Automatic Verification Methods for Finite State Systems, LNCS (1989)*

[BL 89] G. Boudol, K. Larsen, *"Graphical Versus Logical Specifications"*, INRIA Research Report 1104 (1989).

[BB 89] E. Brinskma, T. Bolognesi, *"Introduction to the ISO Specification Language Lotos ", The Formal Description Technique Lotos, North-Holland (1989)*

[CES 83] E.M. Clarke, E. Emerson, A. Sistla, *"Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications: a practical approach"*, Proc. 10th ACM POPL (1983).

[CPS 89] R. Cleaveland, J. Parrow, B. Streffen, *"A semantics Based Verification Tool for Finite State Systems"*, in Proceedings of the Ninth International Symposium on Protocol Specification, Testing, and Verification, North-Holland (1989).

[Fer 87] J.C. Fernandez, *"Aldebaran: un système de vérification par réduction de processus communicants*, French Thesis, IMAG Grenoble (1987).

[Gla 88] R. van Glabbeek *"The Linear Time - Branching Time Spectrum", CWI Report RvG 8801*

[GW 89] R. van Glabbeek, W. Weijland, *"Branching Time and Abstraction in Bisimulation Semantics"*, Proceedings 11th IFIP World Comp. Congress, San Francisco (1989).

[GV 89] J.F. Groote, F. Vaandrager, *"Structured Operational Semantics and Bisimulation as a Congruence"*, Proceedings 16th ICALP, Stresa, LNCS (1989).

[GLZ 89] J. Godskesen, K. Larsen, M. Zeeberg, *"TAV (Tools for Automatic Verification) Users Manual"*, University of Aalborg R 89-19 (1989).

[MV 90] E. Madelaine, D. Vergamini, *"Finiteness Conditions and Structural Construction of Automata for all Process Algebras", this volume (1990)*

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

[Mil 83] R. Milner *"Calculi for Synchrony and Asynchrony"*, TCS 25, (1983)

[MdS 87] E. Madelaine, R. de Simone, *"Ecrins, un Laboratoire de Preuve pour les Calculs de Processus"*, (in french), Inria Research Report 672 (1987).

[Res 90] A. Ressouche, *"Ocauto"*, Esterel Technical Documents, CMA-Ecoles des Mines (Sophia-Antipolis) (1990).

[Roy 89] V. Roy, *"Autograph: un Outil d'Analyse Graphique de Processus Parallèles Communicants", French Thesis, Université de Nice (1990)*

[RS 89] V. Roy, R. de Simone, *"An Autograph Primer", I.N.R.I.A. Technical Report 112 , (1989)*

[SV 89] R. de Simone, D. Vergamini *"Aboard Auto", I.N.R.I.A. Technical Report 111 (1989)*