An autograph primer

Valérie Roy, Robert de Simone

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports Techniques

N° 112

Programme 1
Programmation, Calcul Symbolique
et Intelligence Artificielle

AN AUTOGRAPH PRIMER

Valérie ROY
Robert de SIMONE

Octobre 1989

RT_0112

An Autograph Primer
Une Introduction à Autograph

Valérie Roy
ENSMP-CMA
FRANCE
vrg@cma.cma.fr

Robert de Simone
I.N.R.I.A. Sophia-Antipolis
FRANCE
rs@cma.cma.fr

Abstract

This handbook describes the available preliminary version of the AUTOGRAPH system, version 2. AUTOGRAPH is a graphical, non-structural editor. It is dedicated to manipulation of objects relevant to the process calculi algebraic theory originated with CCS.

AUTOGRAPH is interfaced with several verification tools, translating picture drawings into their textual inputs. But perhaps more significantly it is progressively evolving towards a visual support on which to display back the results of experiments conducted in these systems. This is specially true of AUTOGRAPH's companion verification tool AUTO [SV 89].

Full directions on how to run AUTOGRAPH are provided here. Example cases will be found elsewhere [BRSV 89,Roy 89].

Résumé

Cette notice présente la version 2 disponible du système AUTOGRAPH. AUTOGRAPH est un éditeur graphique, non-structurel, d'objets liés à la théorie algébrique des calculs de processus (issue de CCS).

AUTOGRAPH est interfacé avec plusieurs systèmes de vérification, par des traductions depuis les dessins vers leurs formes textuelles d'entrée respectives. Mais, de façon peut-être plus significative, AUTOGRAPH évolue vers un support de visualisation sur lequel montrer les résultats d'expériences conduites dans ces systèmes de vérification. Ceci est surtout vrai pour le système AUTO [SV 89], développé en collaboration avec AUTOGRAPH.

On trouvera ici toute indication permettant d'utiliser AUTOGRAPH. Des exemples de modélisations et d'études sont décrits dans [BRSV 89,Roy 89].

1 Introduction

Welcome to AUTOGRAPH!

What is AUTOGRAPH? It started initially as an early attempt at graphical description and edition of so-called *process calculi expressions*. Such expressions are perceptive syntactic representations of communicating parallel systems. Process calculi theory emerged from the pionneering work of Robin Milner on the CCS algebraic language [Mil 80], now fairly established as a universally praised body of work.

Graphical representations of CCS constructs were introduced from the start, at least as illustrations in reports and papers. They are highly evocative [Mil 79] and somehow more concise than their textual counterparts. So with the development of bitmaps and graphical workstations, it was natural to try and build a system which could allow the user to draw both individual (or *rational*) process components, and then networks around them to set the communication architecture.

Rational processes are represented under the –very primitive– form of *automata*. Representing simple machines as automata has always been common practice in computer science. Automata are very graphic objects indeed.

Networks are depicted straightforwardly, as tradition requires, as boxes representing subprocesses, showing ports on their periphery. These ports could then be linked together with edges laying potential communications. This was exactly what people used to insert in their publications, with the –huge– difference that now these scribblings could be automatically translated into textual form and handed over to verification systems dedicated to process calculi analysis.

Rational individual processes can of course be translated as well, in a syntax that closely sticks to right linear grammars (so think how nicer it is to have them drawn instead!). AUTOGRAPH is "multiwindow", meaning that a large system may be split for description in several windows, each displaying imbricated subsystems down to the rational processes. The linking is realized by name identity, and subdrawings may be shared in surrounding contexts drawings.

The editing functions of AUTOGRAPH are, fortunately, rather plain and easy to comprehend. They are all applied after selection from a menu, in a menu bar. Most need then pointing with the mouse on a specific desired location, where it makes sense to apply them. Sometimes one needs to drag the mouse to give two such locations, like in creating a new process box for instance. But altogether it is so simple to run that this needs hardly mention. Section 2 of this tract presents the basics one ought to know about menus for most AUTOGRAPH usages. Section 3 follows with a short example giving a glimpse of how to proceed the first time. It is not imperative. Finally section 6 describes systematically each function in menu order. It is to be consulted for clarification sake when something does work differently as the user anticipated.

First there are menus for manipulating the system, for reading and saving files, and other general-purpose functionalities one would expect from any multiwindows editor. Worth mentioning is a Postscript output format [Pos1] which produces neat splined up versions of the drawings for inclusion in reports and articles. So we respectfully

send it back from where it came.

Then there are more specific menus, which really make up for AUTOGRAPH's personality: *Automata* for manipulation of automata states, *Nets* for process boxes and ports, *Edges* both for automata transitions and inter-ports connections, and *Labels* for all sorts of naming on the previous objects, amongst which for instance transitions labellings and ports namings.

You may find an additional menu in your version of AUTOGRAPH, which varies according to the particular process calculus, or verification system, that AUTOGRAPH is to be used with. This part of the system is still constantly under work, and you should check with us to what you would want and what does exist at the time you read. Cases of classical calculi CCS and MEIJE[Bou 85] (our "improved" homegrown algebra) are now fairly settled and operative. In addition progress is being –slowly– made towards the Lotos[BB 89] and Esterel[BG 87] languages, as well as the Concurrency Workbench[CPS 89] and Mec[Arn 89] systems. The main nowadays functions of this menu, whatever its name, is to provide translation on file of a textual version, in the convenient syntax, of what was drawn. As much as possible we try to indicate graphically all incoherencies in the drawings which make them unfit for translation. Historically, our main motivation for starting AUTOGRAPH was that we had been involved for quite a while in developping a tool for manipulation and verification of CCS and MEIJE terms. This tool is called AUTO[Ver 87a,SV 89]. It is fully interfaced with AUTOGRAPH, and it is most likely that if you did not bother it is the name of the additional menu you will get as default to AUTOGRAPH.

Now a few introductory words both about AUTO and process calculi theory should be in order, at least as much as needed to then resume the description of AUTOGRAPH. This other part of AUTOGRAPH will be concerned with some more original aspects, somehow more ambitious, which raised from the opportunity we had to practice on AUTO and AUTOGRAPH together. Roughly speaking, the problem was: if a verification system is to be of any use, it has to deal in depth with defining what to do when something "is wrong". And if the user was providing his specification under a graphical form, he shall expect to view these results, when possible, on the very drawings he understands. In fact we want to confront ourselves with the problem of representing graphically all kinds of information *back*, first from AUTO, then from other such verification systems. First attempts in this direction are promissing, but a full realisation is largely a matter of anticipation. Some ideas still need to be further clarified. Presenting them here shall possibly produce feedback suggestions on behalf of the reader.

Process calculi theory provides the semantics of networks under the form of transition systems: first by providing an interpretation of each construct in the syntax as a *transition systems transformer*, then by defining relevant equivalences *w.r.t.* the fineness of semantical criteria desired, so that these equivalences lead to quotient transition systems. This is not the place to recall these notions, dealt with at length in [Mil 80] and [Bou 85].

This philosophy was brought to an ultimate goal in AUTO, where we pretend to deal with verification activities –almost– entirely through transition system constructions and then reductions. The shrinking of transition systems (or here *automata* as AUTO only deals with the finite case), along various angles, brings forth the temporal dependancies in between sophisticated subsets of behaviours. Our claim is that such dependancies are the very statements one would want to prove while analysing systems from the process calculi theory.

The main AUTOGRAPH function dealing with results from AUTO for the time being is the user guided display of compiled automata. AUTO produces files then that are specially prepared for exploration under AUTOGRAPH, starting from the initial states and then progressively unravelling on demand immmediate successors of any further state. States may be aesthetically rearranged in the mean time, but no automatic positioning is attempted.

Typical further informations one could possibly retrieve from analysis in AUTO are *states equivalence classes* on one hand, and on the other *pathes* –or transitions sequences– joining two particular states. Such informations are spotted and put to evidence through the reduction of systems, and then lifted back in AUTO to the original terms. Visualising either of these types requires mostly visualising a global state of the system in a distributed way on the several windows present on screen. This is under way.

2 Getting Acquainted

In the next section we shall use the well-known scheduler example from CCS to illustrate a small editing session conducted in AUTOGRAPH, and demonstrate most of AUTOGRAPH functionalities on it. Other such examples are [Ver 86,Ver 88,BSV].

The present section deals with the basics, what you really cannot do without, concerning invocation of AUTOGRAPH and the first initial moves (including how to exit the system). Also you will find here a condensed *resumé* of the menus, presenting amongst other things all different editable graphical objects. A more thorough formal presentation is the object of section 6.

2.1 Starting the system

AUTOGRAPH is written in Le_lisp [LeLisp] on top of the ASH screen handler. As such it runs for the moment only on SUN3 Workstations. AUTOGRAPH may be invoked directly or from a Suntools window (but **not** from an X11 window yet, although this is next to come). While under Suntools, you are provided in the standard release of AUTOGRAPH with a command named **metro** which starts the system after popping a suitable Suntools Graphic Window (that is to say large enough to hold the top menu bar). Otherwise you may just type **autograph** in any window, followed with an optional argument to be handed to Le_Lisp for its lists size. Default is twenty.

2.2 Beginners guidance

When initiated, AUTOGRAPH fronts the user with a menu bar containing nine different menus. In addition there is a bottom line by which the system dialogs with the user. Initially it reads:

To get on-line help, click <help> button in <system> menu

In your first row or so with AUTOGRAPH it should be a good idea to obey the suggestion: you click with any button of the mouse into the **<system>** menu, then hold the button down while dragging the mouse down until the rectangle containing **<on/off help>** gets lit in inverse video. You then release the button to select the mode. Now all menu selections will generate a short helping message on the bottom line telling you what to do next to apply the function just selected.

From now on, we shall just call selecting a function from a menu the fact that one rolls down this menu with the mouse till the function menu button is reached (highlighted), and the mouse button is released at that location.

2.3 To Exit

When at the **<system>** menu, you should notice the **<quit>** button which is how one leaves AUTOGRAPH. In case there are windows that were edited since last saved

–these display an asterisk in their title–, you shall be queried for exit confirmation. Answer it with "y", "yes", "Y" or "YES" to effectively exit, anything else will get back to AUTOGRAPH.

2.4 Selection of Edition

In AUTOGRAPH you can edit *automata* and *networks*. But since most AUTOGRAPH functions are repetitive and unprotected, we did choose to impose a certain discipline on the user: at a given time there is only one object (automaton/network) which may receive edition orders. It has disturbing aspects for it induces going through selection; but at least one hardly comes to wreck parts of drawings that were not meant to be touched. We describe now all this in details...

There is at most one selected window open for edition at a given time. It is changed using the `<select>` button in the `<window>` menu. This window bears a stripped banner.

The default object open for edition in this window is **the** network –there is at most one network in any given window–. One may select the automata editing mode with the `<select>` button from the `<Automata>` menu (see below). Conversely, one goes back to editing the network by the `<select>` button from the `<Nets>` menu. The edition mode is local to the window.

There are at most as many resident automata as there are process boxes inside of window, plus one automaton directly attached to the window itself. When an automaton is "tied" to a box, its states may not override its binding box's boundaries (or window's).

The binding of an automaton to a box takes place during any of the following:

- the creation of the automaton first state,
- the pasting of the automaton inside of the window from the buffer window (see below),
- the creation of a new box around an existing automaton.
- the killing-off of an old box around an existing automaton.

In any case the automaton gets bound to the innermost automaton-free box containing it in its whole. If no such box is found then the automaton is tentatively bound to the window itself. If the window already binds an automaton, then the function fails and the automaton is not created.

Notice here that creating a new box around an automaton may "steal it" from another box (or window), by capture.

If two partially overlapping boxes contain entirely a newly introduced automaton, it is bound to one of them, more or less at random (the more recently created in fact). Checking which box is bound to a given automaton is possible only by moving the box. In this case the automaton tied to it is shifted accordingly.

There is at most one automaton selected for edition at a time. It may be changed with the <select> function from the <Automata> menu, by clicking inside any state of the automaton to resume editing on it. Clicking outside any state shall create afresh a new automaton, and bind it to the nearest possible box (or again the window). Remember that further states will never be allowed outside the box's boundaries.

2.5 A first look at Menus and Objects

The menus show the following names :

System

contains functions like <quit>, <redisplay>, <clear> for reinitialising the system, and <on/off help> for on-line indications (printed on AUTOGRAPH bottom line).
An additional <on/off mode> button controls printing of other informations on a second display line just above bottom. This information typically comprises

- the *Edition Mode* (to be picked in between Automata and Nets, see above),
- the applicable *Function* presently selected from the menus,
- the current state of a couple of boolean for display control (amongst which the alternative visualisation of external state names *vs* internal numbering for instance, see <Labels> menu).

Files

deals with the input/output of the system, but not with translation of drawings to textual form files. So you may <write> in two different formats: the simple one, where files may be reentered into AUTOGRAPH to resume the session later on, and a Postscript format which actually generates two files, one complete for direct preview and one stripped for being inserted in a report. When a file name is already known from AUTOGRAPH you may use the <save> function instead of <write>.

The <load> function, on the other direction, reads back files written by AUTOGRAPH in the proper format (but not Postscript!). It also reads files prepared by AUTO, with specific contents (like automata to be explored for instance).

Windows

resembles a window menu from any system: creation, deletion, moving, resetting, hiding and exposing w.r.t. other windows, redisplay, resizing (notice <resize> here only work on the lower right corner).

The <size adjust> command will shrink down the window size to scan exactly the inside drawing, plus a small margin.

<select> opens a window for edition; do not try to draw terms in unselected windows!

There is no other way to manipulate windows but with the menu functions.

The `title` command gives name to the window, to appear in its banner. The title is to become the name of the textual term obtained after translation to process syntax.

The `<term debug>` function checks the drawing inside of window for coherency *w.r.t.* this translation, and visually points at possible mistakes.

`<term debug>` and `<title>` shall be described further at the section 5.

Marks

allows graphical "cut and paste" activities on selected objects. Possible to select are *automata states* and networks boxes (see respective menus description below), depending on the present *editing mode*. They can be marked by clicking inside their surface, or by using a `<global mark>` function which lets you then define a rectangular region, dragging the mouse with button down to fix its two opposite points. Then every objects from the relevant edition mode that lays inside gets marked.

Marked subdrawings may be cut or copied to a buffering window, which is popped into sight by selecting the `<see/hide buf>` mode. Together with sets of automata states are copied all transitions, with labels, that share both their endpoints inside this set. Along with sets of boxes are copied their peripherical ports and the possible connections in between them, as well as the automata directly attached to them as content. On the converse, subnetworks of contained boxes are **not** copied along, unless they are each marked as well individually. Odd enough, the tied automata, while copied, are also left in the window to hung and get reattached. So they are duplicated. This is subject to change.

The buffer window may be edited as any other. You may `paste` from this buffer to any other window, provided it be selected! This function is **not** repetitive and should be selected each pasting time.

Automata

allows to edit round-shaped vertices representing automata states. The functions names are self-explanatory safe for the following:

`<rank/name>` switches back and forth from internal to external naming of states.

Internal naming consists of centered integers. As such it is economical in room on the screen.

The external names are either user-provided as a label, or generated by a verification system, like AUTO for instance. In the second case those names may consist in concatenation of states names from rational processes components, which sometimes prove dreadfully long... Still in this case the external names are specially evocative, so that ke kept it possible to picture them. Notice also that one may locally visualise any external name by using the `<see name>` function from the `<Labels>` menu.

<vrtx explore> is a function performing the user-guided lay-out of an automaton starting from a file description. States are explored on demand, providing immediate one-step transitions and creating subsequent states. The user indicates a principal direction and states are layed out in a peacock tail fashion around it.

The <vrtx create> function lets you only add more states to a given automaton, preexisting and selected. States must be kept inside of boundaries of binding box or window.

The <select> function allows you to start a new automaton or select an already existing automaton to resume working on it, by clicking inside any of its states... Trying to edit an unselected automaton is actually the beginner's major upset dealing with AUTOGRAPH.

Nets

permits to edit "boxes" representing subprocess components, and ports setting their places of interconnections. Connections themselves shall be established using the <Edges> menu (see below).

Boxes are created by dragging the mouse from and to their opposite corners. They may contain one another but should not overlap, although no check of this sort is performed until translation into process calculi terms. So you may work with partial drafts in rather messy intermediate state!

Ports are created by clicking inside of desired box. The corresponding port will then show up on the closest orthogonal projection to the box periphery. Ports may not overlap.

Boxes may either be rescaled or resized: the difference is that in the former case ports are shifted accordingly in an homothetical fashion, while in the latter they are simply projected coordinate-wise onto the new box border. Both functions fail in case boxes, ports or attached automata may not be placed back for lack of room.

While <rescale> only works on the box lower right corner, you may select the corner to be shifted by <resize>: just click inside the proper quadrant. The reason while of this is that in the first case the result would be the same up to box move, while in the second it would require a manual shift of all ports.

Edges

is the menu used to draw both the automata transitions and the connections in between ports. To place an edge, one first clicks inside of a state (resp. a port), hold the mouse button down and then release it at every wanted intermediate "nail", as we shall call them. THis successively produces segments from a broken line. The transition (resp. the connection) meets its end when the button is released inside of another state (resp. port). Of course straight lines are perfectly alright. Still, laying two straight lines in between the same states, no matter their directions, is a forbidden situation. This saves to a large extend

from XOR display oddities. Similarly, an edge looping around a state must contain at least two nails.

Nails may be moved, killed and also created on already existing edges using the corresponding function from this menu. To create a new nail, just grab the transition at one point, then hold the button down and leave it at the new desired place (as you would expect).

Edges may be detached from either their source or target state using the <chg source> or <chg target> function respectively. You click on the edge wherever desired and then resume just as if creating the edge onwards. Labels which have to be deported will be distributed along various newly created edge segments.

Labels

Every object in AUTOGRAPH may be labelled (except nails and labels themselves). Unlike other manipulating functions on objects –move, kill– it is shared. So you may name at once all sorts of objects after you selected the function. This may sometimes be misleading, for the accuracy of the clicking location is somehow matter of debate. While the <on/off help> mode is turned on, the type of object AUTOGRAPH understands to be naming is printed each time on bottom line.

To cope with this slight problem we added two user-friendly functions in this menu, namely <see object> and <see name>, which highlight the binding relation in inverse video. In addition, the <see name> function will display full external naming of the object even though the internal numbering display was turned on. This proved very useful (see Automata menu).

Labels in general may be displayed in either external characters string representation or, to save space on screen, internal integer numbering. The switching is realised by the num/name> function.

Labels are effectively created by going through the following sequence:

- First you select the <create> function in the menu.
- Then each time you press the mouse button on an object a small phantom rectangle shall appear; moving this rectangle and then releasing the mouse button will set the label's first character's location.
- Now a special window pops up for you to enter the label's text. A given label may span several lines, separated with <carriage-return>, and you end up with a last <carriage-return> at the start of a line. An initial <carriage-return> on the first line will abort the edition.
- The resulting label will be placed in the drawing at the location you chose moving the rectangle else at the mouse click on the transition, or centered inside the state in case you were labelling a state.

Labels may contain any characters, as these will not be interpreted by AUTOGRAPH graphic edition part, but instead only at translation to textual form of terms.

Labels enjoy their own "cut-and-paste" system of functions in this menu. It is very similar to the one working on graphical objects like boxes and vertices. Names are of course copied along with objects by the other general copying device.

The `<replace>` function turns all occurrences of any given label you shall thereafter click on to another string which you are prompted to enter, the same way as in the `create` function. This allows to recover from spelling mistakes.

Auto

This menu provides functions to be used in relation to the AUTO verification tool for CCS and MEIJE terms. It is still under development and so may vary in the version of AUTOGRAPH you have from what is described here.

The `<write>` function of this menu produces a translation of window pictures into MEIJE terms on files. It will succeed only when drawings are correct, else raise an error message on bottom line. The user can then use the `<term debug>` function of the `<Windows>` menu to track the mistake down.

The `<save>` function is similar, but supposes the file name is already known of by AUTOGRAPH.

In addition this menu offers AUTO dependent features too, such as : `<masked port>` to designate a port as being internal (renamed as τ); `<pilot port>` to define the ticking operator of MEIJE, which drives a process synchroneously to a clock; and `<stop vrtx>` to declare a vertex to be a dead-end of an automaton.

3 A first Editing Example.

Let us turn now to our example. First thing one should produce is figure 1.

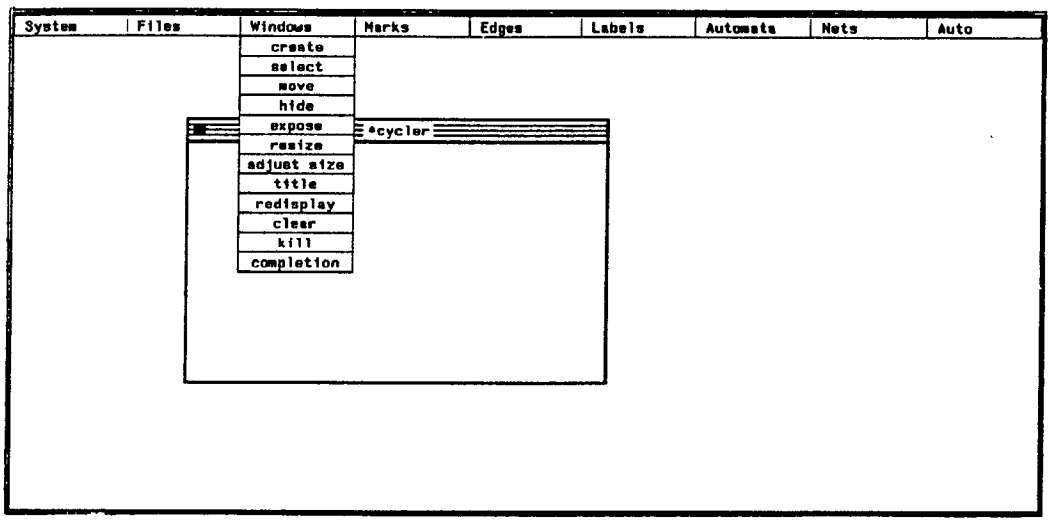

Figure 1: An edition window

This is done with the following succession of moves:

- Select the <create> button in the <Windows> menu, then click with the mouse button in the main editing zone to figure upper-left corner of the desired window. Hold the button down and drag the mouse to set lower-right corner at the mouse location of release.
- Select the <title> button in the <Windows> menu, then click inside of the window (window banners are not sensitive in the present version). You are then prompted on the bottom line for a name to be entered on the keyboard. This name will be used in eventual translations from AUTOGRAPH to other systems as referring to the process then drawned in the window.

Now we shall edit an automaton, providing figure 2.

- First select the <select> function button in the <Automata> menu. Then click at positions inside the window where you want states.

  Notice this function is different from the <vrtx create> which allows you to add vertices to an already selected, preexisting automaton.
- Draw transitions in between relevant states by selecting the <Create> function from the <Edges> menu. The way to draw transitions is described above at the <Edges> menu.

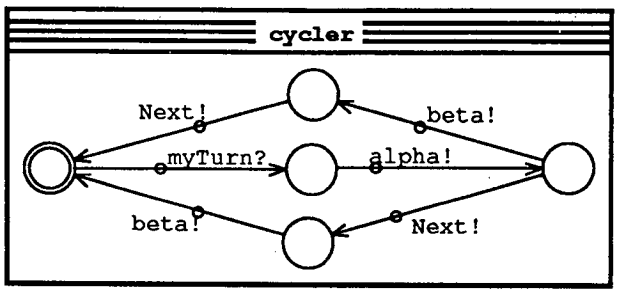

Figure 2: An edited automaton

- Make a state initial by selecting the <initial vrtx> function in the <Automata> menu before clicking inside of state.
- Label the objects, by selecting the <Create> function in the <Label> menu. You must label transitions. One may also label states, but need not to.

  User-provided states names simpler make things easier to remember, specially whenever results are brought back from verification system.

  A given edge may be labelled more than once, thus representing a couple of concrete transitions economically.
- Once the drawing is complete, you can check its coherency with the <term debug> function of the <Windows> menu. Then click inside of window and follow the instructions at the bottom line. They will essentially prompt you to type characters to navigate back and forth in between error messages, or abort. Errors or warnings will result in specific messages while the concerned part of the drawing will be highlighted.

Of course the preceding sequence is in no way mandatory and one can start labelling, then add more states, change transitions and so on.

Last we must produce the network that embeds a ring of such automata, properly connected. Let us do it for the case of three processes, as in figure 3.

- We start by creating a second window entitled scheduler.
- We select the <Create> function of the <Nets> menu and then place a first box by dragging the mouse button down along a diagonal of the desired location.
- Then we select the <Create port> function and click anywhere inside of window to place four of them. They shall appear on the border, at the closest location.
- Then again we select the <Create> function, from the <Labels> menu this time. We then click inside of the box and all four ports in sequence to set proper names as displayed in the figure.

  If needed names may be moved with the <Move> function of the latter menu.

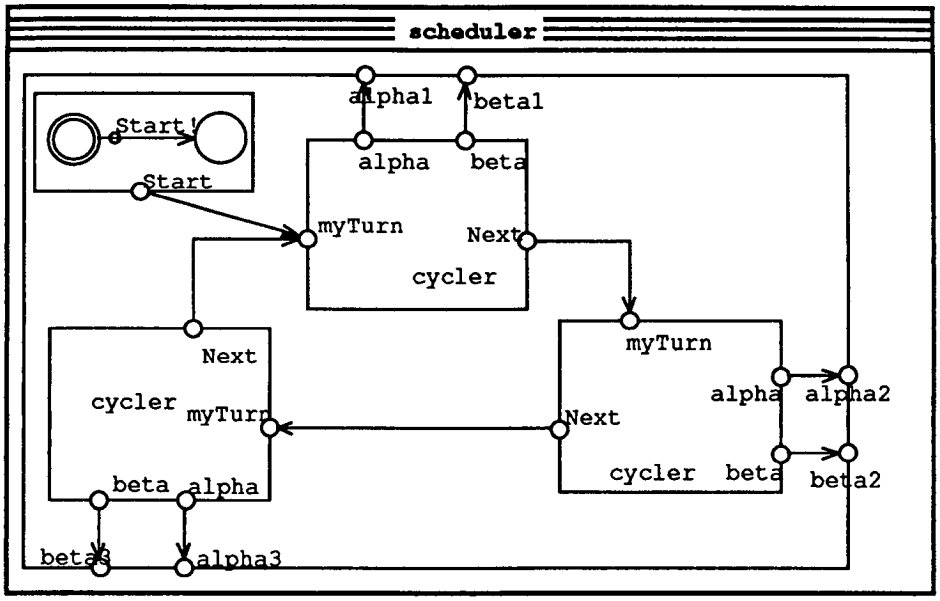

Figure 3: An edited network

- We select the <Mark> function from the <Marks> and mark the box by clicking inside. Ports shall automatically stick to the selected box.
- We use <Copy> from the same menu to stuff this box in the buffer window, then use twice <Paste> to recover two new instances of it in the same window we started with, possibly not overlapping (or else move them afterwards).

  Remember <Paste> is **nonrepetitive**. It provides you with a phantom frame at clicking, and pins it up to show the actual drawing only at release of button.
- After possibly moving ports around, we set the proper connections as shown above, using the <Create> function from the <Edges> menu as for the creation of automata transitions.
- The ring token needs to be originally introduced to the system, from a starter automaton. We chose to draw it in the same window. The drawing of this automaton of course follows the lines of the precedent case. This box may even be named in case one wants to make its content a global variable.

4 Saving Efforts: the conventions

Graphical drawing of process expressions –as opposed to textual versions– is often more adequate and economical in design: n-ary parallel and unnamed internal connexions are instances of this.

But in order to do this and retain all the niceties of the other presentation, we introduced a number of simple conventions to increase the immediacy of the system. The conventions are in no way imperative in the sense that one may perfectly work with AUTOGRAPH without using them. They mostly help do without inopportune far-stretching edges. Instead names are set establishing connections in between equally named entities.

First there are automata conventions. They only concern states out of which no edge leaves, while the state bears a user-provided name. Then two cases occur:

- either the same name does label other states with exactly one of them showing outgoing edges. Then these states have to be conceptually merged into one. This convention is present in graphical representation like LDS for instance.
- If there is no identically named states with outgoing edges, the name supposedly makes reference to another subprocess, described in some other window. The relation shall be set with this second window's title, by name matching.

  Notice this may allow for two-phase automata $(Aut_1; Aut_2)$ to be scattered for drawing in two different windows.

On the other hand there is a way to enforce a state without outgoing edge to really act as a dead-end. This is to mark it as "stop", with the `<stop vrtx>` function of the `<Auto>` menu.

Now we turn to conventions about networks of processes. We shall call *vicinity* a set of sibling boxes, together with their mother if it exists, in the inclusion tree order of boxes. Remember we said boxes should not overlap, at least at translation time.

- An edge may connect two ports which do not belong to the same vicinity. The edge is then crossing through various boxes borders, where we indulge not to place ports when crossing boundaries. The drawing is interpreted as if the ports were effectively present, and unlabelled. In the sequel we shall only consider immediate proximity edges and suppose an internal port generation had been taking place.
- We shall call *wire* a connected sets of ports belonging to the same vicinity. The connection relation here is that of being linked by an edge. In extension we shall consider that **any port not endpoint of any explicit edge is a wire as well.** This is to be kept borne in mind, as it induces what is perhaps the single true convention to be admitted in AUTOGRAPH.

Wires may themselves be labelled (by labelling any edge), but need not be, and most of all must not be labelled twice. It is not clear yet how one should view a labelled edge, spanning over non-adjacent boxes. Actually this should be avoid and raise an error.

- We shall call *equipotential* a set of identically named wires of the same vicinity. All ports included will then be part of the same connection medium scheme. That is to say, synchronisation may take place in between any couple of them. This would correspond in textual version to a renaming that would unify all these labels.

  Remember again that a port without connecting edges is a wire of its own while a port, even if labelled, which shares in a wire should see the name borne by the wire itself, not the port, if one wants to have it included in the corresponding equipotential. While odd at first, this proves to be a pretty convenient feature to have.

  Different unlabelled wires should *not* be seen as belonging to the same (unlabelled) equipotential. Equipotentials may either lead to external ports or not. In the latter case the connection is said to be *internal* to its vicinity. In the former the external ports may be several. This opportunity shall represent simultaneous performance of actions on all external ports of the same equipotential. So this does not make sense when translating to a calculus not endowed with a simultaneity product of actions (same cause, many effects), unlike MEIJE or SCCS.

5 Translating into textual form

As translation goes, most details are pretty straightforward. We shall only provide here some miscellani comments on its less trivial features. A formal presentation will be part of [Roy 89].

There is quite nothing to say about automata translation, apart from the conventions adopted above. An automaton is usually given a (global) name, which is either the title of the window in which it appears alone, or the name of the box framing it.

About networks now: the conventions previously mentioned allow to complete a drawing. Unnamed ports and connections are given labels, using in-as-much as possible the old names, while avoiding clashes. This is true but for the outermost equipotentials and the ports of innermost boxes, for which no guess of their various ports naming would do.

Multiple boxes set in parallel are divided so as to obtain a binary parallel operator. This is done to the best of AUTOGRAPH's knowledge, by partitioning into evenmost halves with the most internalised communications as possible,

If one wants to find out about potential problems occurring at translation, there is a `<term debug>` function in the `<Windows>` menu which scans window's contents as if it were translating, but highlight potential errors instead.

6 Taking Commands in Order

This section sums up all of AUTOGRAPH's functionalities, and describes concisely how they work. As such, it is not intended to be read sequentially. Consult these definitions when stuck somewhere along the track! Unless otherwise specified all functions are repetitive.

6.1 System

redisplay

Refreshes the whole screen. Undesirable characters may sometimes appear on the screen characters or parenthesis. This is rare and does not affect the state of the system, so don't worry.

on/off mode

Boolean indicator, alternatively set on and off by mouse clicking. When on, provides messages displayed on the line before last: information about whether editing an automaton or a net; name of the last menu function selected; on/off position of other global modes (concerning visualisation of labels and so forth).

on/off help

Boolean indicator, alternatively set on and off by mouse clicking inside of button. When on, AUTOGRAPH shall print help messages on last line. These messages try to recall shortly how to apply the function. Mostly they indicate where to click and drag, in which eventual order.

clear

Starts the system anew, clearing away all windows. The user is prompted for confirmation. Answer either "y", "Y", "yes", or "YES".

quit

Exits AUTOGRAPH. If there are windows that were edited since last saved then a query is prompted for confirmation. Answer either "y", "Y", "yes", or "YES".

6.2 Files

load

Prompts for a file name, possibly suffixed with ".sa" for *SunAutograph*. Otherwise this suffix is appended. The file is looked for accordingly. Unix pathes are allowed (but no expansion is ever performed).

save

Writes the content of window into a file, without asking for file name, provided it be known: either since the window was loaded from this file, or has been already written once before in the same session. If no such file exists, it behaves as <write> below. Windows to be saved have to be designated by clicking in each of them in a row.

write

Prompts for a file name. Adds ".sa" as suffix if not present. Unix pathes are allowed. As for <Save>, several windows may be processed successively.

posts save

Same as <save> above, but in Postscript format. Of course here a file name may only be remembered from a previous <posts. write>. Actually two files are created : one bare for inclusion into report, another one prefixed with "pr_" for direct Postscript printing, equipped with header file, scaling factors and other such to ensure alright display on a regular A4 page. As for <Save>, several windows may be processed successively.

posts write

Prompts for a file name. Adds ".ps" as suffix if not present. Unix pathes are allowed. Same as <posts. save>, produces two files. As for <save>, several windows may be processed successively.

6.3 Windows

Care should be taken that for the present the window banners are *not* sensitive!

create
: Requires the user to drag the mouse while holding the button down from a desired corner to the diametricaly oposed corner. There are minimal sizes so that the window may look larger than expected if ever you are attempting to create a really tiny one.

select
: Opens a window for editing. Any editing orders that go to an unselected window will be ignored. The selected window is recognised by its stripped banner.

move
: Displaces windows clicked on to their new position at release of button.

hide
: Pushes the window further down the window stack when clicked inside.

expose
: Pops a window up the window stack when clicked inside.

resize
: First you need select the window with a mouse click. Upper-left corner will stay fixed, while lower-right corner will be taken to the position of release of the mouse button. Aborts whenever the contained drawing (except labels) would not fit into new boundaries.

adjust size
: Shrinks window to the size of contained drawing. Occasionaly may enlarge it to encompass labels. Works on windows by clicking inside.

title
: Prompts for a user-provided name on bottom line. Title names need to fit on one line. Works on windows by clicking inside.

redisplay
: Refreshes the window display. Works on windows by clicking inside.

clear
: Wipes out the window content. Works on windows by clicking inside.

kill
: Removes the window. Works on windows by clicking inside. Prompts each time for confirmation.

completion
: Checks drawings for consistency, and eventually provides warnings and errors messages.

6.4 Marks

mark

Allows to mark subcomponents in the selected window for further treatment. Such components are either boxes or automata vertices, depending on the present editing mode (see subsection 2.4). Selecting a box comprises selecting its ports and an eventual automaton inside, but not selecting its contained boxes (see Global Mark below). Selecting several boxes or vertices comprises selecting their intervening edges. Items are selected by clicking inside.

global mark

Marks all objects inside a rectangular region. One must drag the mouse with button down along a diagonal (a phantom appears). Shall be marked: a box when fully included, a vertex whenever its center lays inside.

unmark, global unmark

Opposite to the previous functions.

move

Moves a rectangular phantom encompassing all marked objects in this window to its new desired location. To do this one drags the mouse with button down. Objects are then placed accordingly.

see/hide buf

Boolean indicator, alternatively set on and off. Controls visualisation of the Copy buffer used by the paste and cut functions. This buffer may be edited as any other window, but not given a title.

cut

Copies onto the Copy buffer from the presently selected window all objects which are marked and which are of the type corresponding to the present edition mode. These objects are erased from their initial window. Cutting a box includes cutting its contained automaton if any.

copy

Same as <cut>, but without erasing objects from the window.

paste

Includes content of Copy Buffer into the presently selected window. Objects location is controlled by a rectangular phantom that appear when pushing the mouse button down. Objects are drawn at time of button release. Unlike most other functions, this one is *not* repetitive.

kill

Same as <cut> above, but objects are not copied onto the Copy Buffer.

6.5 Edges

The <Edges> menu is used for editing both automata transitions and networks connections. So the user should keep in mind that applying a function will succeed only on objects selected for edition. See subsection 2.4.

create

Draws edges in between either vertices or ports couples, depending on the present edition mode. One draws an edge by first pushing the button into a vertex (*resp.* port), keeping it down, and then positioning intermediate nails if desired, one for each mouse button release. The edge meets its end when the button is released inside a vertex (*resp.* port). Straight edges are perfectly alright, but there may be only one, not counting directions, in between two given vertices (*resp.* ports). Looping edges, from a vertex (*resp.* port) to itself, should contain at least two nails. This to limit XOR display problems.

nl create

Adds a nail to an existing edge. Select the edge segment by pulling the mouse button anywhere on it, hold the button down, then release it at desired location. Action labels will fall on the edge side they occupied relative to the clicking location. They are proportionally placed.

nl move

Displaces a nail to location of button release.

see/hide nl

Boolean indicator, alternatively set on and off by mouse clicking. Controls display of black circles at nail positions to help grab them with the mouse.

on/off align

Boolean indicator, alternatively set on and off by mouse clicking. Default is "off". Select automatic straightening of "almost horizontal" –or "almost vertical"– edges segments. Allowed variation is ten pixels. This correction is attempted only at nail creation or nail move, but not at port or vertex move.

source chg, target chg

Changes an edge source (or target). Click wherever desired on an existing edge, then position nails at will by releasing the mouse button at desired locations. The new edge source shall be the first object of relevant type (port or vertex) where the mouse button is released. All labels which were occurring on discarded edge segments shall be distributed on new segments.

nl kill

Obvious. Just click successively on nails.

kill

Kills a whole edge by clicking on it.

6.6 Labels

Labels are shared by all objects inside of window. So you may label several objects at once, unrelative to the present editing mode. So labels may be easily manipulated back and forth from automata to nets.

create

First one needs clicking on any of the following objects: window, box, port, edge, vertex; then you can then move a rectangle standing for the first character to position the labels and thereby a text window is popped up, prompting for a name to label this object. The system says: "Name", followed by an integer. A name may span several lines, and is ended by a <carriage-return> at the start of a line. An initial <carriage-return> on the first line abort the edition. The label is then displayed on the window, at the location of clicking (or centered for vertices).

Only edges may receive several labels along their lay out. For other objects, creating a label shall erase the old one.

see/hide

Boolean indicator, alternatively set on and off by mouse clicking. Controls visualisation of labels. As they may obscure the drawing at first when too many, labels may thus be made invisible to help shape a picture.

num/name

Less radical as the previous function, replaces a label's full name by its internal integer representation. All occurrences of the same label bear the same integer.

move

Obvious. Grab the label by pulling down the mouse button. A phantom shall appear. The label is then placed at the location of button release.

see/hide buf

Boolean indicator, alternatively set on and off by mouse clicking. Default is false.

Controls visualisation of a side window containing last edited label. Content of this window is affected by the <cut>, <copy> as well as <create> functions.

cut

Erases a label when clicked on and saves it in a side buffer window. See above.

copy

Same as <cut>, but leaves the original label as well.

paste

Recovers last edited label from the side buffer label window. Work like <create> without text editing.

see object

Highlights the object the label you click on is attached to. Useful after a number of labels moves or when labels strongly overlap.

see name

Highlights labels attached to the object that is clicked on. An edge may possess several labels.

replace

Replaces all occurrences of a label just clicked on by a new label name which the system prompts the user to enter.

kill

Kills a label by clicking on it.

6.7 Automata

select

Switches the edition mode to automaton. After selecting this function, the first mouse click in the window will be of special importance: if performed inside of an existing vertex of any automaton, it will indicate the user's intention to resume editing this automaton; otherwise if performed inside a box, it will bind the forthcoming edited new automaton to this box, and subsequently shall imply that the automaton should stay inside of box boundaries; if performed in an empty window ,it will bind the automaton to the window itself and if performed outside of any box but in a window containing boxes it will be accept like intermediate edition state.

In any case the system will complain if there was already an automaton residing here.

After this first click the selected function becomes `<vrtx move>` below in case one resumes editing an already existing automaton, or `<vrtx create>` in any other case.

vrtx create

Adds vertices to an already selected automaton if in automaton edition mode. See function above. Vertices cannot overlap, at least not completely.

rank/name

Changes names display of vertices from external labels to an internal representation, without affecting other labels.

vrtx move

A vertex is grabbed by clicking inside. A phantom shall appear whilst the button is held down, encompassing all looping edges around the vertex. Vertex is positioned at location of mouse button release.

vrtx explore

This function concerns automata that were produced in syntax according with AUTOGRAPH but devoid of graphics placements. An unexplored state is endorned with a circle inside of it, smaller even than the circle indicating an initial state. At loading of file containing the unexplored automaton, its initial state, decorated with both these marking circles, is pictured into a newly created window. Editing mode is switched to automaton and this one is selected. One performs a state exploration by clicking inside, hold the button down, then provide a main direction from the center and release the button. All transitions leaving this state shall then be layed out with their target states in a peacock's feather formation, inside of a general ninety degrees angle. In case of an already existing vertex, a nail is positioned instead at the new position, from which a second edge segment leads to the existing edge then.

initial vrtx

Allows one to indicate one or several initial vertices for the edited automaton. Self-inverse, also removes this property. Several initial states shall produce an error at the translation testing.

vrtx kill

Kills vertices by clicking inside. When one kill the last vertex of an automaton we removed the automaton from the window.

kill

Kills the all selected automaton. User is prompted for confirmation, while the concerned vertices are highlighted.

6.8 Nets

select

Changes the edition mode to that of nets in the selected window. Otherwise acts as <box create>.

box create

A box is created by dragging the mouse with button down from a corner to the diametricaly oposed corner. Under a minimal side size the creation abort at the release of the mouse button.

box move

A box is grabbed by clicking on it and holding the mouse button down to new desired position. A box may not be taken outside window limits. Moving a box comprises moving an eventual contained automaton as well as the box's ports, but *not* the subsytem inside of box. See <move> button from <marks> menu.

box resize

A box may be resized from all four corners, by clicking in the appropriate quadrant, then releasing button at new corner location. Ports are projected orthogonally to the corresponding side to where they layed. The processing shall abort would a port fall outside boundaries, a loop edge out of the window or the automaton eventually contain in it too large for the new box size .

box rescale

A box shall only be rescaled from its lower-right corner, by releasing the button to its new location. Ports are moved homothetically. Shall abort for the same reasons tah precedently for the <resize box> function and also in case two ports would come too close and overlap.

box kill

Kills a specified box, leaving its contained automaton to be reattached to the next surrounding structure. This may yield an error in case there was an automaton there already.

port create

To create a port on a box, one clicks inside of box. The port shall appear to the closest location on the border of box. Ports may not overlap.

port move

To move a port one grabs it with the window, then drags it with the mouse button down to new location. Ports in movement follow the box edges, so you should be cautious crossing corners. A port may not be moved from one box to another.

port kill

Kills a port by clicking on it.

6.9 Auto

save

Writes the term in MEIJE process calculi concrete syntax, fit being passed over to AUTO. The window needs to be bound to a file name containing a MEIJE saving already. It may abort due to lack of coherency in the drawing, as described in the previous section. See also <completion>of <windows> menu. As for <save>, several windows may be processed successively.

write

Same as previous, but the system prompts for a file name. As for <save>, several windows may be processed successively.

stop vrtx

Enforces a state to be a deadlock state, in the present version of AUTOGRAPH. Such a state should not have outgoing transitions or translation towards MEIJE format shall raise an error.

masked port

Self-reverse. A masked port shall be considered at translation into MEIJE format as being hidden, its action being made internal (τ) in conventional CCS notations.

pilot port

Used for translation to MEIJE. Provides different semantics to the port, as refering to a clock plug.

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)*

[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)*

[BSV] G. Boudol, R. de Simone, D. Vergamini, *" Experiment with Auto and Autograph on a simple case of Sliding Window Protocol", INRIA Research Report No. 870, (1988)*

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

[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.

[LeLisp] J. Chailloux, *"Le–Lisp de l'INRIA Version 15.2" Le Manuel de référence, INRIA, mai 1986*

[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*

[Pos1] Addison-Wesley Publishing Company *"Postscript Language Reference Manual" Adobe Systems Incorporated*

[Roy 89] V. Roy, "AUTOGRAPH: *un Outil d'Analyse Visuelle de Systemes Concurrents Communicants", Thèse de 3ieme cycle, Université de Nice (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*

[Ver 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)*

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique