Berichte aus dem Psychologischen Institut der Universität Bonn

Band 17 (1991) Heft 3
(slightly modified version of the original report)

MacFAUST: Program for constructing finite automata as instruments in problem solving research. Manual.

Axel Buchner, Lothar Schmitt,
Joachim Funke, & Beate Nikelowski

ISSN 0931-024X

Contents

Summary

This paper presents a manual to MacFAUST (**F**inite **Au**tomata **S**imulation **T**ool), a program that is designed to create and edit discrete dynamic systems. The formal background is based on the theory of finite state automata. MacFAUST also controls experiments investigating human learning in interacting with discrete dynamic systems. Several theoretically well-founded methods of knowledge assessment are implemented, and all data (including a person´s exploratory behavior) are logged. MacFAUST has proven to be a very useful tool for psychological research on learning in dynamic task environments in several experiments. MacFAUST is an easy–to–use, completely menu–driven program that runs on Apple Macintosh computers. Copies of the program may be obtained from the authors.

Part I: About this manual[*]

I.1 Overview

This manual is separated into five parts, the first part containing inevitable information about the program and the manual, and the following two parts explaining how MacFAUST operates using a concrete example. The last two parts are “more abstract” in the sense that they only describe single features of the program without giving detailed examples.

Part I describes the symbols used throughout the manual, some necessary preliminary information for those who would like to play with the program before they read the manual, and a few comments about the purpose of the program and its theoretical background.

Part II describes *in a tutorial way* how one can use MacFAUST to design an automaton. We will show how one can construct an example automaton. At the end of this chapter you will be familiar with the basics of constructing automata with MacFAUST. You can then go on to learn how to run an experiment in Part III of the manual.

Part III describes *in a tutorial way* how one can use MacFAUST to design and run an experiment. We will show how one can set up an example experiment. At the end of this chapter you will be familiar with the basics of running experiments with MacFAUST. To design sophisticated automata and make use of the full capabilities of MacFAUST you should consult chapters in Part III.

Part IV describes additional features of MacFAUST such as how you try out your automaton while editing it (this is a very important feature), how you edit large state transition matrices, or how to make changes to an existing automaton. This chapter is for those who are already somewhat familiar with MacFAUST.

Part V describes the menus MacFAUST offers in the order in which they appear on the screen. Use this section as a reference to look up details that you will inevitably have forgotten as time goes by.

I.2 Explanation of symbols

Before you use the manual you should know what the following symbols and typefaces mean:

👆 This is the standard mouse pointer in MacFAUST. All descriptions of your input are symbolized by this icon.

[*] The development of MacFAUST and this manual have been supported by a grant from the Commission of the European Community (ESPRIT Basic Research Action #3219–KAUDYTE).

This symbol appears whenever we describe the effects of your changes and the feedback from MacFAUST. In addition, we use this rather “technical” font to distinguish this type of information from the rest.

Comments are pieces of information that are not important at a particular occasion but may be interesting or give hints on related topics. They are displayed in this smaller font size. You may skip them if you want.

Menu commands appear in **Chicago** which is the font that is used on the Macintosh to display them.

OK This symbol usually indicates that you must click a Push button to confirm a choice.

◉ This symbol is used to indicate that a particular choice may be made by clicking into a **Radio button**. These buttons indicate that you have “either—or” type choice.

☒ This symbol is used to indicate that a particular choice may be made by clicking into a **Check button**. These buttons indicate that you can combine “multiple” choices.

I.3 Necessary preliminary information—even if you don´t normally read manuals

Many people like to start playing with a new program to see how it works rather than reading manuals, the latter of which often being a tedious if not painful experience. Although we have invested considerable efforts to make this manual as easy to use as possible we of course do understand that some people still will prefer to learn the hard way—i.e. from experience. In any case you should at least know the following details:

1. Creating a new automaton means that you create a “parameter file”. This file contains all the information about your automaton. Even if you run an experiment, almost everything you have specified for a particular experiment is stored in this file.

2. MacFAUST is a completely menu driven program, so at any point in time the menus will tell you what you can do. Commands that are unavailable will appear dimmed. There is one exception to this rule: Some of the experimental conditions are “hidden”. See Part III if you need to know what they are.

3. If you choose **Save** from the **File** menu your data will be written to the parameter file. Note that in order to get your automaton ready for usage (e.g., in an experiment) a few extra figures have to be computed (e.g., the distance of each state from the goal state). This is done when you choose **Save**. However, the necessary computational procedures may be sensitive to slight inaccuracies in

your automaton´s so-called state transition matrix. If this happens, MacFAUST displays a message explaining the type of error that occured. Next, a file dialog appears so you can save your data to a file. Note that this file will contain not enough information to use the automaton. Open it and try to fix the errors in the matrix, then save it again.

4. You can try out how your automaton works any time by choosing **Try OUTomaton** from the **Specials** menu, but you must save your data in a parameter file first. The program forces you to do that and then computes the extra figures mentioned above. If you are not quite sure you really want to save the last changes you have made, simply choose a different name for the file you save these data in.

5. Normally, terminating the program is possible by choosing **Quit** from the **File** menu. However, when you run an experiment you usually have the menu turned off to avoid that subjects do things they are not supposed to do. There is an "emergency break" to stop the program even in this situation. Type **⌘-M** to switch on the menu, then terminate as usual. The subject´s data will be saved to the file you specified at the beginning of the experiment.

If you want to explore MacFAUST´s capabilities by yourself you can now start to explore the discrete system. For those who do not have that much time we have written this manual and recommend that you proceed reading the next chapters.

I.4 About finite automata

MacFAUST has been designed as a tool for researchers interested in how people learn and later use their knowledge in interacting with complex dynamic systems. There is a large diversity of systems currently used in this research domain (see the recent review by Funke, 1991). The problem with most of these systems is that they do not have a homogeneous theoretical background to make them comparable or to allow researchers to manipulate precisely defined properties of the task.
Also, many researchers use diagnostic procedures that are idiosyncratic to the task and have no or no clear connection to the "classical" inventory of cognitive psychological methods of knowledge assessment. There are a number of other problems that have been outlined elsewhere (Buchner, & Funke, 1990, 1993; Funke, & Buchner, 1992) and will not be repeated here. The situation is different with the discrete dynamic systems used here. These systems have the following (plus some more) advantages:

- They are based on a homogeneous formal background. More precisely, the theory of finite automata is used to describe their properties (e.g., Albert, & Ottmann, 1983; Hopcroft, & Ullmann, 1979; Salomaa, 1985).

- They make use of diagnostic measures that have a clear connection to "classical" measures of knowledge within experimental psychology: recognition tests and verification tasks.

- They provide a criterion for optimal performance. Therefore, control performance can be assessed without complication. At any given state of the

system it is possible to determine if there is a sequence of interventions that leads to a defined goals state and how many interventions are required as a minimum.
For more details you are again refered to the papers mentioned above. Having a homogeneous formal background for describing many different dynamic systems makes it possible to create classes of formally well-defined systems that can easily be related to each other. MacFAUST is a tool to do exactly that. In other words, **MacFAUST is an all-purpose discrete system generator**. New discrete systems may be created, and existing ones may be modified according to the researcher´s needs. This manual describes how to do that.
In addition, MacFAUST can control experiments exploring the systems that have been created by the program. This manual describes how to do that also. Subjects´ data are recorded and stored in a text file so they can be conveniently used for data analysis. However, the manual cannot describe all the details of what possibilities the researcher has in analyzing the data produced by MacFAUST. For more details the reader is therefore again refered to existing literature describing experiments with MacFAUST (Buchner, & Funke, 1991, 1993; Buchner, Funke, Schmitt, & Nikelowski, 1990).
A copy of MacFAUST may be obtained from the authors for research purposes only. Commercial distribution is prohibited. The authors´ address for requesting MacFAUST can be found on the last page of this manual.

I.5 Installation and hardware requirements

To install MacFAUST, simply copy the program onto your hard disk. No further installation procedure is required. The program needs a minimum of 2 MB RAM and runs under System Version 6.0.1 or higher. To our knowledge MacFAUST runs on any Macintosh (inclufing PowerPC).

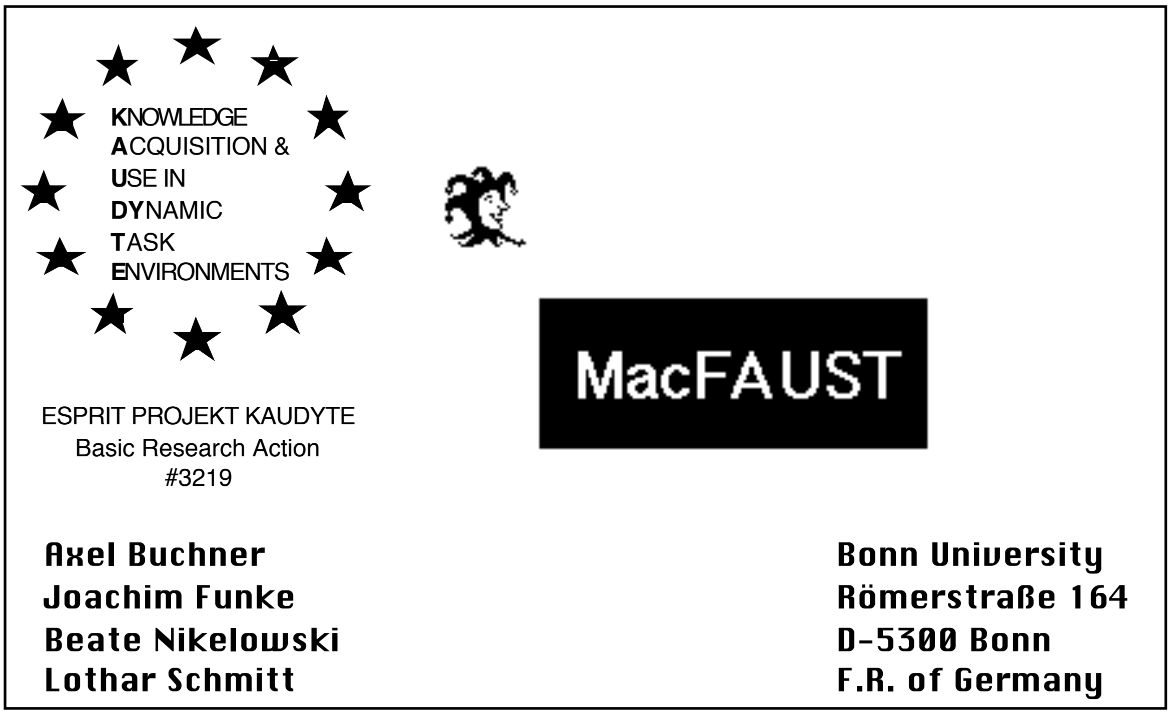
➯ Although we have invested a considerable amount of work into the development of MacFAUST we can, of course, not guarantee that the program does not contain any bugs or malfunctions. We cannot take responsibility for any damage that might occur when the program is used. Also, note that future versions of MacFAUST might perform slightly differently.

Part II: How to design an automaton with MacFAUST

MacFAUST is a completely menu driven program, so at any point in time the menus will tell you what you can do. Commands that are unavailable will appear dimmed. There is one exception to this rule: Some of the experimental conditions are “hidden”. See Part III if you need to know what they are.
Before you can run an experiment you first have to design a new automaton which is equal to creating a parameter file that contains all the information about the automaton. This is what we will do in the present section of the manual in which we show how to construct a small example automaton for illustrative purposes.

II.1 Upon startup

Whenever you start the MacFAUST the screen displays the following message:



KNOWLEDGE
ACQUISITION &
USE IN
DYNAMIC
TASK
ENVIRONMENTS

ESPRIT PROJEKT KAUDYTE
Basic Research Action
#3219

MacFAUST

Axel Buchner
Joachim Funke
Beate Nikelowski
Lothar Schmitt

Bonn University
Römerstraße 164
D-5300 Bonn
F.R. of Germany

At this point the **File** menu offers the following choices:

New to construct a new parameter file.

Open to edit an existing parameter file.

Save to write your parameters to the parameter file. Note, however, that saving your data also implies that MacFAUST computes a few additional figures when saving.

Experiment to run an experiment (you need a valid parameter file to run an experiment).

Experimental conditions to declare the number and nature of the experimental conditions you wish to realize.

Quit to exit the program.

II.2 Planning an example automaton

The first thing you do when you design a new automaton is to figure out how many input buttons and how many states it should have. This is the first thing the program wants to know.

Normally you will use the good old paper & pencil method to construct a rough outline of your new automaton. You should try to do that job carefully. Although MacFAUST allows you to do even drastic changes to your automaton at later occasions it means extra work so it should be avoided.
Let us plan and then create a simple automaton to illustrate how one can go about to achieve this goal. The automaton shall mimic a simplified ticket vending machine. We need to decide on

- the number of input variables and their levels (the input signals),
- the number of output variables and their levels (the output signals),
- the names for the input signals (the “buttons” of the automaton),
- the names for the output signals (the possible “displays” of the automaton),
- the state transition matrix (it contains figures that indicate to which state the automaton will move as a consequence of a given current state and a user intervention).

We need to insert money. Let us assume the machine accepts 1 and 2 ECU coins. We must pull something to get the ticket, and we must have an “eject” option to get our money back in case we change our mind. This makes a total of four input buttons. In terms of a state transition matrix the input buttons are the columns.
As outputs we want the automaton to display the money we inserted (“empty”, “one ECU”, “two ECU”, “enough”, “too much”) and we want it to tell us when we have been successful in operating it (“GOAL”). This makes a total of six output signals. In terms of a state transition matrix the output buttons are the rows.
Look at the following matrix. This *state transition matrix* represents what the simple automaton we want to create should look like.

	Input signals			
	1	2	3	4
States	1 ECU	2 ECU	PULL	EJECT
1: empty	2	3	1	1
2: one ECU	3	4	2	1
3: two ECU	4	5	3	1
4: enough	5	5	6	1
5: too much	5	5	5	1
6: *GOAL*	2	3	1	1

This is how it works: First, we are in State 1 (“empty”). We insert “1 ECU” which results in State 2 (“one ECU”). Next we insert “2 ECU” which results in State 5 (“enough”). Finally, we “PULL” and reach the “GOAL”.

II.3 Creating a new automaton

Let us now create this automaton:

Choose **New** from the **File** menu.

A dialog appears on the screen. It displays the No. of input and output signals. As you can see, the default value for each is “1”.

The number of input and the number of the output variables may be changed using the scroll-bars. You may choose up to five different input and output variables for an automaton. (In our example we leave the default values unchanged).

Click ◉ **Buttons of all lines belong to the same input variable.**

Click **OK** .

A dialog appears on the screen displaying the levels of each input and output variable. As you can see, the default value for each is 2.

To enter the number of levels of the input variable (4 in our example) click on the scroll bar next to the number of input variables.

According to the position of your mouse click on the scroll bar the number of levels of the input variable will be changed.

To enter the number of levels of the output variable (6 in our example) click on the scroll bar next to the number of output variables.

According to the position of your mouse click on the scroll bar the number of levels of the output variable will be changed.

Click **OK** .

Next you see what your automaton looks like: The standard user interface that the program produces. The interface you see now is the same as the interface subjects interact with during an experiment.

Very nice, isn´t it? However, the names of your input and output variables look strange. They are simply what the program puts in there by default.
You want to change these names:

Click in the leftmost input button.

A new dialog appears on the screen.

Enter a name. In this case you should enter “1 ECU”.

Click **OK** .

Repeat this until you have named all of your input buttons according to the state transition matrix displayed above (i.e., “A2” ➝ “2 ECU”, “A3” ➝ “PULL”, “A4” ➝ “EJECT”).

Click in the white part of the output field (it currently displays “V1”).

You may switch to the same input-mask simply by choosing “Output labels” from the “Windows” menu. In the gray part of the output field you can read the name of the output variable. Click on it to edit its name).

A list of the names of all output variables and their levels appears on the screen in the “Labels” column. One of the names is “V1” that you clicked on.

Click on the first one of these names in the “Labels” column.

A new dialog appears on the screen.

Enter a name. In this case you should enter “empty”.

Repeat this procedure for all your output variable levels according to our state transition matrix (i.e., “V2” ➝ “one ECU”, “V3” ➝ “two ECU”, “V4” ➝ “enough” “V5” ➝ “too much” “V4” ➝ “*GOAL*”).

On the left side of this window you can find the “distractor” column. The procedure of entering labels here is identical to the “Labels” column. These distractors may later be used for the verification task (see **V.1** “The Windows menu”: “Options”). At this point you should not change them.

Click **OK** .

Then you will return to the standard user interface that you already know. It will display “empty” as the initial state.

Now everything is done to tell the program *what the automaton will look like*. To tell the program *how to operate* you now must edit the state transition matrix.

Choose **State transition matrix** from the **Window** menu.

A list of all output signals appears on the left side of the screen, and a list of all input signals appears on the right side of the screen.

Click on one of the output signals. For instance, choose “No.:1 empty”.

The output signal you clicked on remains inverted until an input signal is clicked.

Click on one of the input signals. For instance, choose “No.:1 1 ECU”.

A new dialog appears on the screen.

Enter the number of the next state the automaton should move to if this situation occurs. For instance, when you will later use the automaton, and you will be in State 1 (“empty”), clicking Input signal 1 (“1 ECU”) should cause the automaton to display the output signal of State 2 (“1 ECU”). Therefore, enter “2”.

	Input signal			
	1	2	3	4
States	1 ECU	2 ECU	PULL	EJECT
1: empty	2	3	1	1
2: one ECU	3	4	2	1
3: two ECU	4	5	3	1
4: enough	5	5	6	1
5: too much	5	5	5	1
6: *GOAL*	2	3	1	1

Repeat this until you have entered all state transitions according to our state transition matrix. (Here is a hint: You need not edit all 24 cells of the state transition matrix since the default value “1” that has been entered automatically is correct for 8 cells).

Click **OK** .

A new dialog appears on the screen.

You may switch to the same dialog by choosing “Default values” from the “Windows” menu.

Enter the number of the initial state (“1”) and the number of the goal state (“6”).

You must always make sure that your automaton has a goal state and an initial state. In saving, MacFAUST needs these data to compute a number of figures needed for actually “running” your automaton (e.g., the distance of each state from the goal state).

Click **OK** .

Then you will return to the standard user interface.

Congratulations! You have designed your first automaton. The only thing to do is to save the automaton to a file.

Choose **Save** from the **File** menu.

The program comes up with a dialog to ask for some additional information to be stored in the parameter file. This gives you the possibility to enter information that will be saved with the automaton such as the name of the person who designed the automaton, the date, and some additional comments that may be useful if you are working with different versions of an automaton.

Click **OK** to save any changes you have made, but click **Skip** to simply go on and ignore changes you may have made. In the latter case, the original information will be saved.

The program comes up with a dialog to ask for the filename.

Enter the filename. For instance, enter “MATRIX01”.

Click **Save** .

In Part IV that follows later we describe how you can test and change the automaton you just saved. Before skipping to that part of the manual, however, we recommend that read how to design an experiment with MacFAUST. This will facilitate the understanding of concepts used later on in the manual.

Part III: How to run an experiment with MacFAUST

Suppose you have designed your automaton and you now want to prepare an experiment. This section of the manual will provide you with an example of how to do this.

III.1 Planning an example experiment

Like in creating an automaton, running an experiment requires some a priori decisions. We need to decide on the following topics:

- The number of different experimental conditions and their nature (i.e., supports and options).
- The number of intervention trials you want for your subjects to explore the automaton.
- The type of diagnostic procedure you wish to use to assess your subjects´ knowledge, and the number of knowledge probes you want to have for each method.
- The number of “blocks” (cycles of phases with intervention trials and diagnostic phases) you want your experiments to last.

III.2 Setting experimental conditions in MacFAUST

For our example we decide to specify two conditions. Subjects run under the *first condition* will be supplied with two different types of external support. One device tells them their current “distance from the goal-state”, and the second device is an “external memory” that appears automatically and displays miniature versions of past system states. Also, we want the automaton to automatically move to the initial state if a subject has reached the goal state. Subjects run under the *second condition* will have available as support an “external memory” which is “dynamic” in the sense that subjects can use it to make any past state of the system their next present state. All other options will be equal to the first condition.

At this point we will not consider other options that are available for designing experimental manipulations. For details, see **V.1** “The Windows menu”: “Options”.

Let us enter our specifications:

Choose **Experimental conditions** from the **File** menu.

A dialog appears which is divided into two halves: on the left side you choose the number of conditions you want, on the right side you specify what each condition comprises.

To enter the number of different conditions select one of the numbers displayed on the left side of the dialog. For instance, select

◉ **1** ○ **2** ○ **3** ○ **4** ○ **5**

◉ **Condition 1** has already been selected. Choose your setting by selecting

☒ **Support: Distance from goal-state**

☒ **Support: External memory (6 cycles/dyn.)**

☒ **Option: Reset at goal state**

Select ◉ **Condition 2.**

Choose your setting by selecting

☒ **Support: External memory (6 cycles/static)**

☒ **Option: Reset at goal state**

Choose the language you need (◉ **German** or ◉ **English**). This will determine the language that is used for the instructions given to your subjects during the knowledge tests.

Click **Save** to confirm your choice.

The initial screen will appear again. Note that the experimental conditions are not stored in the parameter file. Instead, MacFAUST preserves this information. As a result, you can use the same automaton (i.e., the same parameter file) in different experimental conditions.

III.3 Running the example experiment

Let us try what the experiment would be like. Choose **Experiment** from the **File** menu.

Now you are asked for information that identifies the to-be-recorded data set.

You need to enter the name, age, and sex of your subject and the experimental condition. Note that you *must* enter these data before you can go on. You can also choose whether or not you want the menu to be available during the experiment. You will notice that initially the menu bar is empty. If you choose ◉ **Include menus** they become visible.

Normally you will not select this option for a real experimental situation to avoid that your subjects play with things that have not been made for them. There is, however, an "emergency break" to stop the program even in this situation. Type ⌘ – M to switch on the menu, then terminate as usual. The subject´s data will be saved to the file you specified at the beginning of the experiment.

Click **OK** (you must have entered all the required data!) .

You are then asked to provide a file name for the subject´s data.

By default, this name is composed of the subject´s name plus her/his age, but you may change it to anything you like. This file is a text file so you can edit it with any word processing facility and access it easily with your own software to evaluate experimental data. To learn more about the format of the data in this file, see the "Structure of a subject´s data file" chapter (**IV.8** in this manual).

Click **Save** .

A dialog appears on the screen from which you can choose your matrix.

Click **Matrix01** if "MATRIX01" is the name of you parameter file. Otherwise choose

Different file to bring up the standard file dialog.

The next dialog asks you to wait for MacFAUST to be done with preparing the experiment.

Now you are in the experiment. On the left side of the screen you see the four input buttons, on the other side you see the output field you are already familiar with. At the bottom you find two extra buttons labeled: “Goal state” and “External memory”.

Now you can start to explore the automaton. Click on one of the input buttons and observe the change in the output field on the right side. After a few inputs you may want to have a look at your supports: Let´s first try the **Goal distance** .

A window appears displaying several lines and circles on the screen. The lines symbolize the steps you need to reach the goal; the circles stand for your past interventions, the top most line represents the goal state. You can observe your improvements (and your failures) in your attempts to reach the goal state.

Click on one of the circles to see which state you were in at that point. Then click into the window´s close box in the upper left corner to close it.

Now click on **Past interventions** to try the external memory.

A window appears displaying the last six past system states.

Use the scroll bars at the top of the memory window to see more past states (if there are any).You could now click into the window´s close box in the upper left corner to close it. Let us instead try out the “dynamic“ option of the external memory. Click on one of the miniature versions of the past states to make this particular state the next present state.

You will be asked to confirm your decision.

Click **OK** .

The standard user interface will reappear displaying the selected state.

To try the effects for your second condition simply choose **Quit experiment** from the **File** menu. Your data will be saved in the text file you specified at the beginning of

the experiment. To learn about the format of the data in this file, see the “Structure of a subject´s data file” chapter (**IV.8** in this manual). Choose **Experiment** from the **File** menu and proceed as before except that this time you specify ◉ **Condition 2**. Feel free to try different experimental conditions on your new automaton to find out how they work (you can use this opportunity to play the subject who tries to find out how a complex dynamic environment works…).
Here is an extra piece of information. Whenever you are running an experiment with the menus turned on, all built-in external support systems are available in the **Information** menu (this is one reason as to why you would want to turn off the menus during a real experiment).

Choose **Distance from goal state** to see how far the current state is from the goals state.

Choose **Past states - static** to see the external memory in its “static” version.

Choose **Past states - dynamic** to see the external memory in its “dynamic” version.

Choose **System states** to see how many different states your automaton has. Every state that you have already encountered during your exploration trials is marked by a check symbol (√).

Now quit the experiment (type **⌘-M** if the menu had been turned off in order to turn it back on) and learn more about knowledge tests in the next chapter.

III.4 Adding knowledge tests to your parameter file

In the example experiment we did not get any diagnostic information because we have failed to specify which knowledge tests to include. In this case MacFAUST simply takes some default values to be able to run the experiment. We will now deliberately decide on

- the *number of “blocks”* (cycles of phases with intervention trials and diagnostic phases),
- the *number of intervention trials* for each block,
- the *types of knowledge tests* after the blocks of intervention trials,
- the *order of different knowledge tests* after the blocks of intervention trials,
- the *number of knowledge probes* for each type of knowledge test,
- the *state transitions probed* for in each type of knowledge test.

Choose **Open** from the **File** menu and open your example automaton (“Matrix01” if you followed the example in Part II of this manual).

Choose **No. of intervention trials** from the **Edit** menu.

A window appears that gives you control over the entire course of the experiment with respect to the intervention trials. You may select between 1 and 5 different blocks of intervention trials. Each block may consist of 2 to 200 trials.

Select ☒ **3. block of intervention trials.**

Enter **20** in the editable fields of each of the blocks 1 to 3.

Click **Add** to see the total number of intervention trials (this figure should now be “60”).

Click **Save** .

We are back to the automaton. The logical next step is to select the diagnostic procedures or “knowledge tests” that shall be applied between blocks of intervention trials.

Choose **Knowledge tests** from the **Edit** menu.

A window appears that gives you control over the entire course of the experiment with respect to the knowledge tests.

Select ◉ **Experiment with knowledge tests**.

If you click the alternative option the knowledge tests are not included in an experiment. This option is particularly useful if you have already specified a number of knowledge tests and do not want to loose this information. As soon as you decide to have an experiment with the knowledge tests included the previous knowledge tests are again available. Once specified, this information is never lost.

In principal, you can after each block of intervention trials apply a different set of knowledge tests. However, for the present purpose it will simplify matters extremely if we select

☒ **same kinds of tests each phase**
☒ **same sequence of tests each phase**
☒ **same # of items per test each phase**
☒ **same set of items per test each phase**

You may wonder why it should be useful to have different knowledge tests after each block of intervention trials at all. One situation in which this seems useful is if you want to have a large knowledge test at the end of the experiment and only a few items between blocks to give subjects a chance to practice the procedure. In this case, of course, you cannot control the process of knowledge acquisition.

Click **Continue** .

The next window is already familiar.

Select ☒ **Knowledge tests** for the first two blocks of intervention trials and **Save** the selections.

Note that we have selected three blocks of intervention trials but only two knowledge tests. Thus, after the final block of trials there will be no knowledge tests. If you want to include any extra knowledge tests at the end of the experiment you must also select the knowledge test item for the third block.

MacFAUST then asks you to specify the kinds of items you want to include. There are four different types of knowledge tests, three recognition tasks and one verification task.

In the **recognition tasks**, two elements of a state transition matrix are presented, and the missing element must be selected from a list of alternatives by subjects. The recognition items may be “prognostic” (Type 1: the next state must be recognized), “interpolated” (Type 2: the intervention must be recognized), or “retrognostic” (Type 3: the preceding state must be recognized) in nature.

In the **verification task** an entire state transition is presented, and it must be judged whether or not it is correct for the given automaton. The probed transitions are taken from subjects´ prior intervention trials to guarantee that they have been explored. The transitions may be wrong in two ways: First, the false element(s) of the transition may nevertheless belong to the elements of the automaton (i.e., it is “incorrect”). It is then randomly selected from the remaining labels that indicate a different element. Second, MacFAUST may randomly select the false element(s) from the set of distractor labels (i.e., it is “incorrect with distractor”). In both cases MacFAUST selects the wrong elements for the next state given a previous state and an intervention to make the state transition impossible for the given automaton.

Select ☒ **Recognition 1** and ☒ **Verification test** for our example experiment, then click **OK** .

The next dialog asks you to specify the order in which the different tests will be presented after the blocks of trials.

Click **Position 1** and **Position 2** and notice that the verification test and the recognition 1 test now appear in reversed order.

Click **OK** .

The next dialog asks you to specify the number of items for each test (if, like in our example, you choose to have the same configuration of tests after each block of intervention trials MacFAUST will ask you to specify the number of items for “Phase” 1; otherwise this question is repeated for all phases).

Enter 2 in the editable field for the verification test and click **OK** .

Enter 2 in the editable field for the recognition 1 test and click **OK** .

The next dialog asks you to specify the state transitions that will be probed for in the “Recognition 1” task. We have selected 2 recognition items and these items are displayed in the dialog. The dialog also displays the ”Output” (previous system state) and “Input” (intervention) selected for the state transition of that item (initially, both values are set to “1”).

If you had specified more that 10 items for this test they would have been displayed in more than one set. In this situation, the dialog displays additional button that you must click to get to the next set of items (or back to the previous set) to edit them.

Click ◉ **Item 1** (or simply type in the number of the item you wish to select) and select the system state (“Output signal”) and the intervention (“Input signal”) using the vertical scroll bars. Click on “Output No” or “Input No” to see the verbal labels of a particular state transition (MacFAUST automatically finds the missing element of the state transition; this is true for all types of recognition items). Repeat this with the second item. Click **OK** when you are done with editing these items.

Note that for any type of recognition item—prognostic (”1”), interpolated (“2”), and retrognostic (“3”)—MacFAUST asks you to specify only the previous state (“Output”) and the intervention (“Input”). The program automatically computes the next state from this information. During an experiment, however, MacFAUST will correctly display the two given elements of the transition according to the type of recognition item you selected, and it will correctly ask for the missing element.

The next dialog the asks you to specify the trials from which MacFAUST should select the state transitions for the verification task. In our example, 2 items will be displayed. The dialog also shows the trial which has been selected for each item (initially “1”), and the status of each item (initially “correct”).

Click ◉ **Item 1** (or simply type in the number of the item you wish to select) and select the trial using the scroll bars. Then select one of the following alternatives:
◉ **correct** or

◉ **incorrect** or
◉ **incorrect with distractor**.

Click **OK** . Repeat this for the second item.

If you had specified more that 10 items for this test they would have been displayed in more than one set. In this situation, the dialog displays additional button that you must click to get to the next set of items (or back to the previous set) to edit them.

The automaton will then reappear with the standard user interface. You have now specified the diagnostic items.

Choose **Save** from the **File** menu to append the diagnostic information to your parameter file.

The next time you choose **Knowledge tests** from the **Edit** menu you will be able to edit your diagnostic items. Perhaps you want to try what an experiment with knowledge tests looks like? Choose **Experiment** from the **File** menu and proceed as described in the beginning of the “Running the example experiment” section. All your data—including that concerning the knowledge tests—will be saved to the same text file. To learn about the format of the data in this file, see the “Structure of a subject´s data file” chapter in Part **IV.8** of this manual.

It is easy to imagine that things can get very complicated if you leave

☐ **same kinds of tests each phase**
☐ **same sequence of tests each phase**
☐ **same # of items per test each phase**
☐ **same set of items per test each phase**

all unselected. In this case you will be asked to specify all of these parameters for each of the diagnostic phases following the blocks of intervention trials. **We recommend that you do this only after some experience with specifying knowledge tests** to avoid getting lost.

Part IV: Advanced features of MacFAUST

This chapter assumes that you have already some knowledge about MacFAUST (e.g., you have worked through Parts II and III of this manual). You should be familiar with basic concepts like “input variables“, “output variables”, and the “state transition matrix”. In order to understand the last chapter of the “Structure of a subject´s data file” you should be familiar with the knowledge tests introduced in Part III of the manual.

IV.1 Adding and deleting input signals and changing their positions

Even if you have followed our recommendations and thought carefully about your new automaton you will sometimes be dissatisfied with the results of your work. For instance, considering our example automaton (the ticket vending machine) from Part II of this manual, the idea might cross our minds that it would be nice to have a machine that also accepted 3 ECU coins. The automaton currently has one input variable with four levels. Thus, we want to add a fifth level.

Open your parameter file and choose **Input signals** from the **Edit** menu.

In the “Edit input variables” dialog click **Add variable level**.

The standard user interface appears with the background shaded gray. This is to indicate that you are about to make major changes to your automaton.

Click on one of the automaton´s buttons. The new button will be inserted on the left side of the button you selected. Select the “PULL” button. When it blinks, click **OK** .

The standard user interface appears and the new button has been inserted. By default, all state transitions associated with the new button will be set to “1”. Thus, the state transition matrix will look like this:

	Input signal				
	1	2	3	4	5
States	1 ECU	2 ECU	**NEW**	PULL	EJECT
1: empty	2	3	**1**	1	1
2: one ECU	3	4	**1**	2	1
3: two ECU	4	5	**1**	3	1
4: enough	5	5	**1**	6	1
5: too much	5	5	**1**	5	1
6: *GOAL*	2	3	**1**	1	1

Choose **State transition matrix** from the **Windows** menu to fill in the correct figures (from the top to the bottom of the “NEW” column in the matrix above the correct figures are “4”, “5”, “5”, “5”, “5”, and “4”).

Then click on the input buttons to edit their labels and change these from the default “new” to “3 ECU” for the sake of having a consistent example automaton.

If you wish to try instantaneously how your automaton works after these changes see **IV.5** “Trying the new automaton while editing”.

You can *change the positions of the buttons* on the screen. To achieve this you must

Choose **Input signals** from the **Edit** menu.

In the “Edit input variables” dialog click **Change positions**.

The standard user interface appears with the background shaded gray. This is to indicate that you are about to make major changes to your automaton.

Click on the two buttons which you want to interchange. When they both blink, click **OK** .

A new dialog asks you about the details of the changes to be made.

Select :

◉ **Change functions** to interchange only the way the two buttons work (this is equal to interchanging the columns of the state transition matrices associated with the two buttons).

◉ **Change labels** to interchange only the physical labels of the two buttons (this will leave the columns of the state transition matrices unchanged).

◉ **Change functions and labels** to interchange the two buttons completely.

Click **OK** .

The standard user interface appears and the two buttons have been interchanged.

Note that in order to follow the examples given in this manual you need to revert (or not save) any changes you have made in this section!

You can also *add an entire input variable*. In this case, a subject would have to select two buttons—each button belonging to a different input variable—and then click “OK” (to learn about automata with and without “OK” buttons, see **V.3** “The Windows menu”:“Options”). To add an entire input variable you must

Choose **Input signals** from the **Edit** menu.

In the “Edit input variables” dialog click **Add entire variable** .

The standard user interface appears and one row with two new buttons has been inserted. By default, all state transitions associated with the new buttons will be set to “1”.

Note that in order to follow the examples given in this manual you need to revert (or not save) any changes you have made in this section!

You may also want to *remove an input variable* from your automaton. This is done in a stepwise manner. You actually delete only levels of input variables. However, if an input variable has only two levels removing the second of these two levels will imply that the entire variable will be removed (this should seem plausible since an automaton with only one input button would not make much sense; it would have to be pressed at each intervention).

Choose **Input signals** from the **Edit** menu.

In the "Edit input variables" dialog click **Remove variable level**.

The standard user interface appears with the background shaded gray. This is to indicate that you are about to make major changes to your automaton.

Click on the button you want to remove. When it blinks, click **OK** .

The standard user interface appears and the selected button has been removed. If the input variable had only two levels, the entire variable has been removed.

Note that in order to follow the examples given in this manual you need to revert (or not save) any changes you have made in this section!

IV.2 Adding and deleting output signals

As mentioned in the previous chapter, you will sometimes be dissatisfied with an automaton you created. For instance, considering the modification that you did to your example automaton in the previous chapter you will realize that a machine that accepts "3 ECU" coins is quite uninteresting as long as the price for the ticket is only 3 ECU—two clicks and you have it! Therefore, we will assume that we had a price increase lately, and the cost for a ticket has gone up to 4 ECU (this is what we called "enough"). The automaton currently has one output variable with six levels. Thus, we want to add a seventh level.

Choose **Output signals** from the **Edit** menu.

In the "Edit output variables" dialog click **Add variable level**.

<u>If you have more that one output variable</u>, a dialog will ask you which variable you wish to edit. Click on the hand that points to the variable you wish to edit.

A window appears that is similar to the one you see when you edit the labels of your output signals but this time the background is shaded gray. This is to indicate that you are about to make major changes to your automaton.

Click on one of the bar icons for the output signals. The new output signal will be inserted below the bar icon you selected. Select the "2 ECU" bar icon (it will blink). Click **OK** .

If you click "OK" with no bar icon selected this will leave the automaton unchanged.

The standard user interface appears and the new signal has been inserted. You cannot see it now unless you choose “Output signals” from the “Windows” menu. By default, all state transitions associated with the new signal will be set to “1” and MacFAUST compensates for the shifts in your matrix by increasing the figures in all relevant other cells by the appropriate amount. Thus, the expanded state transition matrix of the ticket vending machine will look like this:

	Input signal				
	1	2	3	4	5
States	1 ECU	2 ECU	NEW	PULL	EJECT
1: empty	2	3	5	1	1
2: one ECU	3	4	6	2	1
3: two ECU	5	5	6	3	1
4: NEW	**1**	**1**	**1**	**1**	**1**
5: enough	6	6	6	7	1
6: too much	6	6	6	6	1
7: *GOAL*	2	3	5	1	1

Note, however, that although MacFAUST tries to compensate for the shifts in your matrix by increasing the figures in all relevant other cells the result is not a correctly working automaton.
For instance, the cell marked by “two ECU”/“1 ECU” contains “5”. This is so because at state “two ECU” inserting “1 ECU” was “enough” in the previous version of the automaton. For the new version the correct cell entry would, of course, be “4”.
MacFAUST is blind to the subtleties and you have to fix these parts of your state transition matrix manually (if your matrix is large, see the “Editing very large state transition matrices” chapter in this part [IV.6] of the manual). Therefore you should now:

Choose **State transition matrix** from the **Windows** menu to fill in the correct figures.
If you wish to try instantaneously how your automaton works after these changes see **IV.5** “Trying the new automaton while editing”.

You can also *add an entire output variable*. In this case, your automaton would have two different output fields—each field belonging to a different output variable. To add an entire output variable you must:

Choose **Output signals** from the **Edit** menu.

In the “Edit output variables” dialog click **Add entire variable** .

The standard user interface appears and the field of the new output variable has been inserted. This new variable has two levels. By default, all state transitions associated with the new variable will be set to “1”. Also, note that although MacFAUST will try to take into account the changes done to your matrix it will most certainly be necessary to fix some problems in the matrix (see the example given in the previous section on “Adding an output variable level”).
Note that adding a new “two–level” output variable will double the number of output states of your automaton since multiple output variables are “nested” and the combinations of their levels form the set of possible output signals.

You may also want to *remove an output variable* from your automaton. This is done in a stepwise manner. You actually delete only levels of output variables. However, if an output variable has only two levels, removing the second of these two levels will imply that the entire variable will be removed (this should seem plausible since an automaton with only one output signal would not make much sense: it would always display the same thing; the reasoning here is parallel to the situation with input variables).

Choose **Output signals** from the **Edit** menu.

In the “Edit input variables” dialog click **Remove variable level**.

If you have more that one output variable, a dialog will ask you which variable you wish to edit. Click on the hand that points to the variable you wish to edit.

A window appears that is similar to the one you see when you edit the labels of your output signals but this time the background is shaded gray. This is to indicate that you are about to make major changes to your automaton.

Click on the bar icon for the output signal you wish to remove (it will blink). Click **OK** .

The standard user interface appears and the selected output variable level has been removed. If the output variable had only two levels, the entire variable has been removed and the corresponding output field will be missing.
As with adding output variables, removing them will cause cell entries (numbers indicating the “next” states) in the state transition matrix to point to the wrong states of the system. MacFAUST tries to compute for each cell entry the correct new reference by taking into account the states that have been removed. However, some entries inevitably will have a reference to states that will have been removed. In this case, the cell entry is set to “1” (which often times will be the initial state of systems), and you will have to fix your matrix manually.

IV.3 Latent states: Adding an invisible output variable

Sometimes you may want to have state transitions that are invisible to the user. In other words, the state does change, but the signal does not. Of course, you can do this by assigning identical labels to output signals for different states. This method will, however, work only for small scale manipulations. In addition, editing such a matrix can become quite confusing. For more “drastic” manipulations and more convenience for the experimenter we added the option to make an entire variable invisible. For instance, assume you have the following state transition matrix:

Output				Input	
Variable 1	**Variable 2**	**Signal**	**States**	1	2
A	1	A1	1	1	4
A	2	A2	2	2	5
A	3	A3	3	3	6
B	1	B1	4	3	6
B	2	B2	5	2	5
B	3	B3	6	1	4

We assume that we have two output variables with three levels each and one input variable with two levels. The output signal is composed of the labels of the two output variables, and each state 1..6 is identified by a unique output signal. However, if we make the Variable 1 invisible (i.e., the labels of this variable no longer appear on the screen), then things get very complicated. We recommend that you try the example automaton presented here (this is worth more than reading two pages of manual prose). Make sure that Variable 1 appears in the top field of the output variables. After you have created the automaton as usual do the following:

Choose **Options** from the **Windows** menu.

In the next dialog select ◉ **Include latent state** to make Variable 1 the latent variable (MacFAUST always selects the variable appearing in the top output field). Click **Save** .

The standard user interface appears and the top most output variable now appears white instead of gray. This is to indicate that it will not be visible in an actual experiment. However, it is still visible for you both during editing and when you try the automaton (see the “Trying the new automaton while editing” chapter [IV.5] in this part of the manual).

IV.4 Process control: Adding time effects

In Parts II and III we have dealt with automata that produced state transitions only after an input button had been hit. In other words, these automata were completely user dependent. In reality, however, there are ample examples for machines that are not so patient. Some ticket vendors, for instance, eject your money if you did not inset it within a specific time interval. If such processes operate on a large scale and have to be taken into account by a person interacting with the system (e.g., in industrial plants) we usually call that process control.

Technically, adding time effects can be thought of as adding an additional input (i.e., an additional column) to the state transition matrix that determines which state will be the next system state if the conditions of the time effect are true. MacFAUST actually adds three columns to the state transition matrix, the first of which contains as entries the next system state given a particular previous state and the time effect conditions being true. The next two columns contain parameters that serve the program to identify which states include time effects and what exactly the condition for a particular time effect is. (You should remember this structure when you edit a state transition matrix as described in **IV.6** "Editing very large state transition matrices".)
MacFAUST offers a selection of three conditions to define time effects.

① The most simple case is that after a fixed time interval the system shifts to a defined state. The time interval starts after the system has shifted from its initial state to a different state. This would, for instance, be the case for one of these modern ticket vendors that automatically initiate a reset to the initial state if the user has not inserted the money within a certain time interval.

② It is possible to construct a system that will shift to a defined state *always* after a fixed time interval. In contrast to the first case this system never "rests". Even once it has reached the defined state it will produce shifts to that state whenever the next time interval is over, thus "using up" the interventions a person could use to move the system to the goal state.

③ The third and most complicated option is to define for each state separately whether there should be a next state for the system to shift to after a specific time interval, and what the length of the interval should be. It will now be described how the options can be handled in MacFAUST.

Choose **Time effects** from the **Edit** menu.

A dialog appears that gives you control over the initial settings of your time effect manipulations. The changes in this dialog will affect all states of the automaton.
Notice that the default option for a new automaton is that the "State transitions depend on user´s inputs" option is selected. This means that the automaton does not shift to any states simply as a function of a time interval. To include time effects you must:

Select
◉ **State transitions depend on users input and time interval**. To implement one of the options ① to ③ follow the appropriate sequence of steps:

① Select ☒ **Start interval independent of user´s input**. Then select ◉ **Start count-down at the initial state**. Next determine how long the interval shall be and what state to shift to by entering in the editable fields the appropriate number of seconds for the time interval and the ordinal number of the state the system will move to when the interval is over. For instance, if you want the ticket vendor from the example in Part II of this manual to *shift to State 1 exactly 5 seconds after users have made their first input* enter

5	**sec**
1	**No. of the new state**

Click **Label of new state** to see the output signal of the new state (in our example this will be "empty").
Click **OK** .

② Select ☒ **Start interval independent of user's input**. Then select ◉ **Start count-down successive**. Next determine how long the interval shall be and what state to shift to by entering in the editable fields the appropriate number of seconds for the time interval and the ordinal number of the state the system will move to when the interval is over. For instance, if you want the ticket vendor from the example in Part II of this manual to *shift to State 1 every 3 seconds* (unless users have reached the goal state) even if the system already is in State 1, enter

3	**sec**
1	**No. of the new state**

Click **Label of new state** to see the output signal of the new state (in our example this will be "empty").
Click **OK** .

③ Select ☒ **Start interval dependent on user's input**. Then click **Continue** .

In the next dialog you can choose a number of parameters that will affect all cells involved in time effects. This feature is particularly useful if you have in mind that time effects should be homogeneous for most states but that there should be some exceptions.
However, if you have already edited time effects in your automaton be aware that saving settings from this dialog will override all you previous time effect parameters. If this would imply that you lost a lot of work you must click **Skip** to get directly to the next dialog.

Depending on the characteristics of your automaton first select the option for the next state after the time interval. This may be

◉ **defined state**	(enter the state number in the editable field that appears when you select this option).
◉ **initial state**	(regardless of which initial state you will select later).

◉ **transition to the same state** (in other words, the system will remain in that state but "use up" an intervention trial).

If most of your states will *not* be involved in state transitions that depend on time intervals select ☒ **Initially no dependency on time**. This is selected by default. If most of your states *are* involved in time effects, *de*select it.

Finally, enter in the editable **Interval** field the number of seconds for the standard time interval. The default value is $\boxed{\text{1 0}}$ **sec**.

Click **Save** to save these parameters as the standard values for your time effects. However, you may also

click **Skip** to avoid saving them. You will always want to skip if you want to avoid that your previous editing of time effects are cancelled and overridden.

In any case, a new dialog will then display for every state the current (standard or previous) settings of the time effects parameters. These are

- whether or not a state is involved in time effects;
- the number of the next state given the time interval is over;
- the length of the interval after which the shift will occur.

To edit these parameters simply click on the bar icon of the particular state you wish to edit. In the dialog that appears you may change the settings for that state.

After you have completed the setting of time effect parameters click **OK** .

Finally, the standard user interface appears again. Remember that initially the default option for the automaton was that the "State transitions depend on user´s inputs" option was selected. This means that the automaton would not shift to any states simply as a function of a time interval. If you later decide that your automaton should not include time effects, simply select this option again. Note, however, that the parameters you have set for your time effects will not be lost. Whenever you select the time effects option ("State transitions depend on users inputs and time interval") again, your old parameter settings are in effect again. Likewise you should remember this when you edit the state transition matrix as described in the "Editing very large state transition matrices" chapter [IV.6]: the last three columns in this matrix will still contain the time parameter settings.

IV.5 Trying the new automaton while editing

Very frequently you want to control whether your changes to the automaton had the intended effects. To make this as convenient as possible we added an option to try out an automaton while editing.

Choose **Try OUTomaton** from the **Specials** menu.

The standard file dialog appears and asks you to save your changes to a parameter file. The standard file name offered is the original name of your automaton´s parameter file. If you are not quite sure that your last changes were good and you do not want to loose your previous work simply save the data to a new automaton.

Type in the new name if necessary and click **Save** .

Before it will save the data MacFAUST must compute a few additional figures necessary for actually “running” the automaton (such as the distances of each state from the goal state). Some of these computations may fail because there are **problems with the state transition matrix**.
For instance, one frequent cause of a failure is that the goal state cannot be reached from any of the other states. It is necessary that there is at least one sequence of states and interventions from the initial state to the goal state. Another frequent problem is that there is a “dead end state” (i.e., an absorbing state from which only transitions to the state itself but not to another state are possible). The same situation may, of course, also be true if two or more states form a “dead end loop” from which there is no escape.

MacFAUST gives you feedback about potential causes of the problem and also about the locations of problematic cell entries. A reference is made to the states that are involved in the problem. It is probably a good idea to copy the information from the messages you receive, save the data to a different file, and then try to remove the bugs in your state transition matrix. In locating the problem and fixing it you must keep in mind that “states” of the automaton correspond to the lines in your state transition matrix.

In case of no problem in computing the figures for your automaton you can now try it out and see if it works as you want it to. The “Try OUTomaton”—menu will be marked by a check symbol (“√”).

We recommend that you make use of the external support systems MacFAUST has been equipped with originally for experimental purposes. If you try out your automaton this always occurs under the current settings of Condition 1 (see the “Setting experimental conditions in MacFAUST” in Part III.2 of this manual).

Finally, if you want to return to editing the automaton

choose **√Try OUTomaton** from the **Specials** menu to *un*select the option.

IV.6 Editing very large state transition matrices

When you have very large state transitions, editing them in the MacFAUST environment may be somewhat tedious and uneconomical. This is particularly so if your matrix contains areas with redundant entries (e.g., transitions to the same state in an entire row) or if sub-parts of the automaton can be derived from each other by a simple algorithm. In this case it is more efficient to do most of the matrix editing in a commercial word processing or—even better—spreadsheet application. For this purpose it is possible to “export” state transition matrices as *tab-delimited text files*. In principle, these text files look like the state transition matrices used as examples throughout this manual. All elements (the state and input labels as well as the cell entries) are separated by tabs. You can now make changes to the cell entries in your matrix (not to the labels, however, since with the current version of MacFAUST they cannot be re–imported). After you have made your changes you can “import” the data again and save them to your parameter file. This section explains the necessary steps. Also, there are a few other things that can be “exported”:

Assuming you have already opened an automaton you can choose **Export matrix** from the **Specials** menu.

In the dialog that appears you may select one of the following options:

◉ **Only the labels of the input and output variables.** This may be useful if you need them for purposes of documentation. Labels cannot be re-imported.

◉ **Knowledge test items.** This may be useful if you need them for purposes of documentation.

◉ **Distance from goal state.** This information can help you to design an automaton with specific properties that depend on that data, and to design control tasks. Goal distance information is always computed when you save your automaton so there is no point in re-importing it).

◉ **Complete state transition matrix.** This is the option you will probably make use of most frequently. Your state transition matrix will be exported as tab–delimited text such that standard word processors and spreadsheet applications can read it.

Click **OK** .

The standard file dialog appears and asks you to save your data in a text file. The standard file name offered is the original name of your automaton´s file plus a specific suffix. This suffix is

-LABELS for a file containing input and output variable labels;
-TESTS for a file containing your knowledge test items;

-**DISTANCE** for a file containing goal distance information;
-**STM** for a file containing a complete state transition matrix.

When you edit a state transition matrix make sure that you change the cell entries only, and not the labels of the variables. These cannot be re-imported with the current version of MacFAUST but changes may affect the importing process in such a way that you will no longer be able to import your matrix! Also, you must not change the size of your state transition matrix (i.e, do not add columns and/or rows). If the data in the text file do not fit your automaton MacFAUST will reject the file!
In order to *import* a matrix first make sure that you have the proper automaton loaded.

Choose **Import matrix** from the **Specials** menu

The standard file dialog appears and asks you for a text file.

Select the file you want to import (if you followed the conventions we suggested this will be a file named according to your parameter file plus the suffix "STM") and click **OK** .

MacFAUST will compare the data of the text file to the parameters of the currently loaded automaton. If the data in the text file do not fit your automaton MacFAUST will reject the file!
Otherwise it will read in the state transition matrix (not the labels, however) and update the cell entries accordingly. You should then save your file to check if your matrix does not contain any formal errors.
If MacFAUST finds an out–of–range error (i.e., a cell entry larger than the number of states of your automaton) it warns you and displays the critical cell entry in a dialog together with information about the location of the entry (row and column) so you can change it.

IV.7 Computing a complexity index for your automaton

MacFAUST offers you to determine a complexity index to assess the decisional structure which the state transition matrix presents to the "naive" subject. By the term "naive" we have two things in mind. First, if you model a device that is more or less known to your subjects they can usually very quickly reduce the complexity of the device by applying prior knowledge to check which concepts they suspect are really present and in what form. Second, even if you use devices that do not occur in real world, subjects will develop concepts to group similar state transitions (e.g., the effects of an »on/off« switch). Thus, the complexity index is of use only to the degree to which the automaton is "completely new" to your subjects. Nevertheless we think the complexity index is a useful indicator particularly for comparing different automata on formal grounds.

Choose **Complexity** from the **Specials** menu.

MacFAUST presents a dialog containing the complexity information. This information is computed according to a measure that has originally been suggested by McCabe (1976) to assess the decisional complexity of computer programs. It takes into account the number of states a device can be in, and the number of interventions with different consequences possible for a given system state. This complexity measure may be used to characterize the overall complexity of automata. Considering a finite state automaton A with n states, e different transitions given each state and summed over all states and p connected components complexity C is defined as

$$C(A) = e - n + 2 * p.$$

Since p is different from 1 only for hierarchically nested automata (a case which is irrelevant for our present purpose) we can say that for the ticket vendor example in Part II $n = 6$ and $e = 19$, resulting in $C = 19 - 6 + 2 * 1 = 13$. This figure may then be used to make ordinal comparisons between different automata.

IV.8 Structure of a subject´s data file

In an experiment all intervention data and all data from the knowledge tests are written to a text file. The general structure of that data file is outlined in the figure below. Note, however, that the format of the columns varies as a function of the automaton´s properties. In particular, the number of input variables (refered to as "Components of input signal") determines how many columns are needed to code each component, beginning at column 3.
Data having to do with the external support systems are coded in the columns left of the "Output signal" column. By default, some columns are provided for each data point relevant to an external support system regardless of whether that support has been present in the experiment or not. In the latter case, the column is filled with "0"s. This way of handling the data logging is, of course, time and (storage) space consuming. However, it has the advantage of leaving the data structure relatively simple (i.e., it does not vary between different experimental conditions). Time is coded in whole seconds for data from the intervention trials (note, however, that the reaction times for the verification task are measured and coded in milliseconds).

Trial	Previous output signal	Components of input signal (multiple input variables)	Input signal	Output signal	N of different next states	Distance from goal state	State transition using the "Dynamic Memory"	Time used to make an intervention	Time spent using the "External Memory"	N of calls to "Distance from Goalstate"	Time spent using "Distance from Goalstate"
1	1	1...	1	1	0	0	3	0	1	0	0
2	1	0...	0	2	0	0	5	0	1	0	0
3	2	2...	3	1	0	0	2	0	1	0	0
"	"	"...	"	"	"	"	"	"	"	"	"
"	"	"...	"	"	"	"	"	"	"	"	"
9	1	1...	1	1	0	0	2	0	1	0	0

The "table" of intervention data has as many rows as there are intervention trials (only 9 in the present example). One special feature is the "0" in the "Input signal" column in row 2. Obviously there cannot be a "normal" No. 0 input. We have chosen to use this figure to indicate state transitions that have been initiated as a function of a time interval and not as a consequence of a user input (➞ **IV.4** "Process control: Adding time effects").
The "table" of knowledge test data has as many rows as there are knowledge test items (summed over all types of tests and all test phases). Time is coded in whole seconds for data from the recognition tasks. However, the reaction times for the verification task are measured and coded in milliseconds. MacFAUST uses "Drexel´s millitimer" (Westall, Perkey, & Chute, 1989) for measuring keystroke–based reaction times. This, finally, is what the data from a fictitious diagnostic phase with different types of knowledge tests would look like:

Kind of knowledge test	Previous output signal	Input signal	Output signal	User´s response	Evaluation of user´s response	User´s Response: Output signal	Item status	Confidence ratings	No. of presentations	N of states which can be reached	N of states this state can be reached from	Time used for answering	Components of previous output signal	Components of output signal	Components of input signal	Components of selected output signal
1	1	1	2	0	0	0	0	0	0	2	2	1	1...	1...	1...	1...
1	4	2	3	0	0	0	0	0	0	2	3	0	2...	2...	4...	2...
1	8	5	9	0	0	0	0	0	0	2	2	1	1...	1...	1...	3...
"	"	"	"	"	"	"	"	"	"	"	"	"	"...	"...	"...	"...
"	"	"	"	"	"	"	"	"	"	"	"	"	"...	"...	"...	"...
4	1	1	2	1	1	0	1	0	0	2	2	1	1...	1...	1...	1...

- “Kinds of knowledge test”: “1” to “3” = Recognition 1 task to Recognition 3 task, respectively; “4” = verification task.

- For Recognition 1 to Recognition 3 tasks “User´s response” contains the number of errors made in selecting the proper components for the missing element of the state transition.

- For the verification task “User´s response” contains the answer: “1”=“Yes”, “2”=“No”.

- For Recognition 1 to Recognition 3 tasks “Evaluation of user´s response” contains the number of selected distractors.

- For the verification task “Evaluation of user´s response” contains the evaluation of the answer: “1” = “Hit”, “2” = “False Alarm”, “3” = “Correct Rejection”, “4” = “Miss”.

- “Item status” is always “0” in the recognition tasks, and in the verification task “1” indicates a false item.

- “No. of presentations” is relevant for the verification tasks only. It contains the number of times an item was repeated during the test because of too fast (< 100 ms) or too slow (> 15000) reaction times or because a wrong key had been hit.

- “Time used for answering” is measured in seconds for Recognition 1 to Recognition 3 tasks, and in milliseconds for the verification task.

- All other columns contain item information (i.e., they characterize the probed state transition) similar to the information recorded during the intervention trials.

Part V: Reference to menu commands and options

V.1 The File menu

New (⌘-N)

Creates a new automaton (➟ **II.2** “Planning an example automaton”, and **II.3** “Creating a new automaton”).

Open (⌘-O)

Opens an existing automaton for editing. The data must be in a MacFAUST parameter file. (To read state transition matrices from text files ➟ **IV.6** “Editing very large state transition matrices”).

Save (⌘-S)

Saves the parameters of an automaton to a parameter file (➟ **II.3** “Creating a new automaton”, and **IV.5** “Trying the new automaton while editing”).

Experiment (⌘-E)

Starts and controls an experiment (➟ **III.1** “Planning an example experiment”, **III.3** “Running the example experiment”, and **III.4** “Adding knowledge tests to your parameter file”).

Experimental conditions

Gives control over the number of different experimental conditions, and the features of each condition (➟ **III.2** “Setting experimental conditions in MacFAUST”). This information in not stored in a parameter file but rather in MacFAUST itself (in its “resource fork”). This way it is possible to run different experimental conditions with the same automaton (=parameter file).

Quit (⌘-Q)

Terminates the program both during editing and during an experiment. If, during an experimental session, the menu has been switched off and you decide to terminate the experiment, type ⌘–M to turn back on the menu and then quit as usual. All data will be saved in the text file you specified at the beginning of the experiment.

V.2 The Edit menu

Input signals (⌘-I)

Gives the option to add, delete, or change the position of input buttons. Also, entire input variables may be added or removed (➟ **IV.1** "Adding and deleting input signals and changing their positions"). In an automaton with more than one input variable the user must select one button from each input variable and then click "OK" before a state transition occurs. In general, automata with more than one input button should also have an "OK" button whereas others may have but do not need one (➟ **V.1** "The **Windows** menu": "Options").

Output signals (⌘-U)

Gives the option to add or delete output signals. Also, entire output variables may be added or removed (➟ **IV.2** "Adding and deleting output signals"). In an automaton with more than one output variable the output signal is composed of two or more elements, some of which may or may not change after a transition.

Time effects (⌘-T)

Gives the option to add state transitions that do not or do not exclusively occur as a function of a user´s intervention, but also as a consequence of a defined time interval. This adds the dynamics of time to the task of exploring a device, and actually makes it a process control task (➟ IV.4 "Process control: Adding time effects").

No. of intervention trials (⌘-L)

Gives the option to determine for an experiment the number of blocks of intervention trials (ranging from 1 to 5) and the number of trials for each block (ranging from 2 to 200) that are available to a subject for exploring the automaton (➟ **III.4** "Adding knowledge tests to your parameter file").

Knowledge tests (⌘-D)

Gives the option to determine for an experiment whether a block of intervention trials will be followed by knowledge tests, the types of knowledge tests (three different types of a recognition task and a verification task), the order of different types of knowledge tests, the number of items of each type knowledge test (ranging from 1 to 20) and the state transitions used as knowledge items. (➟ **III.4** "Adding knowledge tests to your parameter file").

Display options

Gives the option to display either verbal labels (➟ **V.1** "The Windows menu": "Input signals" and "Output signals") or use a graphical display. The graphical display may be particularly interesting if the automaton involves numerically ordered output signals.

Tutoring (⌘-G)

Allows to define a set of interventions and state transitions that are displayed to subjects without giving them the possibility to make an active intervention. In contrast to "learning-by-doing" these subjects must learn by observation.
Use the horizontal scroll bars to select the "current" and "next" states for the transitions to be displayed (this works similar to the way you edit the knowledge test items). Use the vertical

scroll bar to add transitions (by clicking in the arrow pointing downwards) and to scroll through the list of defined state transitions.

V.3 The Windows menu

Input labels (⌘-1)

Returns to the standard user interface if you are in a different window. Click into the input buttons to edit their labels (➟ **II.3** “Creating a new automaton”).

Output labels (⌘-2)

Displays the window containing the icons for the output labels (clicking into the output field has the same effect). Click into the output label icons to edit their labels (➟ **II.3** “Creating a new automaton”). Also, each output variable (i.e., each field in the output display) has a verbal label that you can edit. Click in the gray part of the output field to bring up the dialog for editing.

State transition matrix (⌘-3)

Displays the window containing the bar icons for the input and output signals. Click into an output signal bar icon and an input signal bar icon to invoke a dialog that asks you to enter the “next state” for that state transition (➟ **II.3** “Creating a new automaton”).

Default values (⌘-4)

Gives you control over the initial state and the goal state of the automaton. By default, an automaton has only one goal state. For special control tasks, however, you may want your subjects to first start from an initial state SI_1 and reach a certain goal state SG_1, then move from a specific SI_2 to SG_2, and so on. To create a matrix which has this control task implemented, choose the number of different SI–SG pairs you need (between 1 and 5), and specify by entering the numbers of the appropriate states the sequences of state transitions that will be required.

Options (⌘-5)

Gives you the option to control *for an experiment* whether or not the following things occur:

- The automaton does or does not have an “OK”–button to confirm an input (in devices without an “OK”–button each click into an input signal has an immediate effect). Note that during editing, the “OK”–button is present because it is needed to get your confirmation on certain changes to the system parameters. However, if you try your automaton during editing (➟ **IV.5** “Trying the new automaton while editing”), the “OK”–button will disappear temporarily if no “OK”–button has been selected for the experiment.
  The default is no “OK”–button if you selected “Buttons of all lines belong to the same input variable”, whereas automata with multiple input variables must have an “OK”–button (➟ **II.3** “Creating a new automaton”).

- The knowledge test items (➟ **III.4** “Adding knowledge tests to your parameter file”) include for the “next” states of a probed state transition such state labels that have not actually been present in the automaton but were added during editing (➟ **II.3** “Creating a new automaton”). If this shall be the case select “Include distractors” in this dialog.
- The top most output variable is not displayed to subjects. This option is used to create “latent states” (➟ **IV.3** “Latent states: Adding an invisible output variable”).

V.4 The Specials menu

Complexity (⌘-K)
Computes a complexity index for your automaton (➟ **IV.7** “Computing a complexity index for your automaton”).

About parameter file (⌘-A)
Gives you the possibility to enter information that will be saved with the automaton such as the name of the person who designed the automaton, the date, and some additional comments that may be useful if you are working with different versions of an automaton. You can always look up this information during editing. Also, every time you save your automaton you will be asked if you would like to make changes to this information (➟ **II.3** “Creating a new automaton”). Click “OK” to save changes, but click “Skip” to go on and ignore changes.

Export matrix (⌘-P)
Gives you the possibility to export your state transition matrix as tab–delimited text so you can edit it in a standard word processing of spreadsheet application (➟ **IV.6** “Editing very large state transition matrices”). In addition, you may export the labels of your input and output signals, knowledge test items, and the goal distance information of your system states.

Import matrix (⌘-H)
Gives you the possibility to import a state transition matrix that you previously exported as tab–delimited text (➟ **IV.6** “Editing very large state transition matrices”).

Try OUTomaton (⌘-R)
Gives you the possibility to try out how your automaton works while you are editing it (i.e., without having to start an experiment; ➟ **IV.5** “Trying the new automaton while editing”).

References

Albert, J., & Ottmann, T. (1983). *Automaten, Sprachen und Maschinen für Anwender*. Mannheim: B.I.-Verlag.

Buchner, A., & Funke, J. (1990). Discrete systems as tools for studying knowledge acquisition and knowledge use in dynamic task environments. *Berichte aus dem Psychologischen Institut der Universität Bonn, 16* (4).

Buchner, A., & Funke, J. (1991). Transfer of associations in finite state automata. *Berichte aus dem Psychologischen Institut der Universität Bonn, 17* (2).

Buchner, A. & Funke, J. (1993). Finite state automata: Dynamic task environments in problem solving research. *Quarterly Journal of Experimental Psychology*, *46A*, 83-118.

Buchner, A., Funke, J., Schmitt, L., & Nikelowski, B. (1990). External support systems for a dynamic task environment: Results from two experiments on memory and evaluation aids. *Berichte aus dem Psychologischen Institut der Universität Bonn, 16* (5).

Funke, J. (1991). Solving complex problems: Human identification and control of complex systems. In R.J. Sternberg & P.A. Frensch (Eds.), *Complex problem solving: Principles and mechanisms* (pp. 185-222). Hillsdale, N.J.: Lawrence Erlbaum.

Funke, J., & Buchner, A. (1992). Finite Automaten als Instrumente für die Analyse von wissensgeleiteten Problemlöseprozessen: Vorstellung eines neuen Untersuchungsparadigmas. *Sprache & Kognition*, *11*, 27-37.

Hopcroft, J.E., & Ullmann, S.D. (1979). *Introduction to automata theory, languages, and computation*. Reading, MA: Addison-Wesley.

McCabe, T.J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, *SE-2*, 308-320.

Salomaa, A. (Ed.). (1985). *Computation and automata.* Cambridge, MA: Cambridge University Press.

Westall, R.F., Perkey, N., & Chute, D.L. (1989). Millisecond timing on the Apple Macintosh. Updating Drexel´s millitimer. *Behavior Research Methods, Instruments, & Computers*, *21*, 540-547.

Author´s address

Dr. Axel Buchner is now at the Department of Psychology, Trier University, D-54286 Trier. E-Mail: buchner@cogpsy.uni-trier.de

Dr. Joachim Funke is now at the Department of Psychology, Heidelberg University, Hauptstr. 47-51, D-69117 Heidelberg. E-Mail: joachim.funke@urz.uni-heidelberg.de

Berichte aus dem Psychologischen Institut der Universität Bonn

Die "Berichte aus dem Psychologischen Institut der Universität Bonn" (ISSN 0931-024X) gibt es seit 1975. Die ersten vier Jahrgänge bestehen aus 21 fortlaufend numerierten Heften. Ab Jahrgang 5 (1979) beginnt die Heftzählung in jedem Jahr bei Heft 1. Eine Übersicht über die zuletzt publizierten Hefte gibt folgende Liste.