Operations on Finite Automata

Word recognition

Please take the time to read this document carefully. In particular, it contains certain instructions that you must absolutely comply with. Any failure to do so will inevitably influence your grade.

Further information (clarification, additional information) may be provided at a later date.

Program to be developed

Your program is divided into several stages:

- reading a FA (in a file), storing it in memory and displaying it on the screen;

- if the automaton is not complete deterministic, obtaining the equivalent complete deterministic FA;

- calculating the equivalent minimal FA;

- creating an automaton recognizing the complementary language and testing word recognition on that automaton;

- standardizing the complementary FA and word recognition test on the standardized FA.

Reading a FA from a file

Some time before the presentations, we will send you the test automata. You will have to transcribe them into".txt" files. These files should be named as follows:

      L2Int-<team number>-#.txt

where the # will be replaced by the number of the test FA. For example, if you are the team n° 10 and you are dealing with test FA number 8, the file must be named L2Int-10-8.txt.

To begin with, your program must read the description of a FA in a text file, and save it in memory according to your choice of data structures.

During the execution of your program, the user will choose which test FA he or she wants to study. It must be possible to make this choice in a very simple way: for example, to choose the test FA n°8, it should be sufficient to type "8" in answer to the question "which FA do you want to use?"

As long as you have not yet received the test automata (which will be given to you in a graphic form!), you are free to work on any automaton of your invention, encoding it in a .txt file according to the model you will see below.

Entering a FA from the keyboard (without a file) is forbidden.It is also forbidden to "hard-define" a FA in the program.

The representation of a FA in memory will depend on your choice of data structures. You are free to choose the data structures that you think are most appropriate.

Displaying FA on screen

Displaying the FA stored in memory, with an explicit indication:

        - of the initial state(s);

        - of the terminal state(s) ;

        - of the transition table.

The steps in reading and display can be represented by the following pseudo code:

```
FA ← read_automaton_from file(filename)
display_automaton(FA)
```

Determinization and completion

Let FA be the automaton obtained (and displayed) at the previous stage. The operations can be described by the following pseudo code:

```
IF is_an_asynchronous_automaton (FA)
THEN
      CDFA←determinization_and_completion_of_asynchronous_automaton(FA)
ELSE
      IF is_deterministic (FA)
      THEN
            IF is_complete(FA)
            THEN
                  CDFA←FA
            ELSE
                  CDFA←completion(FA)
            ENDIF
      ELSE
            CDFA←
            determinization_and_completion_of_synchronous_automaton(FA)
      ENDIF
ENDIF
display_complete_dererministic_automaton(CDFA)
```

where:

`is_an_asynchronous_automaton (FA)`

Check whether it is an asynchronous FA (contains "*" transitions).

The result of the test is displayed. If the FA is asynchronous, your program must also tell the user which elements make it asynchronous.

`determinization_and_completion_of_asynchronous_automaton(FA`

Construction of a synchronous, deterministic and complete finite automaton from an initial asynchronous automaton (FA).

`is_deterministic` **(FA)**

Check whether the synchronous automaton FA is deterministic.

The result of the test is displayed on screen. If the FA is not deterministic, your program must tell the user why.

`is_ complete(FA)`

Check whether the synchronous and deterministic automaton FA is complete.

The result must figure on screen. If the automaton is not complete, your program must tell the user why.

`completion(FA)`

Construction of a complete deterministic automaton from a deterministic automaton FA if the latter is not complete.

`determinisation_et_completion_automate_synchrone(FA)`

Construction of a complete deterministic automaton from a synchronous non-deterministic automaton FA.

Here, you should use the algorithm of determinization given in lectures and worked in problem sessions. You should absolutely **NOT** standardize the automaton before the determinizaion.

`display_complete_dererministic_automaton(CDFA)`

Display the automaton on screen using the same format as the one you used for displaying the initial automaton.

Besides displaying the complete deterministic automaton (CDFA), your program must explicitly show the composition of the CDFA states in terms of the states of the original automaton.

Remark: it is preferable to directly use the way of labeling the states of the CDFA as sets of the original states, like "123" for a state composed of 1, 2, and 3. For numbers greater than 9, it is important to make a difference between, say, {1,2,3} and {12,3}. You can implement that by, for example, using a separator of your choice, like "1.2.3" and "12.3".

Warning:

The various tests figuring in the pseudo code are obligatory. For example, you can launch determinization only if the FA has been identified as not being deterministic. Deterlminization of DFA will be considered as an error.

Minimization

There are no tests to do. Your program will minimize the CDFA obtained above and display the result. If the algorithm finds out that the automaton was already minimal, your program must tell the user about it.

Pseudo code:

```
MCDFA ← minimization(CDFA)
display_minimal_automaton(MCDFA)
```

where:

`minimization(CDFA)`

Constructing a synchronous, deterministic, complete and minimal finite automaton (`MCDFA`) from a synchronous, deterministic, and complete finite automaton (`CDFA`).

Notice that there is no verification if the automaton is already minimal.

Your program must display successive partitions, as well as transitions expressed in terms of parts, all the way along the minimization process. Make that display easily readable.

`display_minimal_automaton(MCDFA)`

Displaying the minimal automaton using the same form as before.

Your program must explicitly inform the user to which state(s) of the initial automaton corresponds each state of the minimal automaton. This correspondence may be directly used in the transition table, or, if you opt for

renaming the states of the minimal automaton, there must be a separate "table of correspondence".

Word recognition

Your program should now analyze words typed on the keyboard by the user. It **must** allow the user to test several words one after the other, without having to re-select the word recognition operation.

Pseudo code

```
read_word ( word)
WHILE word ≠ « end » DO
     recognize_word ( word, A )
     read_word ( word)
REDO
```

where:

`read_word ( word)`

storing in memory a string of characters typed by the user on the keyboard.

Warning:

-You must absolutely give the user a way to type in an empty word.

-The word is fully read as a string before it is tested. There should be absolutely no letter-by-letter reading and testing.

-You must tell the user what to type in order to signal that he or she wants to stop reading words. The pseudo code above proposes the string "end", but you may choose otherwise.

```
recognize_word ( word, A )
```

Verifying if the automaton A recognizes the word 'word'.

Result: Yes or No.

Option: informing the user which is the first character of the word that prevents it from being recognized.

Since all automata your program has dealt with so far are equivalent, you can test the word recognition using any kind of automaton (non-deterministic, just deterministic, deterministic and complete, minimal). It is, however, easier to use a deterministic automaton for that purpose than a non-deterministic one. If you opt for using a complete deterministic automaton, you may freely choose between a non-minimized or a minimized automaton.

Complementary Language

Your program must construct a FA that recognizes the language complementary to that recognized by a given FA, and test which words are recognized by this complementary FA.

Pseudo code:

```
AComp ← complementary_automaton ( A )
display-automaton ( AComp )
read_word (word)
WHILE word ≠ "end" DO
      recognize_word ( word, AComp )
      read_word (word)
REDO
```

where:

`complementary_automaton ( A )`

> Constructing an automaton recognizing the complementary language.

> The entry parameter A can be, at your choice, the previously obtained CDFA or MCDFA. Your program must tell the user which automaton has been used to obtain a complementary.

Standardization

Your program must standardize the automaton obtained at the previous stage, and then test the word recognition by the standardized automaton.

Pseudo code:

```
ACompStd ← standard_automaton ( A )
display_automaton ( ACompStd )
read_word ( word)
WHILE word ≠ "end" DO
      recognize_word (word , ACompStd )
      read_word ( word)
REDO
```

where:

`standard_automaton ( A )`

> Standardization of the previously obtained automaton 'A'.

> If the automaton A is already standard, your program must inform the user about it and it should not modify it.

Operation sequence

If you have succeeded in building all the code described above, the following sequence of automata will be obtained:

FA → CDFA → MCDFA → AComp → ACompStd

IF some of the stages have not been done successfully, you may obtain different sequences. For example:

Minimization not done:

FA → CDFA → AComp → ACompStd

Minimization and complementarizaion not done:

FA → CDFA → AStd

In that case, the standard automaton will be obtained from a complete deterministic automaton, not minimized, not complementarized.

Evidently, you'll have to clearly announce during your presentation what exactly your program is capable of doing.

The multiple automata loop

You must insert all the described operations in a general loop that allows the user to deal with several finite automata without having to restart the program.

Programming environnement

You can use the languages such as C, C++, Python, or Java.

Your program must be compilable and executable on my PC. I'll tell you the software versions I will have installed. (You may give me advice if you wish).

Automata your program must be able to deal with:

The tests will be done on automata:

- dealing with the alphabet consisting of characters from 'a' to 'z':

for example, an automaton whose alphabet consists of 3 symbols will use the characters 'a', 'b', and 'c';

- whose states are labeled by numbers starting from '0':

for example, an automaton consisting of 5 states will have the states labeled from 0 to 4, without breaking the sequence.

There is no limit on the number of states a test automaton may contain.

The text file representing an automaton read by your program may have the following syntax (but you may prefer a slightly different structure as well):

Line 1: number of symbols in the automaton's alphabet.

Line 2: number of states.

Line 3: number of initial states, followed by their numeric labels.

Line 4: number of final states, followed by their numeric labels.

Line 5: number of transitions.
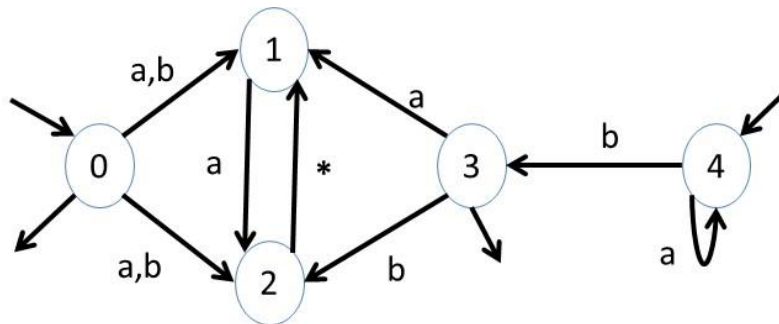
Lines 6 and the following lines: transitions in the form

<source state><symbol><target state >

For an asynchronous automaton, an epsilon-transition can be represented by the symbol "*".

For example:

Automate: File:

2
5
2 0 4
2 0 3
10
0a1
0a2
0b1
0b2
1a2
2*1
3a1
3b2
4b3
4a4

2 symbols in the alphabet

> A={a,b}

5 states

> Q={0,1,2,3,4}

2 initial states

> I={0,4}

2 final states

> T={0,3}

10 transitions (of which one is an epsilon-transition)

Technicalities

The teams

> Group Int1 consists of 32 students, and group Int2, of 33. This means that group Int1 will form 8 teams of 4, and group Int2, 7 teams of 4 and one team of 5. I'll tell you by which date the teams must be formed.

Tests (preparing the presentations)

Some time in the future you'll be given test automata you'll have to use during your presentation.

At the presentation, all the text files corresponding to test automata must be present on your computer and accessible to the program. It's your responsibility to ensure that your PC works well and that your program can be compiled and/or executed on your PC.

There will be no computer provided by EFREI.

Sending your project

The rules of how to send and what to send will be given you slightly later. Same for the deadline. (The deadline will be the same for all groups, be it the French or English groups).

Presentation

Time slots: 45 minutes per team.

Deadline for choosing the time slots for each team will be given to you later.

The presentation sequence: an oral presentation, demonstration, a possible additional discussion.

Oral presentation

length: 15 minutes.

Each team member will present a part of the total, without any help from the others. Make sure you know who presents what!

The presentation times of each team member, as well as the relative difficulties of the operations presented by each team member, must be comparable to each other. It's your responsibility to ensure that is the case.

You must prepare a written presentation support on paper (to be physically given to me – not as a file! - in the very beginning of your presentation).

Your data structures and algorithms must be clearly described in that support.

Advice: *prefer a clear and precise diagram or a well-structured pseudo-code that you will comment on, rather than a long text that cannot be read during the presentation! Your data structures and algorithms must be absolutely clear, and you must not use the C or C++ code directly for the presentation. Be complete and precise: saying, for example, that an automaton is represented by an object class is not enough if we do not know more precisely the data structures used and the functions allowing their manipulation; saying that it is an array is not enough either if we do not know what the cells of the array contain...* **Explaining a data structure means making people understand without having to ask additional questions about how a FA is represented in the machine memory**. *It is also necessary to spontaneously specify, without waiting for a request, if you are using the same data structure or two different data structures for non-deterministic and deterministic automata.*

Warning:

It is advisable to prepare a Powerpoint presentation. Rooms are usually equipped with a video projector or with a TV screen. If this is not the case, a laptop with a sufficiently readable screen can usually suffice.

Demonstration

Put all the text files representing the test automata in your working environment **before** the presentation.

You will compile your program (which must be exactly what you send to me, without any last-minute corrections or additions) in my presence, then you'll launch execution. I will choose the automaton to work on, and then I will test the recognition of some words. I will also have the right to suggest a little change in an existing automaton, that we'll save, read and test; or even an entirely new automaton, easy enough to type as text in a file.

Discussion / Questions and Answers

Naturally, I can ask you questions at any moment. I can also ask a specific person to answer, without any help from the others. You have to understand that even though you may not directly participate in work on all components of the program (which is perfectly normal for a team work), each must be capable to answer any sufficiently general question about the project. It is advisable for each team to meet regularly, and during those meetings you should explain to each other the work each has done.

Besides giving to each of you the information necessary for the presentation, this will also force each of you to learn how to implement all algorithms you will have learned in this course.

Some grading tips

Naturally, the grade will depend on the functioning of your program as well as on the quality of your presentation. The following elements will also influence on the grade:

- Clarity of the man-machine interface, allowing for a smooth testing of your program and for an easy and versatile demonstration (sequence of actions, execution trace, …). It should be easy for someone not having participated in the work on the program to check the results.
- The choice and the justification of the of data structure(s) representing automata;
- The readability of the code. This includes commenting. The absence of comments is usually perceived, except rare cases of extremely well written code, as a badly readable code.

Each team will get a common grade, but this grade may be raised or lowered individually depending on that person's oral presentation and/or answers to questions.