

Didactic Approach for Teaching Nondeterminism in Automata Theory

György Maróti, Szeged (Hungary)

Abstract: Nondeterminism plays a central role in almost all fields of computer science. It has been incorporated naturally as well as in the theory of automata as a generalization of determinism. Although nondeterministic finite automata do not have more recognition power than deterministic ones their importance and usefulness is of no doubt.

Unfortunately, the operation of the mathematical model of nondeterministic automata is difficult to understand which means a didactic challenge for every teacher and lecturer. This paper gives a didactic approach to introduce the notion of finite (deterministic and) nondeterministic automata.

As a teaching tool we make use of the automata theory package developed in Maple. We put emphasis on the process character of learning and work out a method that promotes the repetitive experiments.

Kurzreferat: Didaktische Betrachtung des Lehrens von Nicht--Determinismus in der Automatentheorie. Der Nicht-Determinismus spielt in fast allen Bereichen der Informatik eine bedeutende Rolle. Dieser Begriff wird auch in der Automatentheorie verwendet als Generalisierung des Determinismus. Obwohl nichtdeterministische endliche Automaten nicht über mehr Erkennungskapazität verfügen, als deterministische, steht ihre Wichtigkeit und Nützlichkeit ausser Frage.

Leider ist die Funktion des mathematischen Modells von nichtdeterministischen Automaten nicht leicht zu verstehen, deshalb ist es eine Herausforderung für jeden Lehrer. Dieser Artikel soll eine didaktische Betrachtung des Begriffs von endlichen deterministischen und nichtdeterministischen Automaten geben.

Wir benutzen als Unterrichtshilfe das in Maple entwickelte Programmpaket für Automatentheorie. Wir betonen den kontinuierlichen Charakter des Lernens und entwickeln eine Methode, die das wiederholte Experimentieren unterstützt.

ZDM-Classification: C20, D40, H70, P20

1. Introduction

The basic notions of automata theory are suitable to illustrate the usage of didactic rules. This paper offers a didactic approach to introduce the mathematical concept of finite state machines. We start from the real world, namely from intuitive notion of deterministic finite state machines (Archimedes Rule, see [6]) which proves to be enough for the implementation and the introduction of two representations of finite automata: transition table and transition graph (Rule of Four, see [6]). At this phase we use different procedures from Maple's automata theory package, but we use them as black boxes. We do not care how these procedures work. It is sufficient to be aware of their usage, as we want to use them only as experimental tools.

The didactic goal is to highlight the process of abstraction. In the course of building mathematical models we have to find pure mathematical tools which are suitable to describe the constituent parts and the

operation of sequential machines. We use the existing implementation and gather experiences about how the generalized transition function works. We realize that the composition of functions is the suitable means to model real world things like input tape and reading head. We explore all fundamental features of generalized transition function and this leads us to formulate the definition of deterministic automaton, the generalized transition function and the concept of word and language recognition. By the end of this learning process we completely understand the behaviors of the implementation which becomes white box from the initial black box.

Next we continue the process of abstraction. The first level abstraction built the concept of a deterministic automaton from the intuitive notion of sequential machines. The second level abstraction is used to build the notion of a nondeterministic automaton from a deterministic one. As before we use the inductive method again: experiment, formulate the results of observations, generalize and introduce the new concepts.

Our starting point is the transition graph of deterministic automata, which have a certain special property due to determinism. We ask what would happen if we drop this property and tried to consider every directed-labeled graph as the transition function of an "imaginary automaton". We completely explore the consequences of this assumption. First we realize that the concept of transition function has to be changed. This change yields, however, that we have to introduce new notions such as the set of runs instead of run or the current set of possible states instead of current state when processing an input word. We discover another useful representation of processing the input word called the tree of possible runs. Similarly to the deterministic case by the end of our experiments, we know everything to formulate the definition of (nondeterministic) automaton. From didactic point of view we make the black box of nondeterministic generalized transition function to be white box.

Learning is a process, which runs in the course of time with stops, stepping back, feedback and experimental stages. Experiments play fundamental role in observation, in gathering experiences. The most difficult question is how much time has to be spent for a certain phenomenon to experiment with it. In other words how much time has an experiment to be repeated, so that every student in the class be able to draw up the results of experiment and then suspect and formulate the general rule behind it. Student's comprehension in a class may differ significantly.

Our answer for this challenge is based on the use of CAS. Maple, like every interactive CAS, allows users to repeatedly execute the same sequence of commands, while user can change the value of any data taking part in the calculation. What we have to do is to draw student's attention to the beginning and ending point of the sequence of consecutive Maple commands which constitutes an experiment. If we organize our Maple worksheet in such a way that the first command of the sequence always contains only the data which the experiment depends on then we obtain a **Point of**

Practice (PoP) in the course of learning process. PoPs are used to indicate the beginning of an experiment and help student to find out which data have to be changed to obtain different output of the experiment. The last command of the experiment is called as **End of PoP** and denoted by EoP.

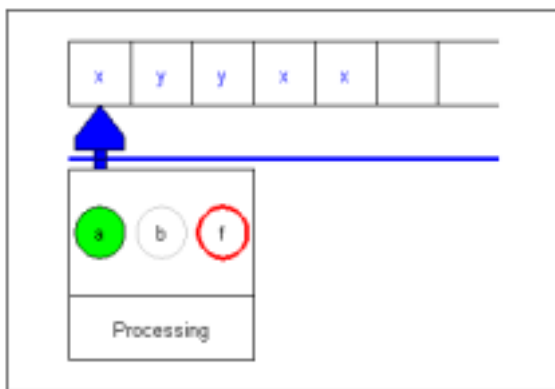
In this way out teaching material may contain several PoP-EoP pairs each of which designate the first and the last command of an experiment, whose input data can be found in the PoP command. The sequence of commands can be repeated by the students as many time as desired. This number may vary student by student.

2. Deterministic case

2.1 Motivation

Intuitively, we think of a deterministic **automaton** as a sequential **finite state** machine, which has an input tape and a reading head. The **input tape** holds the input word, which is read by the input head from left to right one character at a time. The automaton begins its operation in its **initial state**, reads the first **input signal** and changes its state. This change is determined by the **transition function**. Then the automaton reads the next input signal and changes its state again, and so on till the end of the input word. The three-step event, which consists of reading the current input signal, changing the state and move the input head to the right, is called **transition**. This means that the operation of automaton is nothing else than a sequence of transitions. If the state to which the automaton moves from the initial state when reading the input word W is a **final state** then we say the automata **accepts or recognizes** the word W .

```
> showaut(genaut([a,x,b,a,y,f,b,y,a,\
b,x,f,f,y,a,f,x,b]),xyyxx);
Finite state automaton
```



2.2 Implementation

We use the intuitive notion of automaton to implement it in Maple. First of all an automaton possesses finitely many states. These states form a finite set, which we can call the **set of states**. As Maple handles the set data structure we easily define a set of state S consisting of three states a, b and f .

```
> S := {a, b, f};
```

$$S := \{f, a, b\}$$

The input tape of automaton holds the **input word**,

which is nothing else than a sequence of **input signals**. Collecting all possible signals we obtain a set, which we call **alphabet**. Let us denote the alphabet by X .

```
> X := {x, y};
```

$$X := \{x, y\}$$

Notice the intuitive notion tells nothing about the cardinality of input signals but we feel sure this set must also be finite. It would be strange to call something finite if one of its components is infinite.

Next we implement the transition function, which we denote by δ . This function determines the next state to which the automaton moves from the current state in response to reading current input signal. As we see δ works on pairs of the form (s, ξ) where s is a state and ξ is an input signal. The value $\delta(s, \xi)$ is the next current state.

```
> delta[a,x] := {b} : delta[a,y] := {a} :
delta[b,x] := {f} : delta[b,y] := {a} :
delta[f,x] := {f} : delta[f,y] := {f} :
```

Commands $\text{delta}[a,x] := \{b\}$ and $\text{delta}[a,y] := \{a\}$ prescribe that the input signal x moves the automaton from state a to state b while the signal y leaves the state a unchanged. The other commands are to be interpreted similarly.

Note, furthermore, that we use Maple's table data structure to implement the transition function. In the sequel we do not distinguish the transition function from its implementation. Although Maple requires the syntax $\text{delta}[s,\xi]$ in the commands we often write $\delta(s, \xi)$ instead.

Returning to the implementation the automaton must have an initial state and a set of final states. Like in the specification of delta we use one element set $Q = \{a\}$ instead of the single state a . The set of final states is denoted by R .

```
> Q := {a};
```

$$Q := \{a\}$$

```
> F := {b, f};
```

$$F := \{f, b\}$$

We finish the implementation of automaton by collecting all defined components in a five-element list...

```
> Xi := [S, X, delta, Q, F];
```

$$\Xi := [\{a, b, f\}, \{x, y\}, \delta, \{a\}, \{b, f\}]$$

... which is recognized by Maple as an object of type **automaton**.

```
> type(Xi, automaton);
```

true

Unfortunately the list

$$[\{f, a, b\}, \{x, y\}, \delta, \{a\}, \{f, b\}]$$

tells nothing about the behavior of transition function except its name. What we need is a better representation of automata.

The procedure **printaut** is used to display the **transition table** (see [5]) of automata. We can see the states in the first column whilst the input signals can be seen in the first row of transition table. The intersection of row labeled by the state s and the column labeled by the input signal ξ contains the state $\delta(s, \xi)$.

```
> printaut(Xi);
```

$$\begin{bmatrix} STA- INP & x & y \\ a & b & a \\ b & f & a \\ f & f & f \end{bmatrix}$$

Initial state: , a

Set of final states: , { f, b }

Using the transition table we can easily describe the different transitions of automata. For example if the automaton Ξ is in the state b and reads the input signal y then the next state is $\delta(b, y) = a$. Observe that this equality itself describes the transition and because δ is defined for all states and input signals we can determine the next state for every state and input signal.

2.3 Runs

Notice we have lost input tape and reading head during the implementation process. Indeed, all components of the implemented automaton are mathematical notions: set, table and list. Implementation requires abstraction.

On the other hand if we do not have input tape and reading head we have to find mathematical means with which we can describe sequences of transitions.

Imagine our automaton Ξ wants to process the input word xyy . At the beginning Ξ is in state a (the initial state) and reads the first signal of w , which is x . The next state is determined by δ , namely

$$\text{Transition 1 : } \delta(a, x) = b .$$

Therefore the automaton is now in the state b and reads the second signal, which also coincides with x .

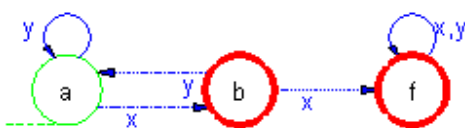
$$\text{Transition 2 : } \delta(b, x) = f .$$

In the end the automaton reads the last signal y while the current state is f .

$$\text{Transition 3 : } \delta(f, y) = f .$$

This process can be observed very well in the **transition graph** (see [5]) of automaton Ξ , which can be created by procedure **plotaut**.

```
> plotaut(Xi, line);
```



As we can see the transition graph is directed graph whose nodes represent the states while the labeled edges represent the transitions of automata.

Take the node of the transition graph and follow the

arrow, which is labeled by x . We reach the state b , which indicated that the new current state is b . From this state we follow the arrow again labeled by x we reach the state f , which remains unchanged when the automaton reads y , as the arrow labeled by y is a loop. As we see the transition graph offers clear graphical representation of the sequence of transitions.

Let us return to the algebraic representation, namely to the qualities above, which describes three consecutive transitions. Note we can substitute the left-hand side of second equation for f in the third equation. We obtain $\delta(\delta(b, x), y) = f$. Next substituting the left hand side of first equality for b we get

$$\delta(\delta(\delta(a, x), x), y) = f$$

What we see on the left-hand side is the composition of the transition function by itself. The state $\delta(\delta(\delta(a, x), x), y)$ is nothing else than the state to which automaton moves in the course of reading the signals of input word xyy . Similarly, if we are interested in the state to which automaton Ξ moves when reading the word yxx , we have to determine the state $\delta(\delta(a, y), x)$.

At this point we have found the mathematical tool, which is capable to describe the sequence of transitions, and this is the composition of functions.

For the sake of simplicity we introduce a new function Δ which works on input words instead of input signals. More specifically Δ determines the last state to which the automaton moves from the current state when reading the input word w . For example if $w = xyy$ then

$$\begin{aligned} \Delta(a, xyy) &= \delta(\delta(\delta(a, x), x), y) = \\ &= \delta(\delta(b, x), y) = \delta(f, y) = f \end{aligned}$$

This concept is implemented by procedure **Delta** in Maple. The first parameter of Delta is an automaton the second is a state while the third parameter is an input word. Let see how Delta works.

```
> w := xxy; # POP
```

$$w := xxy$$

```
> Delta(Xi, a, w);
```

$$f$$

Delta calculated the current state but told nothing about the details of calculation. If we, however, use the option **trace1** then Delta displays all intermediate states, which are touched during calculation of the state to which the automaton moves.

```
> Delta(Xi, a, w, trace1):
```

```
-Delta begins with a
```

```
-Delta(a, x) = b
```

```
-Delta(b, x) = f
```

```
-Delta(f, y) = f
```

$$\Delta(a, xyy) = f$$

The list of states touched during processing the input word w is called the **run corresponding the input word**

w (see [5]). For example the list $[a, b, f, f]$ is a run of automaton Ξ , which corresponds to the input word $w = \text{xyx}$. Note the last component of the run corresponding to w is nothing else than the state to which the automaton moves in response of w .

If we use the option **runs** procedure Delta returns the run corresponding to the input word given as its third argument.

```
> Delta(Xi, a, w, runs); # EOP
      {[a,b,f,f]}
```

Take two input words u and v . $\Delta(s, u)$ is the state to which the automaton is moved by u from state s . Denote this state by \hat{s} , and take the state $\Delta(\hat{s}, v)$. Substituting \hat{s} by $\Delta(s, u)$ we obtain $\Delta(\Delta(s, u), v)$. How has the automaton reached this state? It started in the state s processed the word u and reached an intermediate state. From this state the automaton processed the word v . What our automaton has done is nothing else than the process of the word $w = uv$. We formulate our observation as

$$\Delta(s, uv) = \Delta(\Delta(s, u), v)$$

This equality, which is a simple consequence of the associative feature of composition of function, expresses the fact that if we divide the input word into two subwords, then we can perform processing of the input word in two steps. We can start processing the first subword and next from that state to which the automaton has moved in response of first subword we process the second subword. (What is the connection between runs corresponding to u and v , respectively, and the run that corresponds to the input word $w = uv$?)

```
> u:=yx:v:=xx: # P OP
      Delta(Xi, Delta(Xi, a, u, 1), v, 1):
      -Delta begins with a
      -Delta(a, y) = a
      -Delta(a, x) = b
```

$$\Delta(a, yx) = b$$

```
-Delta begins with b
-Delta(b, x) = f
-Delta(f, x) = f
```

$$\Delta(b, xx) = f$$

```
> w:=cat(u, v):
      Delta(Xi, a, w, 1): # EOP
      -Delta begins with a
      -Delta(a, y) = a
      -Delta(a, x) = b
      -Delta(b, x) = f
      -Delta(f, x) = f
```

$$\Delta(a, yxx) = f$$

Notice Δ can be considered as a generalization of the transition function δ as for single input signals the two functions are identical. Furthermore it is practical to define Δ for empty word in the natural way that the empty word leaves every state unchanged. We formulate these by the following equalities

$$\Delta(s, \xi) = \delta(s, \xi) \text{ and}$$

$$\Delta(s, \varepsilon) = s,$$

where ε denotes the empty word.

2.4 Definitions

At this point we have already prepared everything to be able to introduce the exact mathematical definition of automata and the function Δ , the generalization of the transition function, which turned to be adequate tool to describe the operation of automata.

Definition 1

An **automaton** is a five-tuple $\Xi = [S, X, \delta, Q, F]$, where

S is a finite nonempty set. The elements of S are called the **states of automaton** Ξ .

X is a finite nonempty set called **alphabet**. The elements of X are called the **input signals of automaton** Ξ .

δ is a function $S \times X \rightarrow S$, the **transition function of automaton** Ξ .

Q is a one element subset of S which contains the **initial state**.

F is an (empty or nonempty) subset of S , the set of **final states**.

Definition 2

The **generalized transition function** $\Delta: S \times X^* \rightarrow P(S)$ is defined by the following recursive definition. For every state s , every input word w and input signal ξ

$$\Delta(s, \varepsilon) = s \text{ and}$$

$$\Delta(s, w\xi) = \delta(\Delta(s, w), \xi).$$

Definition 3

We say automaton Ξ accepts or recognizes an input word w if $\Delta(q, w) \in F$, where q is the initial state ($Q = \{q\}$). (See [3], [4] and [5])

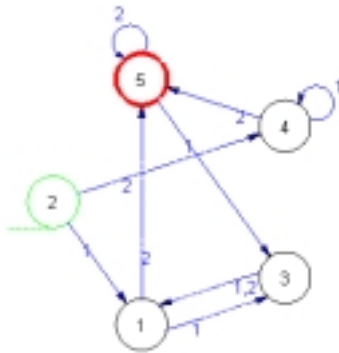
3. Nondeterministic case

In the previous section we developed the mathematical model of finite deterministic sequential machines. Now we continue the process of abstraction and in this section we introduce the notion of nondeterministic automaton.

3.1 Motivation

As we have already seen before deterministic automata can be represented by labeled directed graph. Next command displays a random five-element automaton whose input alphabet is the set $\{1, 2\}$.

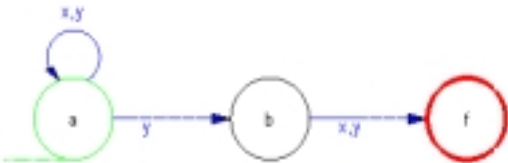
```
> numberofstates:=5:
      numberofsignals:=2: #P OP
> plotaut(randaut(5, 2,
      deterministic)); #EOP
```



Observe that this graph has a special property. Namely, for every s state and every input signal ξ there exists exactly one arrow which is labeled ξ and whose starting point is s . Indeed, determinism yields that at most one arrow exists while the completeness of the transition function provides at least one arrow.

What if we omit this limitation and want to consider every labeled directed graph as a transition graph of certain automata? Consider for example the graph in the figure below.

```
>Phi:=genaut([a,{x,y},a,a,y,b,\
              b,{x,y},f]):
plotaut(Phi,line);
```



This graph owns all the basic properties of transition graphs. Indeed, nodes are labeled by states, directed arrows are labeled by input signals. What is wrong in this graph is the existence of

- undefined transitions (for example $\delta(f, x)$ is not defined) and
- nondeterministic transitions (for example $\delta(a, y)$ is not uniquely defined).

Temporarily disregard the fact that this graph is not the transition graph of any (deterministic) automaton and let us play with this graph as if it was the transition graph of an "imaginary automaton". We try to find answers for the following questions.

- What is the effect of this assumption for the definition of automata?
- If we can find an appropriate new definition then how can we describe the operation?
- How can we define the word (and language) recognition by automata satisfying the new definition?

3.2 The nondeterministic transition function

Two things remain surely unchanged. The set of states S and the set of input signal X . On the same time it is clear that we have to change the definition of the transition function.

Because of nondeterministic transitions in the graph the new transition function has to map the pairs the form (s, ξ) into the set $P(S)$, the powerset of S .

Furthermore, as we want to allow undefined transitions as well, δ must be partial function. Summarizing, the new concept of the transition function is that δ is a partial function $S \times X \rightarrow P(S)$.

This change in the definition of δ seems to be sufficient. For example if we define

$$\begin{aligned} \delta(a, x) &= \{a\} \\ \delta(a, y) &= \{a, b\} \\ \delta(b, x) &= \{f\} \\ \delta(b, y) &= \{f\} \end{aligned}$$

then using the process of drawing the transition graph as we did in the deterministic case we obtain the original directed labeled graph.

How can we describe the operation of automata with this concept of transition function?

3.3 Set of possible runs instead of run

Consider the input word yx and suppose the current state is a . As there are two arrows labeled by y the new state can be either a or b . It seems our imaginary automaton has got two possibilities. If it chooses state a then the second signal x takes it to state b . In this case we obtain a **possible run** $[a, a, a]$. If, however, the automaton chooses the state b in the first transition the second transition moves it to f . So we obtain another possible run $[a, b, f]$.

Our first observation is that because of the multiple labeled arrows the operation of our imaginary automaton becomes **nondeterministic**. It must make choices from the possible transitions, but there is no way to determine which one. The only thing what we can do is to collect all possible runs as the next command shows.

```
> w:=yx; # PoP
w:=yx
>Delta(Phi,a,yx,runs); # EoP
{[a,a,a],[a,b,f]}
```

Remember, in case of deterministic automaton we always get exactly one run corresponding to an input word. Now this property has changed. Using our new concept of transition function we obtain a **set of possible runs** corresponding to an input word.

What are the effects of the existence of undefined transitions? Suppose the current state is b and let the input word be xy . The first transition is deterministic, the new current state is f . Now our automaton reads input signal y but no transition is defined. The operation is interrupted. We have got no valid run.

```
>Delta(Phi,b,xy,runs);
{}
```

The set of runs corresponding to xy is empty.

As for the word recognition it is easy to find appropriate definition. We say a word w is recognized if

at least one of the possible runs corresponding to w ends with final state. Notice, if we apply this definition to deterministic automaton we reach the original concept of recognition.

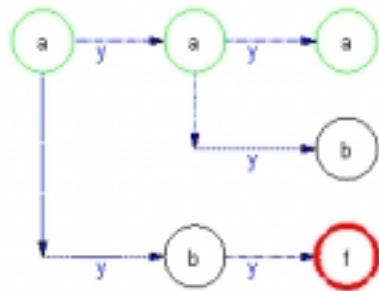
3.4 Current set of possible states instead of current state

Consider now the input word $w = yy$ and create the set of runs corresponding to w .

```
> w:=yy: # POP
> Runs:=Delta(Phi,a,w,runs);
Runs := {[a,a,a],[a,a,b],[a,b,f]}
```

We can represent these runs by a tree an acyclic directed graph.

```
> plotaut(Phi,w);
```



Observe that this representation nicely shows the possible behaviors of the automaton in response to the input word $w = yy$. We always have a branch in this tree if a transition is not unique. The different paths from the root of this tree represent different runs corresponding to the input word w . This graph is called the **tree of runs (ToR)** corresponding to the input word w .

Collect the set of states, which have the same distance from the root of ToR.

```
> L:=[{}$length(w)+1];
for xr in Runs do
for xi to nops(xr) do
L[xi]:=L[xi] union {xr[xi]}
od:
od:
L;
[[{a},{a,b},{f,a,b}]
```

Notice, the second set in this list $\{a,b\}$ consists of the states, which can be reached from the state a using the first input signal y . Likewise the third set $\{a,b,f\}$ consists of the states which can be reached from states belonging to the set $\{a,b\}$ using the second input signal. This can be expressed formally by the following equalities

$$\{a,b\} = \delta(a, y)$$

$$\{a,b,f\} = \delta(a, y) \cup \delta(b, y)$$

The set $\{a,b\}$ is nothing else then the **set of possible states (SoPS)** which can be chosen by the automaton if its current state is a and it reads the first input signal

y . Likewise, the set $\{a,b,f\}$ is the SoPS, which can be chosen by the automaton if the current state belongs to $\{a,b\}$ and the automaton reads the second signal y .

In this way we have obtained a new representation of the operation. The first SoPS is set of initial states $S1 = Q$. The automaton reads the first input signal $x1$. We determine the second SoPS

$$S2 = \bigcup_{s \in S1} \delta(s, x1)$$

Next the automaton reads the second signal and we repeat the process.

In deterministic case we introduced the function Δ to formally describe the operation of automata. This function worked on states and input words. Now he have to generalize this function is such a way that Δ has to be able to work on subsets of states and input words.

This process is shown by the output of next command

```
> Delta(Phi, {a}, w, trace2); # POP-EOP
-Delta begins with {a}
.a,y->{a, b}
-Delta({a},y) = {a, b}
.a,y->{a, b}
.b,y->{f}
-Delta({a, b},y) = {f, a, b}
Delta({a}, yy) = {f, a, b}
```

$$\{f, a, b\}$$

Note, in case of deterministic automaton every SoPS is one element set.

4. The definition of automata

Definition 1

A five-tuple $\Xi = [S, X, \delta, Q, F]$ is called **automaton** where

S is a finite nonempty set. The elements of S are called the **states of automaton** Ξ .

X is a finite nonempty set called **alphabet**. The elements of X are called the **input signals of automaton** Ξ .

δ is a partial function $S \times X \rightarrow P(S)$, the **transition function of automaton** Ξ .

Q is a one element subset of S which contains the **initial state**.

F is an (empty or nonempty) subset of S , the set of **final states**.

Note that our notion of automaton is **nondeterministic**. Indeed, $\delta(s, \xi)$ equals to a set of certain states instead of one well defined state.

Note if $|Q|=1$ and for every state s and input signal ξ the set $\delta(s, \xi)$ has exactly on element then we obtain the concept of deterministic automaton.

Definition 2

The **generalized transition function** $\Delta : P(S) \times X^* \rightarrow P(S)$ is defined by the following

recursive definition. For every $A \subseteq S$, every input word w and input signal ξ

$$\Delta(A, \epsilon) = A$$

$$\Delta(A, \xi) = \bigcup_{s \in A} \delta(s, \xi)$$

$$\Delta(A, w\xi) = \Delta(\Delta(A, w), \xi)$$

This concept of Δ coincides with that what we introduced in deterministic case if $|A| = 1$.

Definition 3

We say an input word w is recognized (or accepted) by the automaton Ξ if $\Delta(Q, w) \cap F \neq \{\}$. The set of all recognized words is denoted by $L(\Xi)$, which is called the language recognized by Ξ (see [3], [4] and [5]).

5. Determinism versus nondeterminism

Automata are tools to recognize words and languages. As deterministic automata are special nondeterministic automata the class of languages recognized by deterministic automata is a subclass of the languages recognized by nondeterministic automata.

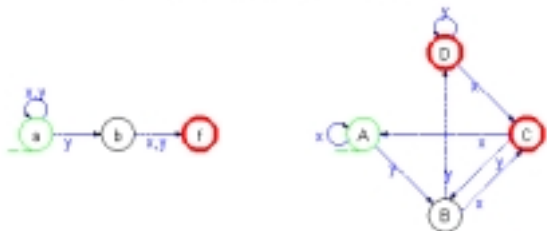
One of the basic results of automata theory is the recognition power of nondeterministic automata is not stronger than that of deterministic automatata. More specifically for every nondeterministic automaton there exists a deterministic automaton, which recognizes the same language (see [3], [4] and [5]).

In spite of this fact we need nondeterministic automata for several reasons. Nondeterministic automata are exponentially more succinct.

Next figure shows the transition graph of a nondeterministic and the equivalent deterministic automata.

```
> Theta:=renaut(ConDet(Phi),sta=A):
p1:=plotaut(Phi,[-3,0],line):
p2:=plotaut(Theta,[2,0]):
plots[display]({p1,p2},
scaling=unconstrained,
title="Nondeterministic and
determinisitic automata");
```

Nondeterministic and determinisitic Automata



Looking at the transition graph of the above nondeterministic automaton we at once see that the words recognized by it are of the form $py\xi$ where p is an arbitrary input word while ξ is an arbitrary input signal. Although the deterministic automaton recognizes the same set of words this statement is far from being obvious if we work with its transition graph.

In general nondeterministic automata possesses less state and less transitions, which makes the work with

them much easier. Furthermore the way nondeterministic automata work is very close to human thinking. When constructing different automaton, which performs a certain task or recognizes a certain language we almost always reach nondeterministic automaton.

On the other hand deterministic automata can be considered as algebraic structures and can be handled by very effective algebraic tools. Most of the results in the theory of automata have been worked for deterministic automata.

6. Motivated mathematics

The didactic rules and recommendations which are the results of didactic research concerning the usage of CAS in the education has already been applied in several fields of mathematics such as calculus, linear algebra or geometry (see [2],[6],[9]). In this paper a new field, the theory of automata has been involved in this process.

We have seen that the didactic rules can rear be applied chemically purely. More or less modifications are often inevitable. For example in case of investigating automata we can not speak about numerical representation, as this is not applicable. The lack of numerical representation, however, does not contradict to Rule of Four.

We emphasize the process character of learning. This process runs in course of time, possesses a starting point, one or more ending points, branches and practice stages. CAS systems like Maple are excellent tools to make practicing easy. Suitably organizing the teaching material we can specify several PoP-EoP pairs, which specify repetable experiments. The most important feature of PoP-EoP pairs is that the sequence of commands belonging to it can be repeated as many time as required. In this way both the teacher and the students are allowed to execute arbitrary number of experiments and check the results while changing the values of input parameters.

Karl Joseph Fuchs pointed out the importance joyful mathematics (see [2]). We prefer to speak about **motivated mathematics**, which should be a "guided tour". The teacher, who is aware of the teaching goals, is responsible for the choice of one of possible ways for reaching these goals. Every step of this tour must be clearly explained and motivated. We are convinced if these requirements are fulfilled the resulting mathematics will be enjoyable too.

References

- [1] A.V. Aho, J.E. Hopcroft & J.D. Ullman: Data structures and algorithms. - Addison-Wesley, Reading, Mass. 1983
- [2] K.J.Fuchs, Computer Algebra Systems in Mathematic Education, International Symposium - Anniversary of Pollack Mihály College of Engineering, 2002
- [3] F. Gécseg & I. Peák: Algebraic theory of automata. - Akadémiai Kiadó, Budapest 1972.
- [4] J. E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison- Wesley, Reading, MA, 1979.
- [5] Kozen: Automata and Computability. - Springer-Verlag, New York, 1997.
- [6] W. Lindler, CAS-Supported Multiple Representations in the Elementary Linear Algebra, The case of Gaussian Algorithm, International Symposium - Anniversary of Pollack Mihály

College of Engineering, 2002

- [7] M.B. Monagan, K.O. Geddes, K.M. Heal, G.Labahn, S.M.Vorkoetter, J. McCaron. P.DeMarco: Maple 8 Introductory Programming Guide -Waterloo Maple , 2002
- [8] M.B. Monagan, K.O. Geddes, K.M. Heal, G.Labahn, S.M.Vorkoetter, J. McCaron. P.DeMarco: Maple 8 Advanced Programming Guide -Waterloo Maple , 2002
- [9] O. Wurnig, From th first use of the computer up to the integratin of DERIVE in the teaching of mathematics, IGJ Vol 3, No.1. p.11-24, 1996

Autor

György Maroti, – Epigramma Kft.
Szent-Györgyi Albert u. 2., H-6726
E-mail: maroti@epigramma.hu